Systèmes à composants synchronisés : contributions à la vérification compositionnelle du raffinement et des propriétés

Arnaud Lanoix

N° d'ordre 1083

THÈSE

présentée à

L'U.F.R. DES SCIENCES ET TECHNIQUES
DE L'UNIVERSITÉ DE FRANCHE-COMTÉ

pour obtenir le

GRADE DE DOCTEUR DE L'UNIVERSITÉ DE FRANCHE-COMTÉ
Spécialité : Informatique

Systèmes à composants synchronisés : contributions à la vérification compositionnelle du raffinement et des propriétés

par **Arnaud Lanoix**

Soutenue le 31 août 2005 devant la Commission d'Examen :

Directeurs	O. KOUCHNARENKO	HDR, Maître de conférences à l'Université de Franche-Comté, LIFC
	J. JULLIAND	Professeur à l'Université de Franche-Comté, LIFC
Rapporteurs	S. BENSALEM	Professeur à l'Université Joseph Fourier, VERIMAG
	J. SOUQUIERES	Professeur à l'Université Nancy 2, LORIA
Examinateurs	F. BELLEGARDE	Professeur à l'Université de Franche-Comté, LIFC
	F. LANG	Chargé de recherche, INRIA Rhône-Alpes

Présentation

Ces travaux de thèse proposent une approche pour la vérification des systèmes combinant spécification par composants et raffinement. L'objectif est de résoudre – au moins partiellement – le problème de l'explosion combinatoire lors de la vérification algorithmique des systèmes finis mais de grande taille, qu'il s'agisse de la vérification du raffinement ou de la vérification de propriétés temporelles.

Modéliser un système complexe ne peut s'accomplir ni en une seule étape, ni en un seul "morceau". Généralement, en partant d'une description informelle, on construit la spécification en *ajoutant* de nouveaux éléments et en *réutilisant* des éléments existants en les combinant.

- Une approche classique est de spécifier par raffinement le système : il s'agit d'ajouter des "détails" issus du cahier des charges à chaque étape de la spécification.
- Une autre approche consiste à spécifier le système par composition : les composants sont des éléments réutilisables qui collaborent en se synchronisant.

Des travaux menés précédemment ont permis d'établir une relation de raffinement entre deux systèmes à états finis qui préserve la LTL (Linear Temporal Logic) : une propriété ϕ satisfaite par un système abstrait SA est aussi satisfaite par un système plus concret SR, si SA est raffiné par SR. Le système abstrait est souvent beaucoup plus facile à appréhender que le système raffiné, puisque les détails introduits par le raffinement augmentent la taille du système. Il est donc bien moins coûteux de vérifier ϕ sur SA que sur SR.

Dans le cadre de cette thèse, nous proposons deux approches complémentaires pour la vérification compositionnelle du raffinement et des propriétés.

- La première est basée sur la *décomposition* du système à vérifier. Le raffinement est assuré en étudiant une relation de raffinement *affaiblie* entre chacun des sous-systèmes abstraits et raffinés. Des propriétés peuvent également être vérifiées par décomposition. Des invariants ou des invariants dynamiques sont vérifiés sur chaque sous-système, puis préservés par le système complet. Un algorithme s'assure de manière compositionnelle de l'(in)atteignabilité d'un état en parcourant les différents sous-systèmes. Plus généralement, une propriété LTL de sûreté qui n'est pas satisfaite par l'un des sous-systèmes, ne le sera pas non plus par le système complet.
- La deuxième approche développée dans ce document s'intéresse aux systèmes décrits sous la forme de *composants synchronisés*. Le raffinement d'un système à composants est assuré en adaptant les résultats précédents : la relation de raffinement "affaiblie" est

étudiée entre chaque composant abstrait et raffiné pris dans leur contexte (collaborant dans un contexte de réutilisation particulier). Des propriétés locales à un composant – notamment les invariants et plus généralement toutes les propriétés LTL de sûreté – sont automatiquement satisfaites par le système à composants complet.

L'intérêt de cette démarche est double. Les approches de spécification par décomposition et par composants synchronisés permettent d'étudier le raffinement du système complet "morceaux par morceaux", en diminuant la taille des systèmes à explorer. La vérification compositionnelle de propriétés – sur les composants ou par décomposition – combinée avec la préservation par raffinement des propriétés abstraites permet aussi de réduire l'impact du phénomène d'explosion combinatoire lié aux systèmes de grande taille.

La démarche de vérification du raffinement d'un système à composants synchronisés a été expérimentée sur un prototype appelé SynCo. Cet outil construit la relation de raffinement entre chacun des composants et permet de conclure de manière automatique à propos du raffinement du système complet.

Remerciements

Cette thèse n'aurait jamais pu aboutir sans Olga Kouchnarenko, directrice de thèse, que je remercie pour sa disponiblité, sa rigueur, et la confiance qu'elle m'a toujours témoignée. Merci également à Jacques Julliand, co-directeur de thèse, pour ses conseils, la rigueur ainsi que la précision de ses remarques.

Je tiens à remercier Françoise Bellegarde pour avoir accepté de présider à mon jury mais aussi pour ses nombreuses idées et son soutien permanent.

Merci également à Jeanine Souquières et Saddek Bensalem, pour m'avoir fait l'honneur de rapporter cette thèse, et à Frédéric Lang, d'avoir accepté de participer à mon jury. Leurs remarques m'ont permis d'améliorer la qualité de ce document.

Merci aux étudiants qui m'ont aidé, au travers de différents projets et stages, à implanter SynCo.

Merci aussi à tous les membres du LIFC que j'ai côtoyé tout au long de mes études, d'abord comme étudiant à l'IUP GMI, puis lors de mon DEA et, finalement, pendant mes trois années de thèse.

Merci enfin à ma famille (et tout particulièrement à mes deux soeurs, Lucile et Olympe, et à mon neveu, Alexandre), ainsi qu'à toute ma famille "de coeur" : vous avez tous contribué de près ou de loin à la réussite de cette thèse...

Table des matières

Table des figures

Introduction

Il n'est pas nécessaire de rappeler l'importance qu'ont pris les logiciels informatiques dans le contrôle de systèmes aussi variés que les transports, l'énergie, les télécommunications, l'aéronautique ou encore l'automobile, ni le prix que peuvent coûter les erreurs de spécification, de conception ou d'implémentation de ceux-ci. La moindre défaillance d'un logiciel critique peut entraîner de lourdes pertes humaines ou financières et la destruction ou la perte d'intégrité du système et de son environnement.

En effet, de plus en plus de systèmes automatiques embarquent une composante logicielle significative. La correction d'erreurs découvertes trop tard dans des logiciels diffusés à grande échelle – comme dans des systèmes embarqués dans l'automobile ou les cartes à puces – peut induire des coûts financiers énormes nécessitant le rappel de tous les exemplaires du produit. Les défaillances de systèmes sécuritaires, comme par exemple un système de régulation de trafic ferroviaire ou d'assistance au pilotage, peut avoir des conséquences dramatiques pour les usagers. Or, la production de logiciels *sûrs* demeure une activité complexe, difficile à mettre en oeuvre. Notons également que les temps de validation par les méthodes traditionnelles de test croît exponentiellement avec la complexité des systèmes considérés.

Les méthodes classiques de développement parviennent mal à maîtriser la sûreté de fonctionnement des programmes. Les démarches usuelles doivent évoluer et intégrer dans le cycle de développement des logiciels des techniques rigoureuses liées aux mathématiques et à la logique.

En particulier, l'activité de *vérification* doit prendre de plus en plus d'importance. La vérification intervient en amont de la phase d'implantation d'un logiciel. Étant donnée une *spécification* plus ou moins détaillée du système à réaliser exprimée dans un *modèle* possédant une sémantique formelle il s'agit de vérifier les propriétés comportementales de la spécification afin de corriger le plus tôt possible d'éventuelles erreurs de conception.

Ce n'est que très récemment que le monde industriel s'est ouvert à l'utilisation des méthodes formelles en les intégrant dans le cycle de développement des logiciels. Pour l'activité plus spécifique de la vérification, les techniques de *model-checking* et de *theorem proving* sont en train d'être adoptées pour compléter celles plus traditionnelles de simulation et de test. Tant en raison des limitations techniques des outils que pour des raisons évidentes de coût de développement, l'utilisation des méthodes formelles est réservée à de petites applications – éventuellement complexes – dans des domaines précis (aéronautique, télé-

communications, transport, énergie, etc.). Néanmoins, dans ces domaines, les chercheurs et les ingénieurs effectuent de plus en plus d'études de cas à échelles réellement industrielles, mettant en évidence les limitations à l'utilisation plus systématique des méthodes formelles dans l'industrie.

C'est dans ce cadre très général que va se situer ce travail de thèse : améliorer et compléter les méthodes de vérification existantes.

Contexte scientifique : l'utilisation des méthodes formelles

Les *méthodes formelles* regroupent tout un ensemble de notations et de concepts liés aux mathématiques et à la logique. L'intérêt des méthodes formelles est d'offrir la possibilité de vérifier de manière rigoureuse des propriétés sur un modèle formel. Pour s'assurer qu'un programme respecte certaines propriétés, on lui associe une spécification formelle à partir de laquelle des outils implémentent des méthodes de raisonnement mathématique. Le contexte de mon travail est de *spécifier* pour *vérifier* des systèmes.

Spécifier...

L'activité de spécification consiste avant tout à établir une description formelle d'un système et de ses propriétés. Une spécification formelle utilise un langage avec une syntaxe et une sémantique définies mathématiquement.

La spécification intervient en amont de tout processus formel de développement. Il va s'agir de formaliser le comportement du programme ou du système que l'on souhaite mettre en place. Cette formalisation s'effectue à partir du cahier des charges et de discussions avec le donneur d'ordre. Le système à modéliser est souvent décrit en langage naturel et de manière extensive : le cahier des charges regroupe l'ensemble des fonctionnalités requises par le système à vérifier ainsi qu'une description informelle des besoins. Un modèle formel devra permettre de lever toutes les ambiguïtés de l'expression informelle et fournir une description claire du système et des propriétés à vérifier.

L'étape de spécification est une étape difficile. Il peut sembler vain d'essayer de modéliser toutes les fonctionnalités et tous les besoins. Tout dépend de l'objectif de la phase de vérification. Pour cela se placer au bon niveau d'abstraction est essentiel afin de ne pas prendre en compte les détails inutiles à la vérification.

Il est impossible de dresser une liste exhaustive de l'ensemble des méthodes de spécifications existantes. On trouve des méthodes et des modèles adaptés à tous les concepts technologiques des sciences informatiques (systèmes distribués, langages à objet, systèmes synchrones, systèmes concurrents, etc.). Citons, à titre d'exemple, les systèmes d'événements B [Abr96a, AM98], VDM, les algèbres de processus comme CCS, les Statecharts [Har87, Har98], TLA [Lam94], etc.

Certains systèmes trop complexes sont difficiles à spécifier de manière monolithique, en une

seule étape et en un seul morceau. Deux méthodes aident à la spécification des systèmes informatiques trop complexes : la *composition* et le *raffinement*.

Composition

De nombreux systèmes informatiques sont, par nature même, formés de plusieurs composants : systèmes d'exploitation, protocoles de réseaux, gestionnaires de processus, et tous les programmes concurrents et parallèles d'une manière générale. Pour construire une modélisation du système global, il semble alors naturel de commencer par spécifier chacun des composants du système.

La *composition* est une caractéristique importante lorsqu'on développe des systèmes complexes de grande taille. Elle permet une spécification structurée avec des composants simples et réutilisables. Le principe de composition consiste à construire la spécification "morceau par morceau" : le système complet est spécifié en ajoutant de nouveaux éléments ou en réutilisant des éléments existants en les combinant les uns avec les autres de manière formelle. Il est nécessaire de préciser comment les interactions pour la communication entre les différentes parties de la spécification ont lieu. Elle peut s'effectuer grâce à des variables partagées ou par échange de messages. Le mode de communication peut être synchrone ou asynchrone.

Une approche connexe est la *décomposition* : certains systèmes complexes ne sont pas manipulables pris dans leur totalité. On peut alors envisager de les décomposer en plusieurs sous-systèmes relativement indépendants, correspondant chacun à une partie de la spécification de départ. Le mode de communication peut là encore être synchrone ou asynchrone.

Raffinement

Le modèle d'un système complexe ne saurait être construit en une seule étape. En pratique, une spécification est toujours réalisée par *approximations successives*. Le *raffinement* est un concept qui formalise ces approximations, et le lien qu'elles ont entre elles. Le raffinement est avant tout un processus de conception, allant de l'*abstrait* vers le *concret*.

Le raffinement consiste, en premier lieu, à détailler et à préciser les structures de données et les traitements décrits par le modèle : en général, on part d'une spécification riche en variables d'états *mathématiques* et l'on aboutit, par raffinements successifs, à une implémentation où il y a peut-être d'avantages de variables, mais celles-ci sont simplifiées et plus proches des structures informatiques usuelles (tableaux, scalaires, fichiers, etc.).

Dans un processus de raffinement, on peut aussi ajouter des détails nouveaux extraits petit à petit du cahier des charges initial : il s'agit alors de réduire la "granularité" du temps [Abr96a], c.à.d. de rendre observable de nouvelles actions qui détaillent le comportement de notre système. Chaque ajout de détails à un modèle initial se traduit par un nouveau modèle qui doit rester cohérent avec le précédent, et qui nécessite donc des vérifications. Si c'est bien le cas, on dit que le deuxième modèle *raffine* le premier.

... pour vérifier

L'objectif principal de l'utilisation des méthodes formelles est la vérification de systèmes informatiques. Une fois une spécification formelle du système établie, il s'agit de raisonner à son propos afin d'assurer que les propriétés que l'on veut garantir pour un système soient bien vérifiées. Il existe deux approches bien établies pour la vérification des propriétés des programmes, le *model-checking* et le *theorem proving*.

Techniques de preuve

Avec les techniques de preuve[1], le système et la propriété à vérifier sont donnés par des formules d'une logique. La logique dépend du formalisme mathématique choisi. Elle est donnée par un système formel, composé d'un ensemble d'axiomes muni de règles d'inférence, définissant une théorie. Le principe du *theorem proving* consiste alors à trouver une preuve de la propriété, c.à.d. une instantiation des axiomes du système qui utilise les règles d'inférence de la théorie considérée pour inférer la propriété, considérée comme un but à atteindre.

L'automatisation des techniques de preuve nécessite l'utilisation de règles d'induction, d'heuristiques, de règles de réécritures et de simplifications. A l'heure actuelle, il existe un large éventail de prouveurs de théorèmes : ils vont du plus automatisé, au plus interactif et du plus généraliste au plus spécialisé. Citons, à titre d'exemple, Coq [Coq], HOL [HOL], PVS [PVS], STeP [BBC+98] ou encore l'Atelier B [Abr96a, Ate].

Contrairement aux techniques de vérifications énumératives, l'infinitude des systèmes ne pose pas de problème aux outils de preuve. Ils reposent sur des techniques telles que l'induction pour faire des preuves sur les domaines infinis.

Néanmoins, l'une des limitations qui freine encore l'utilisation à l'échelle industrielle de la preuve assistée par ordinateur est le fait que les prouveurs de théorèmes doivent être guidés par le vérificateur pour obtenir des résultats : à l'heure actuelle, même si les prouveurs de théorèmes arrivent à traiter automatiquement un grand nombre de preuves, il en reste toujours qui requièrent l'intervention d'un spécialiste. Quelque soit leur niveau d'automatisation, tous les outils de preuve actuels nécessitent d'être guidés par l'utilisateur, ce qui est extrêmement long et demande une très grande connaissance de la part de l'utilisateur, aussi bien au niveau du modèle à vérifier que des techniques de preuves utilisées.

Les techniques de preuves forment donc un ensemble de méthodes semi-automatiques, en contraste avec les approches fondées sur la vérification algorithmique du modèle, qui nécessitent quant-à elles une intervention minimale de la part du vérifieur.

Vérification algorithmique

Le model-checking est une technique fondée sur la construction d'un modèle du système à vérifier. On parle de technique de vérification *fondée sur un modèle*. Le premier objectif est

[1] Theorem proving en anglais

alors de déterminer l'espace d'états du système étudié, c.à.d. l'ensemble des états auxquels le système peut accéder. Intuitivement, la vérification est effectuée par une énumération exhaustive de l'espace d'états : elle s'applique à un graphe représentant le comportement du système, appelé graphe d'accessibilité, dont les sommets correspondent aux états accessibles et les arrêtes aux évolutions possibles du système, passant d'un état à l'autre. On qualifie souvent de *système de transitions*, le modèle servant de base à l'application des techniques de vérification algorithmique.

La vérification fondée sur un modèle peut se diviser en deux grandes branches comme suit.

- Soit on vérifie que le graphe d'accessibilité satisfait une certaine propriété, formalisée le plus souvent grâce à une logique adaptée. On parle alors de *model-checking*. De nombreuses logiques peuvent être utilisées, regroupées sous le terme générique de logiques temporelles, car elles permettent d'exprimer des propriétés liées à la succession des états et/ou des actions dans le graphe d'accessibilité. Citons la LTL (Linear Temporal Logic), la CTL (Computation Tree Logic), les modalités B, le μ-calcul, etc. Un algorithme efficace d'exploration du graphe d'accessibilité est alors utilisé pour s'assurer que le système est bien un modèle de la propriété à vérifier.
- Soit on vérifie que le graphe d'accessibilité satisfait une certaine relation de "conformité" avec un autre graphe d'accessibilité issu d'un autre système. Différentes notions de conformité ont été explorées, que ce soit l'inclusion de langage, l'équivalence observationnelle ou la bisimulation. Ce que l'on prouve dans tous les cas, c'est que le système étudié exhibe un comportement "similaire" à celui de l'autre système. Une exploration exhaustive conjointe des graphes d'accessibilité des deux systèmes à comparer permet, le plus souvent, de conclure.

Reposant sur une énumération exhaustive des états, les techniques de vérification fondées sur un modèle ne traitent pas des systèmes infinis. Les systèmes infinis permettent notamment de tenir compte de certains aspects du système réel qui ne sont pas modélisables avec des systèmes de transitions finis : structures de données complexes et domaines des variables infinis, files d'attente non bornées, contraintes temps-réels, etc.

Le model-checking présente l'avantage d'être une technique complètement automatisée et rapide, quelque fois même qualifiée de "presse bouton". Un autre avantage du model-checking est qu'il est capable de fournir des contre-exemples, mettant en évidence les problèmes de conception et aidant au débogage. En contrepartie, le principal inconvénient du model-checking est le problème de l'explosion d'états. Actuellement, les model-checkers peuvent traiter des systèmes dont le graphe d'accessibilité peut être constitué de 10^{20} états atteignables [BCM+90] – avant d'atteindre les limites mémoires actuelles – ce qui permet déjà de traiter de nombreux exemples industriels. Plusieurs outils permettent d'utiliser en pratique les techniques de vérification algorithmique. Citons entre autres, SPIN [Hol97, SPI], SMV [SMV], Uppaal [Upp], Kronos [Kro] HyTech [HyT] ou CADP [GJM+97, VAS].

Néanmoins, les méthodes de vérification fondées sur l'exploration des modèles finis sont souvent exponentielles sur la taille de la formule spécifiant la propriété à vérifier et (surtout) le nombre d'états et de transitions augmente également de manière exponentielle sur la taille des domaines des variables d'état. Un autre critère d'explosion du nombre d'états est

le nombre de composants dans le cas de systèmes concurrents. Les limites des techniques de model-checking sont encore atteintes dans de nombreux exemples industriels, et le challenge technique est de découvrir des méthodes permettant de traiter des espaces d'états encore plus importants :
- optimiser la représentation mémoire en utilisant des structures de données efficaces comme les tables de hashage ou les BDDs[2],
- réduire la taille du modèle à vérifier par des techniques d'ordre partiel, d'abstraction, de *raffinement*, de *composition/décomposition*,
- combiner model-checking et preuve de théorèmes.

Problématique, contributions et plan

Le contexte précis de cette thèse est celui de la modélisation par composition de systèmes de transitions et du développement par raffinement pour la vérification algorithmique de propriétés. L'objectif principal est la définition de techniques de vérification par exploration des modèles apportant une réponse, même partielle, aux problèmes de l'explosion combinatoire.

La *composition* et le *raffinement* peuvent être des paradigmes de spécification et de développement compatibles pour s'assurer de la sécurité des logiciels. Ces travaux proposent une approche pour la vérification de systèmes qui combine spécification par composants et raffinement. L'objectif est de repousser – au moins partiellement – le problème de l'explosion combinatoire en étudiant les composants, plutôt que le système complet lors de la vérification par exploration de modèles à états finis de grande taille, qu'il s'agisse de la vérification du raffinement ou de la vérification de propriétés : au lieu d'appréhender le système dans sa globalité, il s'agira de l'étudier "morceaux par morceaux", en diminuant ainsi l'impact du phénomène d'explosion combinatoire, les composants étant par nature plus petits que le système complet global.

Des travaux antérieurs menés au LIFC ont permis d'établir une relation de raffinement entre deux systèmes finis SA et SR basée sur une relation de τ-simulation garantissant que le raffinement n'introduit ni de nouveaux états de blocages, ni des cycles formés uniquement de nouvelles transitions [BJK00]. L'avantage principal de cette relation de raffinement est qu'elle préserve la LTL [DJK03] : une propriété LTL satisfaite par un système abstrait SA est automatiquement satisfaite par un système plus concret SR, si la relation de raffinement peut être établie entre SA et SR.

- Dans un premier temps, nous étudions la *décomposition* d'un système combinée avec le raffinement. La décomposition d'un système de transitions en plusieurs sous-systèmes peut être vue comme une sorte d'"union" des sous-systèmes qui permet d'étudier séparément chacun des sous-systèmes.

[2]Binary Decision Diagram : arbre de décision binaire ordonné.

Notons que la décomposition introduit des blocages dans les sous-systèmes. Afin d'étudier compositionnellement le raffinement d'un système décomposé, nous définissons une *relation de raffinement affaiblie* qui prend en compte les états de blocages introduits par la décomposition. Une procédure de réduction permet de s'assurer qu'il s'agit bien de blocages issus de la décomposition et non du raffinement. Il suffit alors d'étudier le raffinement affaibli de chaque sous-système pour conclure à propos du raffinement (affaibli) du système complet.
La décomposition permet également de vérifier compositionnellement certaines propriétés du système complet. Des invariants et des invariants dynamiques (c.à.d. une propriété LTL de la forme $\Box(sp \Rightarrow \bigcirc sp')$) sont vérifiés sur chacun des sous-systèmes, puis préservés pour le système complet. Nous constatons que si un état est atteignable dans l'un des sous-systèmes, il le sera, *a fortiori*, dans le système complet : un algorithme est donné qui permet de s'assurer, de manière compositionnelle, de l'(in)atteignabilité d'un état (ou d'un ensemble d'états) en parcourant successivement les différents sous-systèmes. D'une manière plus générale, nous montrons que les propriétés LTL de sûreté qui ne sont pas satisfaites par au moins l'un des sous-systèmes, ne seront pas non plus satisfaites par le système complet.

- Nous proposons ensuite de spécifier par raffinement un système décrit sous la forme d'un *système à composants synchronisés*. Les composants représenteront des éléments réutilisables que nous ferons collaborer en les synchronisant grâce à un *produit synchronisé* spécifique. Nous souhaitons pouvoir étudier le système à composants sans calculer ce produit, mais en focalisant notre analyse sur les composants du système pris les uns après les autres. Nous considérons pour cela un *composant sous-contexte*, c.à.d. un composant enrichi par le contexte des autres composants. Nous montrons que les composants sous-contexte forment une *décomposition* du système à composants, obtenue automatiquement, sans avoir besoin de considérer le système dans son ensemble.
  Etudier compositionnellement le raffinement d'un système à composants synchronisés se ramène alors à étudier un raffinement par décomposition, en considérant successivement chacun des composants sous-contexte du système à composants. Le raffinement du système à composants synchronisés est assuré en étudiant successivement le raffinement affaibli de chacun des composants sous-contexte, puis en calculant l'ensemble des "nouveaux" blocages *réductibles* pour chaque composant sous-contexte.
  Dans le cadre des systèmes à composants synchronisés, certaines propriétés locales à un composant – notamment les invariants et plus généralement toutes les propriétés de sûreté – sont automatiquement satisfaites par le système à composants synchronisés complet : il est en effet plus facile de vérifier une propriété sur l'un des composants – souvent plus petit – plutôt que sur le système complet. De plus, comme dans le cas des systèmes simples, le raffinement garantit la préservation par le système raffiné des propriétés satisfaites par le système abstrait, ce qui permet de diminuer le phénomène d'explosion combinatoire : nous pouvons envisager de vérifier, par model-checking, une propriété sur un des composants du système abstrait. Elle sera automatiquement préservée sur le système à composants raffinés.

Grâce aux composants sous-contexte, nous bénéficions toujours des résultats à propos de la vérification par décomposition de certaines propriétés globales, notamment, les invariants, les invariants dynamiques et l'(in)atteignabilité.

- La démarche de vérification compositionnelle du raffinement d'un système à composants synchronisés a été expérimentée sur un prototype appelé SynCo. Cet outil construit la relation de raffinement *affaiblie* entre chacun des composants sous-contexte et permet de conclure de manière automatique à propos du raffinement du système complet.

Ce document est organisé de la façon suivante.

Contexte scientifique

- Le chapitre 1 nous permet de présenter les notions de bases sur lesquelles se fonde notre travail, en particulier les systèmes de transitions doublement étiquetés ainsi que le produit synchronisé de systèmes de transitions. Nous présentons aussi succinctement les systèmes d'événements B.
- Le chapitre 2 est consacré au raffinement. Après avoir présenté différentes notions classiques de raffinement, nous présentons de manière détaillée la relation de raffinement définie dans [BJK00] et qui est à la base de ce travail de thèse.
- Au chapitre 3 nous présentons la Logique Temporelle Linéaire et plus généralement les logiques temporelles et les différentes catégories de propriétés. Nous établissons un lien entre la vérification de propriétés et la description compositionnelle des systèmes, avant de présenter en détail la préservation des propriétés LTL dans le cadre de la relation de raffinement.

Contributions

- Le chapitre 4 est consacré à la vérification par *décomposition*. Nous définissons la décomposition d'un système de transitions doublement étiquetés. Nous étudions le raffinement par décomposition en définissant une relation de raffinement *affaiblie*. De plus, nous montrons que certaines propriétés peuvent être vérifiées par décomposition.
- Dans le chapitre 5, nous présentons les *systèmes à composants synchronisés*. Nous établissons un lien entre les systèmes à composants et la décomposition avant de présenter le raffinement compositionnel d'un système à composants synchronisés. Nous indiquons que certaines propriétés locales à un composant sont préservées pour le système à composants synchronisés complet.
- L'objectif du chapitre 6 est de présenter SynCo, le prototype que nous avons développé pour valider la démarche de vérification compositionnelle du raffinement des systèmes à composants synchronisés. Deux études de cas – *Essuyage avant* et *CEPS* – sont aussi détaillées. La première porte sur l'étude du module d'essuyage avant d'un système d'essuie-vitre pour un pare-brise automobile. La seconde décrit un porte-monnaie électronique respectant la norme CEPS.

Chapitre 1

Modèles des systèmes

Ce chapitre vise à présenter les notions de base sur lesquelles se fonde notre travail.

De nombreux langages de spécification de haut niveau existent et permettent de décrire le comportement d'un programme. Cependant, ce comportement peut toujours être caractérisé, en première approximation, par une suite d'états stables, entrecoupés de sauts qui font passer d'un état au suivant. Au lieu d'envisager des modèles adaptés à telle ou telle catégorie de systèmes, nous allons plutôt favoriser une formulation unique, les systèmes de transitions doublement étiquetés (ST2Es). Ce choix relève avant tout de la simplicité.

C'est pourquoi la section 1.1 contient plusieurs définitions à propos de systèmes de transitions, de structures de Kripke et de systèmes de transitions doublement étiquetés. Certaines définitions dans cette section et dans les sections suivantes sont librement inspirées de celles données dans [S+99].

Comme nous nous plaçons dans un contexte compositionnel, nous utilisons des systèmes de transitions pour spécifier et modéliser des composants que nous ferons ensuite interagir les uns avec les autres. La section 1.2 dresse un état de l'art à propos des *compositions* de systèmes au niveau des ST2Es. Nous étudions certaines compositions classiques et en particulier le *produit synchronisé*, puis nous établissons un lien avec d'autres formes de composition plus spécifiques, pouvant être vues comme des instances particulières du produit synchronisé.

Finalement, nous présentons brièvement dans la section 1.3 les *systèmes d'événements B*, ainsi que la méthode de vérification par preuve associée. Les systèmes d'événements B peuvent être vus comme un "modèle de haut niveau" permettant de spécifier de manière simple des ST2Es de grande taille.

1.1 Systèmes de transitions étiquetés comme modèles

1.1.1 Systèmes de transitions (étiquetés)

Classiquement, les systèmes de transitions sont utilisés comme un modèle du comportement des systèmes.

Définition 1.1 (Système de transitions (ST))
Un système de transitions est un couple de la forme (Q, T) *où :*
– Q *est un ensemble d'états, et*
– $T \subseteq Q \times Q$ *est une relation de transition.*

Souvent, un ensemble d'états $Q_0 \subseteq Q$ (ou uniquement un état initial q_0) est ajouté pour représenter l'ensemble des états initiaux du système.

Les transitions d'un ST représentent souvent des actions différentes du système modélisé et l'on souhaitera pouvoir les différencier. On étiquettera pour cela les transitions avec le nom de l'action modélisée. On parle alors de système de transitions étiqueté.

Définition 1.2 (Système de transitions étiqueté (STE))
Un système de transitions étiqueté est un quadruplet de la forme (Q, Q_0, E, T) *où :*
– Q *est un ensemble d'états,*
– $Q_0 \subseteq Q$ *est un ensemble des états initiaux du système,*
– E *est un ensemble de noms d'actions, et*
– $T \subseteq Q \times E \times Q$ *est une relation de transition étiquetée.*

On dira qu'un STE $S = (Q, Q_0, E, T)$ est *fini* s'il est à nombre d'états et de transitions finis, c.à.d. si Q et E sont finis. Autrement, on dira que S est *infini.*

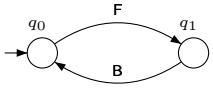


Fig. 1.1: Exemple de STE

La figure 1.1 donne une représentation graphique d'un STE très simple décrivant le comportement d'un bras robotisé qui peut s'avancer (F pour *front*) et se reculer (B pour *back*).
Les états sont représentés par des cercles. Les transitions sont représentées par des flèches orientées de l'état source vers l'état cible. L'étiquette de la transition est indiquée sur la flèche. Les états initiaux du STE sont identifiés par des transitions entrantes, non-étiquetées et sans état source.

Un élément $(q, e, q') \in T$ sera aussi noté $q \xrightarrow{e} q'$. On utilisera quelquefois les notations suivantes (quand le contexte le permettra).
– $q \xrightarrow{e} q' \stackrel{\text{def}}{=} q \xrightarrow{e} q' \in T$
– $q \xrightarrow{e} \stackrel{\text{def}}{=} \exists q'.\ (q \xrightarrow{e} q' \in T)$
– $q \rightarrow q' \stackrel{\text{def}}{=} \exists e.\ (q \xrightarrow{e} q' \in T)$
– $q \rightarrow \stackrel{\text{def}}{=} \exists e, q'.\ (q \xrightarrow{e} q' \in T)$
– $q \nrightarrow \stackrel{\text{def}}{=} \forall q', e.\ (q \xrightarrow{e} q' \notin T)$

Les chemins et les traces décrivent les comportements d'un STE, c.à.d. l'ensemble des états et des transitions qui ont été exécutés.

Définition 1.3 (Chemins, traces et exécutions)
Soit $S = (Q, Q_0, E, T)$ *un STE. Un* chemin σ *de* S *est une suite, finie ou infinie, de transitions* $q_i \xrightarrow{e_i} q_{i+1}$ *telle que* $\forall i.(i \geq 0 \Rightarrow \exists e_i.(e_i \in E \wedge q_i \xrightarrow{e_i} q_{i+1}))$. *On notera une telle suite* $\sigma = q_1 \xrightarrow{e_1} q_2 \xrightarrow{e_2} q_2 \ldots q_i \xrightarrow{e_i} q_{i+1} \ldots$.
On désignera par $tr(\sigma) = e_1.e_2 \ldots e_i.e_{i+1} \ldots$, *la* trace *d'un chemin* σ, *c.à.d. la séquence des étiquettes des transitions rencontrées lors du parcours de* σ. *La longueur d'un chemin* σ, *notée* $|\sigma|$, *est le nombre, éventuellement infini, de transitions qu'il contient.*
On notera $\sigma(i)$ *pour désigner l'état d'un chemin* σ *atteint après* i *transitions et* σ^i *pour désigner le suffixe de* σ *commençant par* $\sigma(i)$, $\Sigma(q)$ *pour désigner l'ensemble des chemins*

partant de q et, $\Sigma(S)$, l'ensemble des chemins de S.
Parmi tous les chemins de S, on distinguera ceux qui partent depuis des états initiaux de S. Une exécution *π est un chemin de S tel que $\pi(0) \in Q_0$. $\Pi(S)$ désigne l'ensemble de toutes les exécutions de S.*

Considérons un univers d'états $\mathcal{Q}$. L'ensemble des états Q de S est le sous-ensemble des états atteignables de $\mathcal{Q}$ à partir des états initiaux de Q_0, via la relation de transitions T. Plus formellement, un état q est *atteignable*, noté $reachable(q, S)$, s'il existe une exécution $\pi \in \Pi(S)$ et un indice i tels que $\pi(i) = q$, c.à.d.

$$reachable(q, S) \stackrel{\text{def}}{=} \exists\, \pi, i.(\pi \in \Pi(S) \wedge i \geq 0 \wedge \pi(i) = q)$$

1.1.2 Structures de Kripke

Dans certains cas, on souhaitera retrouver au niveau du modèle, les variables du système modélisé. Les transitions correspondront alors à des changements de valeurs des variables considérées. Nous noterons $V = \{x_1, \ldots, x_n\}$ un ensemble fini de variables x_1, ..., x_n avec leurs domaines finis respectifs $\mathbb{D}_1$, ..., $\mathbb{D}_n$. Soit AP_V l'ensemble des propositions atomiques qui peuvent être formées à partir des éléments de V (c.à.d. un ensemble de couples variable/valeur). Formellement, AP_V est défini de la manière suivante : $AP_V \stackrel{\text{def}}{=} \{x_i = v_i |\ x_i \in V \wedge v_i \in \mathbb{D}_i\}$.
Une structure de Kripke est un système de transitions dans lequel chaque état est étiqueté par l'ensemble des propositions atomiques qui sont valides dans cet état. On parle alors de l'interprétation d'un état.

Définition 1.4 (Structure de Kripke (SK))
Une structure de Kripke définie sur V est un quintuplet de la forme (Q, Q_0, T, V, l) où :
- *Q est un ensemble d'états,*
- *$Q_0 \subseteq Q$ est un ensemble des états initiaux du système,*
- *$T \subseteq Q \times Q$ est une relation de transition,*
- *V est un ensemble de variables, et*
- *$l : Q \to 2^{AP_V}$ est une fonction d'étiquetage qui associe à chaque état du système l'ensemble des propositions atomiques.*

Souvent, il est demandé que la relation de transition soit totale, c.à.d. sans états de blocages, comme par exemple dans [CGP00].

Remarque. Contrairement à certaines approches, nous considérons que l'interprétation d'un état $l(q)$ est *injective*, c.à.d. qu'elle n'est associée qu'à un seul état $q \in Q$. Nous ne voulons pas avoir plusieurs états ayant la même interprétation.

$$\forall q_1, q_2.\ (q_1 \in Q \wedge q_2 \in Q \Rightarrow (l(q_1) = l(q_2) \Leftrightarrow q_1 = q_2))$$

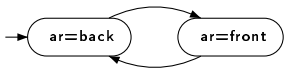


Fig. 1.2: Exemple de SK

La figure 1.2 donne une représentation graphique d'une SK correspondant au STE précédent. Une variable $ar \in \{front, back\}$ indique la position du bras. Les états, toujours représentés par des cercles contiennent l'interprétation de l'état, c.à.d. la valeur des variables du système dans cet état. Souvent on omettra les noms des états lorsqu'on donnera une représentation graphique d'une SK.

1.1.3 Systèmes de transitions doublement étiquetés

On souhaite combiner l'interprétation des états avec l'étiquetage des transitions pour bénéficier du pouvoir d'expression des STE et des SK. Dans [DNV95], De Nicola et Vaandrager ont introduit une nouvelle structure (voir définition 1.5) qui peut être projetée de manière naturelle vers les systèmes de transitions étiquetés et vers les structures de Kripke. Nous définissons, à la manière de [DNV95], un système de transitions doublement étiqueté.

Définition 1.5 (Système de transitions doublement étiqueté (ST2E))
Un système de transitions doublement étiqueté est un six-uplet de la forme (Q, Q_0, E, T, V, l) *où :*
- Q *est un ensemble d'états,*
- $Q_0 \subseteq Q$ *est un ensemble des états initiaux du système,*
- E *est un ensemble de noms d'actions,*
- $T \subseteq Q \times E \times Q$ *est une relation de transition étiquetée,*
- V *est un ensemble de variables, et*
- $l : Q \to 2^{AP_V}$ *est une fonction d'étiquetage qui associe à chaque état du système l'ensemble des propositions atomiques.*

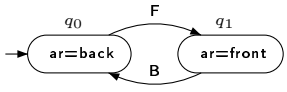


Fig. 1.3: Exemple de ST2E

Dans la suite du document, on parlera quelquefois indifféremment de ST2E, de STE ou de SK. On pourra toujours se ramener d'un ST2E à un STE ou à une SK, par projection. Soit $S = (Q, Q_0, E, T, V, l)$ un ST2E, alors $\mathrm{ST}(S) = (Q, T')$ est le ST associé à S, $\mathrm{STE}(S) = (Q, Q_0, E, T)$ est le STE associé à S, et, $\mathrm{SK}(S) = (Q, Q_0, T', V, l)$ est la SK associée à S ; avec T' qui est une "abstraction" de $Q \times E \times Q$ sur $Q \times Q$ définie par $q \to q' \in T'$ s'il existe $e \in E$ telle que $q \xrightarrow{e} q' \in T$.

Opérations particulières sur les ST2E

On souhaitera pouvoir masquer – supprimer – une partie du comportement dans un ST2E, c.à.d. enlever de S les transitions étiquetées par des éléments d'un sous-ensemble $H \subseteq E$ de l'ensemble des noms d'actions.

Définition 1.6 (Masquage)
Soit $S = (Q, Q_0, E, T, V, l')$ un système de transitions doublement étiqueté. Soit $H \subseteq E$ l'ensemble des étiquettes de transitions à masquer. Le système de transitions $\mathcal{H}(S, H)$, est défini par le six-uplet $(Q', Q'_0, E', T', V, l')$ suivant :
– $E' = E \setminus H$,
– $T' = \{(q, e, q') |\ (q, e, q') \in T \ \wedge \ e \in E'\}$,
– $Q' = \{q |\ \exists q'.\ ((q, e, q') \in T' \vee (q', e, q) \in T')\}$,
– $Q'_0 = Q_0 \cap Q'$, *et*
– l' *est la restriction de* l *sur* Q'.

On souhaitera aussi renommer certaines transitions d'un ST2E par une nouvelle étiquette de transition. On notera $e \backslash e'$ une étiquette e qui a été renommée par une nouvelle étiquette e' afin de conserver la trace de l'ancienne étiquette de la transition.

Définition 1.7 (Renommage)
Soit $S = (Q, Q_0, E, T, V, l)$ un système de transitions doublement étiqueté. Soit $R \subseteq E$ l'ensemble des étiquettes de transitions à renommer et e' la nouvelle étiquette de transition. Le système de transitions $\mathcal{R}(S, R, e')$ est défini par le six-uplet (Q, Q_0, E', T', V, l) suivant :
– $E' = (E \setminus R) \cup \{e'\}$ *et*
– $T' = \{(q, e \backslash e', q') |\ (q, e, q') \in T \ \wedge \ e \in R\} \cup \{(q, e, q') |\ (q, e, q') \in T \ \wedge \ e \in E \setminus R\}$.

1.1.4 Exemple de ST2E : un bras mobile robotisé

Remarque. L'exemple que nous allons introduire dans cette section nous servira de fil conducteur pour illustrer la plupart des idées et des résultats présentés dans l'ensemble de ce document.

Le système que nous nous proposons de modéliser constitue l'un des éléments d'un dispositif industriel de fabrication de pièces. Nous nous intéresserons plus particulièrement à spécifier le comportement d'un bras mobile robotisé chargé de déplacer des pièces qui arrivent sur un tapis roulant situé en haut du dispositif, vers un autre tapis roulant situé plus bas : voir figure 1.4.

Le bras mobile robotisé (que nous appellerons *robot*, pour simplifier) est muni à son extrémité d'une pince et peut se déplacer horizontalement et verticalement. De plus, le dispositif doit respecter les contraintes de fonctionnement suivantes :
– En position haute (face au dispositif d'arrivée), la pince du robot doit être ouverte, avant de s'avancer, ou s'ouvrir et s'avancer simultanément, afin d'éviter de percuter une pièce, pince fermée.
– En position haute, la pince doit toujours être refermée avant que le bras ne commence à reculer afin de ne pas lâcher la pièce.

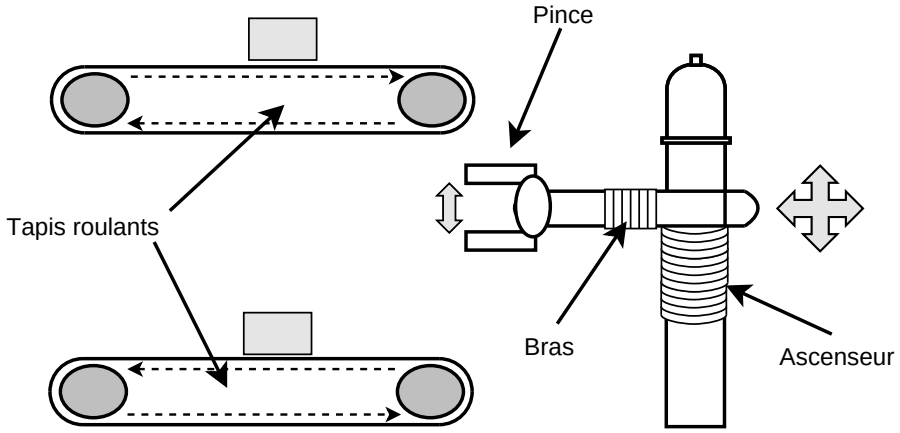


FIG. 1.4: Bras mobile robotisé

- Le bras robotisé doit forcément effectuer ses déplacements verticaux (c.à.d. passer par la position médiane) en position reculée et pince fermée.
- En position basse, le bras s'avance forcément avant que la pince ne s'ouvre afin d'éviter de lâcher trop tôt la pièce transportée.
- Une fois la pièce déposée, la bras robotisé peut soit reculer, puis refermer sa pince, puis commencer à remonter, ou bien reculer tout en refermant sa pince, puis commencer à remonter, ou finalement reculer, refermer la pince et commencer à remonter simultanément.

Au départ, le dispositif devra être en position basse, reculé et pince fermée.

$$
\begin{array}{rcl}
Q & = & \{\ q_0,\ q_1,\ q_2,\ q_3,\ q_4,\ q_5,\ q_6,\ q_7, q_8\ \}\ , \\
Q_0 & = & \{\ q_0\ \}\ , \\
E & = & \{\ F,\ O,\ B,\ C,\ BC,\ U1,\ D2,\ U2,\ D1,\ OF,\ BCU1\ \}\ , \\
T & = & \{\ q_0 \xrightarrow{F} q_1,\ q_1 \xrightarrow{O} q_2,\ q_2 \xrightarrow{B} q_3,\ q_2 \xrightarrow{BC} q_0,\ q_2 \xrightarrow{BCU1} q_4,\ q_3 \xrightarrow{C} q_0,\ q_0 \xrightarrow{U1} q_4,\ q_4 \xrightarrow{D2} q_0, \\
 & & \quad q_4 \xrightarrow{U2} q_5,\ q_5 \xrightarrow{D1} q_4,\ q_5 \xrightarrow{O} q_6,\ q_5 \xrightarrow{OF} q_7,\ q_6 \xrightarrow{F} q_7,\ q_7 \xrightarrow{C} q_8,\ q_8 \xrightarrow{B} q_5\ \}, \\
V & = & \{\ el,\ ar,\ cl\ \}\ \text{, avec } \mathbb{D}_{el} = \{down, mid, up\},\ \mathbb{D}_{ar} = \{back, front\} \text{ et } \mathbb{D}_{cl} = \{close, open\}, \\
l & = & \{\ q_0 \mapsto \{el = down,\ ar = back,\ cl = close\},\quad q_1 \mapsto \{el = down,\ ar = front,\ cl = close\}, \\
 & & \quad q_2 \mapsto \{el = down,\ ar = front,\ cl = open\},\quad q_3 \mapsto \{el = down,\ ar = back,\ cl = open\}, \\
 & & \quad q_4 \mapsto \{el = mid,\ ar = back,\ cl = close\},\quad q_5 \mapsto \{el = up,\ ar = back,\ cl = close\}, \\
 & & \quad q_6 \mapsto \{el = up,\ ar = back,\ cl = open\},\quad q_7 \mapsto \{el = up,\ ar = front,\ cl = open\}, \\
 & & \quad q_8 \mapsto \{el = up,\ ar = front,\ cl = close\}\quad \}
\end{array}
$$

FIG. 1.5: ST2E *Robot* : six-uplet (Q, Q_0, E, T, V, l)

Le ST2E *Robot* décrit par le six-uplet (Q, Q_0, E, T, V, l), figure 1.5, donne une modélisation du comportement du bras mobile robotisé. Il est composé de 9 états et de 15 transitions. Trois variables permettent de décrire les différentes configurations que peut prendre le bras robotisé :

- La variable el (pour *Elevator*) donne la position verticale du robot : soit le bras robotisé est situé en haut du dispositif ($el = up$), c.à.d. au niveau du tapis roulant qui fait

arriver de nouvelles pièces, soit en position médiane ($el = mid$), soit en bas du dispositif ($el = down$), c.à.d. au niveau du dispositif qui évacue les pièces.
- La variable ar (pour *Arm*) décrit la position horizontale du bras robotisé. Le bras peut être avancé ($ar = front$) – afin de pouvoir saisir ou déposer une pièce – ou bien reculé ($ar = back$).
- La variable cl (pour *Clip*) donne l'état de la pince située à l'extrémité du bras robotisé. Soit la pince est ouverte ($cl = open$), soit elle est fermée ($cl = close$).

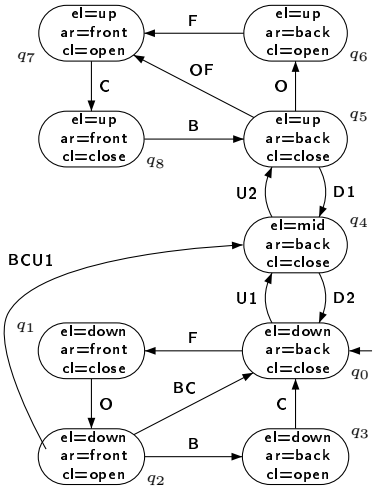


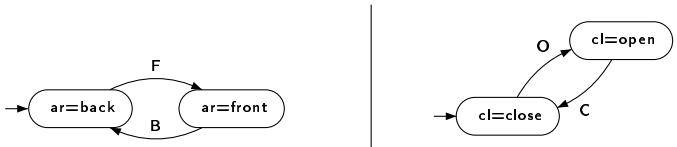
Fig. 1.6: ST2E *Robot* : représentation graphique

Une représentation graphique du ST2E *Robot* est donnée figure 1.6. Toutes les transitions modélisent des changements d'états du robot et les étiquettes de transitions associées indiquent quel est le changement effectué :
- les transitions O (pour *Open*) et C (pour *Close*) représentent l'ouverture et la fermeture de la pince,
- B (pour *Back*) et F (pour *Front*) représentent les mouvements horizontaux du robot,
- les transitions $U1$, $U2$ (pour *Up*), $D1$ et $D2$ (pour *Down*) représentent les mouvements verticaux du robot,
- la transition OF (pour *Open* et *Front*) représente l'avancée du bras simultanément à l'ouverture de la pince,
- la transition BC pour (*Back* et *Close*) représente le recul du bras simultanément à la fermeture de la pince, et
- la transition $BCU1$ (pour *Back*, *Close* et *Up*) représente le recul du bras simultanément à la fermeture de la pince est à un mouvement de montée.

Remarque. Dans cette spécification, on ne modélisera pas l'arrivée d'une pièce, ni son évacuation, ni même le transport d'une pièce. Nous nous intéresserons uniquement aux différents mouvements du bras mobile robotisé.

1.2 Compositions classiques des systèmes de transitions

Dans la section précédente, nous avons présenté différentes notions permettant de modéliser le comportement de systèmes réels assez simples. Lorsque l'on essaye de donner un modèle d'un programme ou d'un système plus complexe, ceux-ci sont souvent décomposés en "modules" ou "composants". Pour donner un modèle du système complet, il semble assez naturel de commencer par donner un modèle pour chacun des composants. Il faut ensuite expliquer comment les différentes composantes du système coopèrent ensemble – se synchronisent – pour donner un modèle du système complet.



(a) ST2E *Arm* (b) ST2E *Clip*

Fig. 1.7: ST2Es composant un système

Exemple. La figure 1.7 présente deux ST2E qui serviront à illustrer les différentes définitions que nous allons présenter dans la suite de cette section.
- La figure 1.7(a) illustre le ST2E *Arm*, déjà présenté précédemment figure 1.3. Ce ST2E spécifie un bras robotisé vertical prenant deux positions : avant ($ar = front$) et arrière ($ar = back$).
- La figure 1.7(b) illustre le ST2E *Clip* qui modélise le comportement d'une pince. La pince peut être fermée ($cl = close$) ou bien ouverte ($cl = open$).

1.2.1 Produit cartésien de ST2Es

La manière la plus simple de modéliser la façon qu'ont de coopérer les différents composants d'un système dans le cas où ils n'interagissent pas les uns avec les autres est d'utiliser le *produit cartésien*. Ce produit permet de définir la coopération de deux ST2E indépendants l'un de l'autre.

Remarque. On introduit une nouvelle étiquette de transition ' $-$ ' correspondant à l'action fictive "ne rien faire". On notera $E^1 \otimes E^2 \stackrel{\text{def}}{=} ((E^1 \cup \{-\}) \times (E^2 \cup \{-\})) \setminus \{(-,-)\}$. De même, on notera (q_1, q_2) un état de $Q^1 \times Q^2$ et $(q_1, q_2) \stackrel{(e_1,e_2)}{\longrightarrow} (q'_1, q'_2)$ une transition de $(Q^1 \times Q^2) \times (E^1 \otimes E^2) \times (Q^1 \times Q^2)$.

Définition 1.8 (Produit cartésien (ou produit libre))
Soient $S^1 = (Q^1, Q^1_0, E^1, T^1, V^1, l^1)$ et $S^2 = (Q^2, Q^2_0, E^2, T^2, V^2, l^2)$ deux ST2E. Le produit cartésien de S^1 et S^2, noté $S^1 \times S^2$ est défini par le six-uplet (Q, Q_0, E, T, V, l) :
- $Q = Q^1 \times Q^2$,
- $Q_0 = Q^1_0 \times Q^2_0$,
- $E = E^1 \otimes E^2$,

- $T = Q \times E \times Q$ *est définie ainsi :*
  - $(q_1, q_2) \xrightarrow{(e_1,-)} (q'_1, q_2) \in T$ *si* $q_1 \xrightarrow{e_1} q'_1 \in T^1$
  - $(q_1, q_2) \xrightarrow{(-,e_2)} (q_1, q'_2) \in T$ *si* $q_2 \xrightarrow{e_2} q'_2 \in T^2$
  - $(q_1, q_2) \xrightarrow{(e_1,e_2)} (q'_1, q'_2) \in T$ *si* $q_1 \xrightarrow{e_1} q'_1 \in T^1$ *et* $q_2 \xrightarrow{e_2} q'_2 \in T^2$
- $V = V^1 \cup V^2$,
- $l((q_1, q_2)) = l^1(q_1) \cup l^2(q_2)$.

Dans le système résultant du produit cartésien, on retrouve tous les comportements des deux composants. Les composants peuvent évoluer simultanément ou séparément. Chaque composant peut effectuer une transition locale ou participer à une transition globale (on dira alors qu'il se "synchronise"), ou bien encore ne rien faire.

Remarque. Par soucis de lisibilité nous avons fait le choix d'exprimer la définition précédente (ainsi que les définitions suivantes) pour 2 composants uniquement, bien que ces définitions puissent être généralisées à n composants, sans grande difficulté.

> **Exemple.** La figure 1.8 montre le produit cartésien de Arm avec $Clip$. Nous remarquons que le ST2E $Arm \times Clip$ présente tous les comportements possibles issus de Arm et de $Clip$, aussi bien des comportements locaux : avancée $(F, -)$, recul $(B, -)$, ouverture $(-, O)$, ou fermeture $(-, C)$; que des comportements synchronisés : avancée/ouverture (F, O), avancée/fermeture (F, C), recul/ouverture (B, O) ou recul/fermeture (B, C).

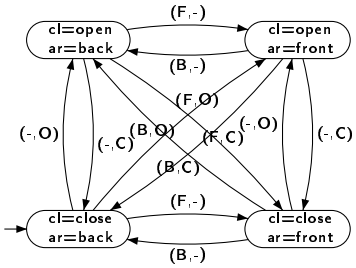


Fig. 1.8: Produit cartésien $Arm \times Clip$

1.2.2 Produit synchronisé de ST2Es

Comme nous l'avons précisé précédemment, le produit cartésien autorise tous les comportements possibles. Si l'on souhaite maintenant modéliser un système dans lequel une transition e_1 d'un composant S^1 ne peut avoir lieu que simultanément à une transition e_2 d'un autre composant S^2, le produit cartésien n'est pas assez expressif. Pour répondre à cela, Arnold et Nivat ont introduit dans [AN82, Arn92] la notion de *produit synchronisé*.

Afin de décrire plus précisément les comportements autorisés, les transitions du produit cartésien sont restreintes grâce à un *ensemble de synchronisations*.

Définition 1.9 (Ensemble de synchronisations)
Soient $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ *et* $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ *deux ST2E. Un ensemble de synchronisations* Syn *est un sous-ensemble de* $E^1 \otimes E^2$.

Soit Syn un ensemble de synchronisations. Il indique parmi les éléments de $E^1 \otimes E^2$, lesquels correspondent à des comportements souhaités, et ceux qui n'ont pas lieu d'être[1]. On parlera de *transitions synchronisées* pour désigner les transitions présentes dans Syn.

Il est maintenant possible de définir formellement le produit synchronisé. Ce produit ne contiendra que les transitions synchronisées qui figurent dans l'ensemble des transitions autorisées.

Définition 1.10 (Produit synchronisé)
Soient $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ *et* $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ *deux ST2E. Soit* Syn *un ensemble de synchronisations. Le produit synchronisé de* S^1 *et* S^2, *noté* $S^1 \times_{Syn} S^2$ *est défini par le six-uplet* (Q, Q_0, E, T, V, l) :
- $Q \subseteq Q^1 \times Q^2$,
- $Q_0 \subseteq Q_0^1 \times Q_0^2$,
- $E = Syn$,
- $T \subseteq Q \times E \times Q$ *est défini ainsi :*
  - $(q_1, q_2) \xrightarrow{(e_1,-)} (q_1', q_2) \in T$ *si* $q_1 \xrightarrow{e_1} q_1' \in T^1$ *et* $(e_1, -) \in Syn$
  - $(q_1, q_2) \xrightarrow{(-,e_2)} (q_1, q_2') \in T$ *si* $q_2 \xrightarrow{e_2} q_2' \in T^2$ *et* $(-, e_2) \in Syn$
  - $(q_1, q_2) \xrightarrow{(e_1,e_2)} (q_1', q_2') \in T$ *si* $q_1 \xrightarrow{e_1} q_1' \in T^1$, $q_2 \xrightarrow{e_2} q_2' \in T^2$ *et* $(e_1, e_2) \in Syn$
- $V = V^1 \cup V^2$,
- $l((q_1, q_2)) = l^1(q_1) \cup l^2(q_2)$.

Notons aussi que l'on a $S^1 \times_{Syn} S^2 = \mathcal{H}(S^1 \times S^2, \overline{Syn})$ où $\overline{Syn} = (E^1 \otimes E^2) \setminus Syn$ est le complémentaire de Syn dans $E^1 \otimes E^2$, c.à.d. l'ensemble des comportements interdits. Le produit synchronisé peut être défini comme la restriction du produit cartésien à l'ensemble Syn des comportements souhaités.

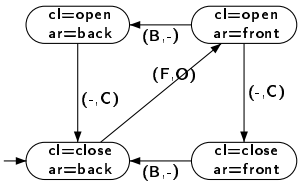


Fig. 1.9: Produit synchronisé $Arm \times_{Syn} Clip$

Exemple. Le produit cartésien $Arm \times Clip$, figure 1.8, présente trop de comportements. Supposons que l'on veuille modéliser un dispositif dans lequel l'avancée et l'ouverture aient toujours lieu simultanément. La fermeture et le

[1]Dans [Arn92], c'est un *vecteur de synchronisation* qui est utilisé pour indiquer parmi toutes les transitions de $(Q^1 \times Q^2) \times (E^1 \otimes E^2) \times (Q^1 \times Q^2)$, celles qui sont souhaitées.

recul auront toujours lieu indépendamment. Nous exprimons un ensemble de synchronisations $Syn = \{(F, O), (B, -), (-, C)\}$ afin d'autoriser uniquement les comportements souhaités, puis nous calculons le produit synchronisé de Arm avec $Clip$ sous l'ensemble de synchronisations Syn. Le résultat est illustré par la figure 1.9.

1.2.3 Compositions basées sur le produit synchronisé

Plusieurs compositions pouvant être vues comme des instances particulières du produit synchronisé sont intéressantes à remarquer : le produit asynchrone, le produit synchrone, la composition basée sur des étiquettes de transitions communes ou encore les systèmes communicants.

Produit asynchrone de deux ST2Es

Le produit asynchrone est un cas particulier du produit synchronisé dans lequel seuls les comportements individuels des composants sont autorisés. Pour calculer le produit asynchrone, on réutilise la définition 1.10 du produit synchronisé avec un ensemble de synchronisations spécifique.

Définition 1.11 (Produit asynchrone)
Soient $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ *et* $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ *deux ST2E.*
L'ensemble de synchronisations Syn *pour le produit asynchrone est défini ainsi :*

$$Syn = (E^1 \times \{-\}) \cup (\{-\} \times E^2)$$

Le produit asynchrone de S^1 *et* S^2 *est égal à* $S^1 \times_{Syn} S^2$.

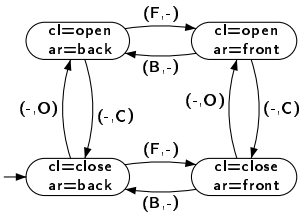


Fig. 1.10: Produit asynchrone de Arm et de $Clip$

Exemple. Nous pouvons calculer le produit asynchrone de Arm avec $Clip$. Pour cela, nous considérons un ensemble de synchronisations $Syn = \{(F, -), (B, -), (-, O), (-, C)\}$ formé uniquement des transitions simples des deux composants, puis nous calculons le produit synchronisé $Arm \times_{Syn} Clip$. Le résultat est illustré par la figure 1.10.

Produit synchrone de ST2Es

Le produit synchrone est aussi un cas particulier du produit synchronisé dans lequel uniquement les comportements synchronisés entre tous les composants sont autorisés. Pour calculer le produit synchrone, on réutilise la définition 1.10 du produit synchronisé avec un ensemble de synchronisations spécifique.

Définition 1.12 (Produit synchrone)
Soient $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ *et* $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ *deux ST2E.*
L'ensemble de synchronisations Syn *pour le produit synchrone est défini ainsi :*

$$Syn = E^1 \times E^2$$

Le produit synchrone de S^1 *et* S^2 *est égal à* $S^1 \times_{Syn} S^2$.

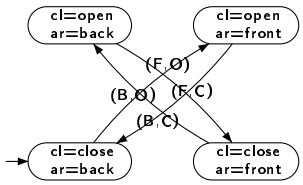


Fig. 1.11: Produit synchrone de *Arm* et de *Clip*

Exemple. Nous pouvons aussi calculer le produit synchrone de *Arm* avec *Clip*. Nous considérons un ensemble de synchronisations $Syn = \{(F,O),(B,C),(B,O),(F,C)\}$ formé de toutes les transitions synchronisées envisageables à partir de *Arm* et de *Clip*. Nous calculons ensuite le produit synchronisé $Arm \times_{Syn} Clip$. Le résultat est illustré par la figure 1.11.

Remarque. Chez Clarke [CGP00] – entre autre – le produit synchrone est appelé *composition parallèle* et est noté $S^1||S^2$.

Etiquettes de transitions communes

Une autre façon de modéliser un produit synchronisé, consite à considérer deux composants $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ et $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ tels que $E^1 \cap E^2 \neq \varnothing$, puis à réaliser la synchronisation des transitions en utilisant ces étiquettes communes. Les transitions d'étiquettes communes auront lieu simultanément, les autres transitions auront lieu indépendamment les unes des autres.

Il s'agit une fois encore d'un produit synchronisé. L'ensemble de synchronisations Syn, dans ce cas, est défini ainsi :

$$Syn = \begin{array}{l} \{(e,e) \mid e \in E^1 \wedge e \in E^2\} \ \cup \\ \{(e_1,-) \mid e_1 \in E^1 \wedge e_1 \notin E^2\} \ \cup \\ \{(-,e_2) \mid e_2 \notin E^1 \wedge e_2 \in E^2\} \end{array}$$

Exemple. Considérons les deux composants Arm et $Clip$ donnés figure 1.7. Supposons que la transition étiquetée F dans Arm ainsi que la transition étiquetée O dans $Clip$ sont renommées par FO. En synchronisant les étiquettes de transitions communes, on obtiendra le même système que le produit cartésien $Arm \times Clip$ donné figure 1.8.

Systèmes de transitions communicants

On souhaite ici modéliser le comportement de deux composants qui se synchronisent par envoi/réception de messages. Le premier composant envoie par exemple un message m qui doit être reçu par le second composant.

Pour modéliser un tel comportement, il faut distinguer parmi les transitions des deux composants, celles dont les étiquettes correspondent à l'envoi d'un message m, notée $!m$, et celles dont les étiquettes correspondent à la réception du message m, notée $?m$. A partir de là, on n'autorise comme transitions synchronisées que les transitions où l'émission d'un message est accompagnée de la réception du même message ou vice-versa : on parlera de *synchronisation par échange de messages*. Les autres transitions – celles ne correspondant ni à l'envoi, ni à la réception d'un message – ne sont pas synchronisées.

Définition 1.13 (Systèmes de transitions communicants)
Soient $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ *et* $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ *deux ST2E.*
L'ensemble de synchronisations Syn *pour deux systèmes communicants est défini ainsi :*

$$\begin{array}{rl} & \{(e_1, e_2) \mid e_1 \in E^1 \wedge e_2 \in E_2 \wedge ((e_1 = !m \wedge e_2 = ?m) \vee (e_1 = ?m \wedge e_2 = !m))\} \\ Syn = \cup & \{(e_1, -) \mid e_1 \in E^1 \wedge e_1 \neq !m \wedge e_1 \neq ?m\} \\ \cup & \{(-, e_2) \mid e_2 \in E^2 \wedge e_2 \neq !m \wedge e_2 \neq ?m\} \end{array}$$

Le système complet résultant de la synchronisation par échange de messages entre S^1 *et* S^2 *est égal à* $S^1 \times_{Syn} S^2$.

Remarque. On peut remarquer que la synchronisation par échange de messages est définie ici comme un produit synchronisé implicite qui synchronise envoi et réception de messages. On pourrait aussi souhaiter modéliser des échanges de messages qui auraient lieu de manière asynchrone. Il faut alors préciser ce que deviennent les messages émis, mais pas encore reçus. Habituellement, on considère qu'ils transitent par des *canaux de communications*, gérés le plus souvent comme des files d'attentes suivant un modèle *fifo*.

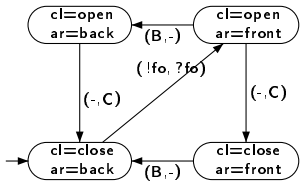


Fig. 1.12: Systèmes Arm et $Clip$ synchronisés par échange de messages

Exemple. Nous pourrions définir le produit synchronisé de $Arm \times_{Syn} Clip$ donné figure 1.9 par des systèmes communicants. Pour indiquer que l'avancée du bras et l'ouverture doivent toujours avoir lieu simultanément, nous pourrions par exemple envisager que, quand le bras commence à avancer, il envoie un message à la pince pour lui indiquer de s'ouvrir. Nous remplaçons dans Arm l'étiquette F par $!fo$, pour indiquer l'envoi du message, et nous remplaçons dans $Clip$ l'étiquette O par $?fo$, pour indiquer la réception.

L'ensemble de synchronisations du produit synchronisé implicite est $Syn^m = \{(!fo, ?fo),\ (B,-),(-,C)\}$. La figure 1.12 illustre le système résultant de la synchronisation des échanges de messages entre Arm et $Clip$.

1.2.4 Système de transitions séparable

Quelquefois, au lieu d'essayer de composer un système à partir de ses composants, on souhaitera *décomposer* le système complet en plusieurs "morceaux". Bien entendu, on peut imaginer toutes les décompositions possibles. La notion de système de transitions *séparable* introduite dans [BQ96], formalise une séparation exprimée par une *couture*, c.à.d. un ensemble d'états de liaison.

Intuitivement, un ST2E séparable S est composé de deux sous-systèmes S^1 et S^2 tels que :
- l'intersection des ensembles d'états Q^1 et Q^2 est exactement la couture.
- l'ensemble de transitions T de S est partitionné, c.à.d. que $T^1 \cap T^2 = \varnothing$.
- aucune séquence de transitions ne peut conduire d'un composant à l'autre sans passer par la couture. Les transitions entre états appartenant à la couture peuvent appartenir à l'un ou l'autre des composants, mais pas aux deux.

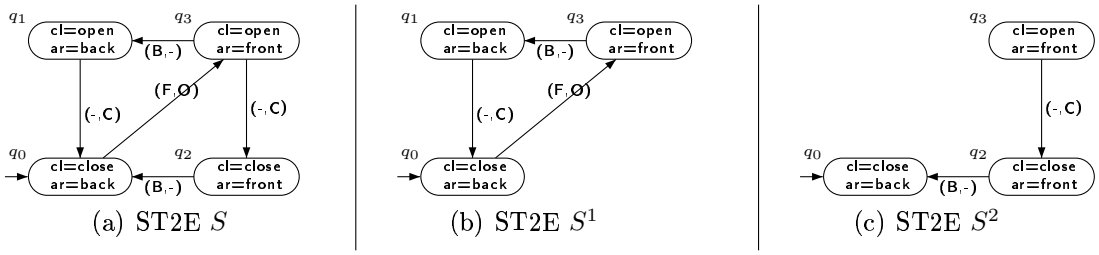


Fig. 1.13: ST2E séparable $S = S^1 \setminus_{\tilde{Q}} S^2$

Définition 1.14 (ST2E séparable [BQ96])
Soit $S = (Q, Q_0, E, T, V, l)$ un ST2E. Nous dirons que S est séparable selon la couture $\tilde{Q}$ en deux sous-systèmes $S^1 = (Q^1, Q_0^1, E, T^1, V, l)$ et $S^2 = (Q^2, Q_0^2, E, T^2, V, l)$, noté $S = S^1 \setminus_{\tilde{Q}} S^2$ si les conditions suivantes sont vérifiées :

1. $Q^1 \cup Q^2 = Q$, $Q^1 \cap Q^2 = \tilde{Q}$,
2. $T^1 \cup T^2 = T$, $T^1 \cap T^2 = \varnothing$,
3. $\neg(\exists q_1, q_2, e.(q_1 \in Q^1 \setminus \tilde{Q} \wedge q_2 \in Q^2 \setminus \tilde{Q} \wedge e \in E \wedge (q_1 \xrightarrow{e} q_2 \vee q_2 \xrightarrow{e} q_1)))$

Exemple. Considérons le ST2E $S = Arm \times_{Syn} Clip$ que nous rappelons figure 1.13(a) ainsi que la couture $\tilde{Q} = \{q_0, q_3\}$. Nous pouvons construire les ST2E S^1 et S^2, résultats de la séparation de S suivant $\tilde{Q}$: voir figures 1.13(b) et 1.13(c).

1.2.5 Systèmes à variables partagées

Les composants manipulés jusqu'ici sont des systèmes dont le comportement est complètement déterminé par l'état courant du composant, c.à.d. par la valeur des variables dans cet état. On parle alors de *système fermé* (*closed system*). Il est assez naturel d'envisager qu'une même variable puisse être utilisée – partagée – par plusieurs composants. On parle de *système ouvert* (*open system*) pour désigner un composant pour lequel la valeur de chacune des variables du système n'est pas forcément fixée par le système, mais qui peut être modifiée par un autre composant. On parle alors de composition par *variables partagées*.

Un composant à variables partagées est classiquement appelé *module* [KV96, KV97, KV98, AH99, AG00]. Pour représenter un module M, nous allons étendre la définition 1.5 en partitionnant l'ensemble des variables V en trois ensembles deux-à-deux disjoints V_p, V_i et V_e où

- V_p désignera l'ensemble des variables privées de M, c.à.d. des variables qui ne peuvent ni être lues, ni être modifiées par un autre module que M ;
- V_i désignera l'ensemble des variables d'interface de M, c.à.d. des variables qui peuvent uniquement être lues par les autres modules et qui sont modifiées par M ;
- V_e désignera l'ensemble des variables externes de M, c.à.d. des variables qui peuvent uniquement être lues par M, mais qui sont modifiées par d'autres modules.

$V_c = V_p \cup V_i$ désigne l'ensemble des variables contrôlées par M et $V_o = V_i \cup V_e$ l'ensemble des variables de M observables depuis un autre module. $V = V_p \cup V_i \cup V_e$ désigne l'ensemble de toutes les variables intervenant dans M.

Définition 1.15 (Module [KV97, AH99])
Un module M est un 8-uplet de la forme $(Q, Q_0, E, T, V_p, V_i, V_e, l)$ où :

- *Q est un ensemble d'états,*
- *$Q_0 \subseteq Q$ est un ensemble des états initiaux du système,*
- *E est un ensemble de noms d'actions,*
- *$T \subseteq Q \times E \times Q$ est une relation de transition,*
- *V_p est un ensemble de variables privées de M,*
- *V_i est un ensemble de variables d'interface de M,*
- *V_e est un ensemble de variables externes de M, et*
- *$l : Q \to 2^{AP_V}$ est une fonction d'étiquetage qui associe à chaque état du module l'ensemble des propositions atomiques vraies dans cet état.*

Bien entendu, un module est conçu pour interagir avec d'autres modules. Dans [AH99, AG00], la composition parallèle de deux modules $M^1 \| M^2$ est définie afin de pouvoir combiner ces deux modules en un seul. Un langage de spécification pour des systèmes à variables

type ARM **is** {back, front}
type CLIP **is** {close, open}

module M_{Arm}
  external cl : CLIP
  interface ar : ARM
  init
    [] true → ar := back ;
  update
    [] ar = front → ar := back ;
    [] ar = back ∧ cl = close → ar := front ;

module M_{Clip}
  external ar : ARM
  interface cl : CLIP
  init
    [] true → cl := close ;
  update
    [] cl = open → cl := close ;
    [] ar = back ∧ cl = close → cl := open ;

FIG. 1.14: Système à variables partagées $M_{Arm} \| M_{Clip}$

partagées, appelé *Reactive Modules*, est aussi défini dans [AH99, AG00, AdAG+01].

> **Exemple.** A titre d'illustration, nous exprimons le produit synchronisé $Arm \times_{Syn} Clip$ donné figure 1.9 en termes de systèmes à variables partagées. Nous utilisons la syntaxe des *Reactive Modules* [AH99] pour décrire deux modules M_{Arm} et M_{Clip}. Afin d'indiquer que l'avancée du bras ne peut avoir lieu que si la pince est fermée, nous avons déclaré la variable cl correspondant au module M_{Clip} comme une variable externe du module M_{Arm} et réciproquement. Ainsi, l'avancée et l'ouverture ont toujours lieu simultanément.

1.2.6 Systèmes à états hiérarchiques

Le terme *système à états hiérarchiques* regroupe un ensemble de systèmes représentant de manière hiérarchique le comportement de plusieurs systèmes de transitions interagissant les uns avec les autres. La notion de *hiérarchie* a été popularisée par les Statecharts [Har87, HN96, Har98], les Modecharts [JM87] ou les diagrammes d'état-transition d'UML [BIJ97]. De manière synthétique, un état, dans un système à états hiérarchiques, est soit un état basique, soit un état *composite* représentant un ensemble d'états, c.à.d. un sous-système ou plusieurs sous-systèmes évoluant en parallèle. Le modèle des *automates hiérarchiques* proposé dans [MLS97] permet le dépliage du système à états hiérarchiques, c.à.d. sa transformation en ST2E. Dans [BLA+99], une méthode pour transformer un système à états hiérarchiques en un produit synchronisé de ST2E est proposée. Dans [AG00, AY01], la sémantique d'un système à états hiérarchiques est rapprochée de celle d'un système à variables partagées décrit comme un module.

> **Exemple.** La figure 1.15 représente un système à états hiérarchiques. Ce système fait réagir en parallèle deux sous-systèmes Arm et $Clip$ synchronisés par l'action FO pour obtenir le même comportement global que le produit synchronisé $Arm \times_{Syn} Clip$ donné figure 1.9.

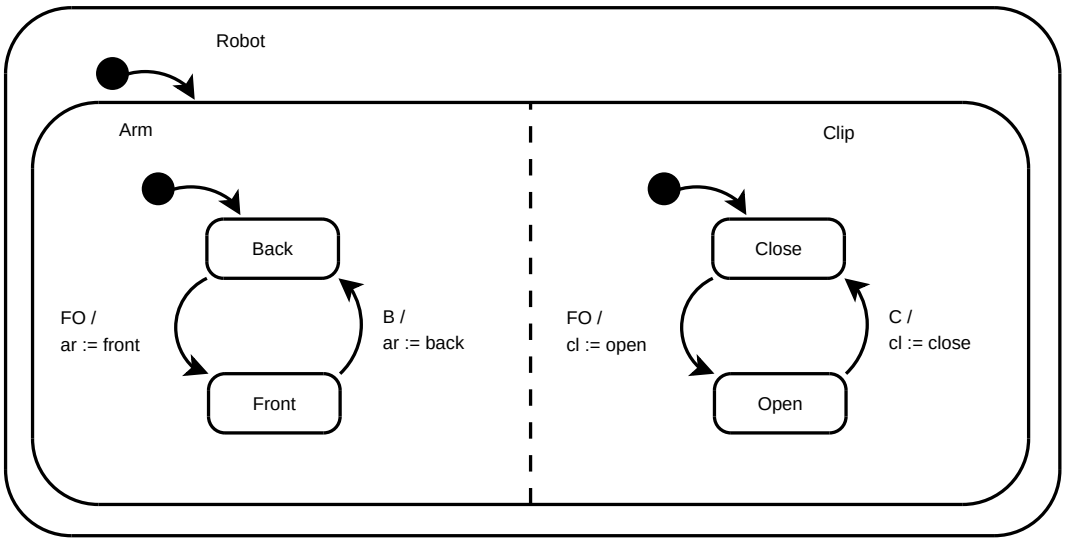


Fig. 1.15: Système à états hiérarchiques correspondant à $Arm \times_{Syn} Clip$

1.2.7 Les systèmes non-bloquants

Sifakis propose un cadre général pour la spécification de systèmes à composants : l'objectif recherché est de concevoir des systèmes non-bloquants à partir de composants non-bloquants [GS02, GS03b, GS03a, Sif05].

Le principe repose sur un modèle de composition en trois couches :

1. les différents composants du systèmes spécifiés par un ensemble de systèmes de transitions,
2. un modèle d'interaction qui spécifie toutes les interactions possibles entre les différents composants ; Ce modèle correspond l'ensemble de synchronisations du produit synchronisé
3. Un modèle de restriction qui limite le comportement du système complet grâce à un prédicat portant sur toutes les variables des composants.

Des conditions suffisantes sont données sur ce modèle de composition en trois couches pour garantir l'absence de blocages dans le système complet si les composants de départ étaient eux-même non-bloquants.

1.3 Les systèmes d'événements B : un exemple de spécifications de haut niveau

Pour spécifier des systèmes réels, un modèle donné sous la forme de ST2E est souvent difficile à réaliser. Des langages de plus haut niveau permettent de décrire des systèmes et prennent comme modèle sous-jacent des ST2E. L'objectif de cette section est de présenter un langage de spécification de haut niveau.

Le langage B est un langage de spécification logico-ensembliste introduit en 1996 par Abrial [Abr96a, ABL96, Abr97] et auquel est associé une méthodologie de développement combinant preuve et raffinement : la *méthode B*. L'idée consiste à développer par étapes successives – par raffinement – le système, depuis la spécification abstraite jusqu'à une dernière étape qui conduit à une traduction directe en une implantation et à valider par preuve chacune des étapes du processus (voir la section 2.1.3 pour plus de détails à propos du raffinement B). Il s'agit d'une des réussites les plus importantes de l'application des méthodes formelles dans un cadre industriel. Les applications réalisées à l'aide de la méthode B, par exemple le projet Météor, reposent sur une intégration de l'utilisation d'une méthode formelle dans un cycle de développement de de validation.

L'objectif de cette section est d'introduire de manière synthétique la méthode B. Cette section présente d'abord la méthode B classique, puis le B événementiel. Finalement, nous présenterons quelques travaux qui visent à mêler à B des notions de composition.

Machine M
Sets set_1 ; set_2 ; ...
Constants $const_1$, $const_2$, ...
Properties $proper(const_1, const_2)$
Variables var_1, var_2, ...
Invariant $inv(var_1, var_2)$
Initialisation $init(var_1, var_2)$
Operations
$ope \mathrel{\hat{=}}$ **pre** $pre(var_1, var_2)$ **then** $subst(var_1, var_2)$ **end**;
...
End

Fig. 1.16: Machine B abstraite

1.3.1 Le B classique

Dans la méthode B classique, le système est décrit par une *Machine*. Une spécification B est écrite à l'aide de structures de données basées sur la théorie des ensembles permettant de modéliser les données du modèle. La clause **Sets** permet de définir des ensembles set_1, set_2 (énumérés ou non) que va manipuler la machine. Des constantes $const_1$, $const_2$ (clause **Constants**), et des variables var_1, var_2 (clause **Variables**) définissent l'état du système.

Les propriétés et l'invariant du système sont donnés en logique du premier ordre, étendue à la théorie des ensembles. La clause **Properties** permet d'exprimer des propriétés (typage, domaine, etc.) à propos des constantes $const_1$, $const_2$ de la machine. Les variables var_1, var_2 sont contraintes par un **Invariant** inv, c.à.d. une conjonction de propriétés statiques, qui doit toujours être vérifiée par la machine.

L'initialisation et les opérations sont décrites à l'aide du langage des *substitutions généralisées*, une sorte d'extension du langage des commandes gardées de Dijkstra [Dij75, Dij76]. L'**Initialisation** $init$ initialise les variables var_1, var_2 de la machine. La clause **Operations** donne la liste des opérations réalisées par la machine. Chaque opération ope est en

fait une substitution généralisée *subst*, préconditionnée par un prédicat *pre*, et modifiant les variables var_1, var_2 de la machine.

Remarque. La syntaxe précise de B et les règles de réécriture des substitutions généralisées peuvent être trouvées dans [Abr96a]. Nous noterons simplement que $[subst]P = wp(subst, P)$ est le prédicat obtenu par l'application de la substitution *subst* au prédicat P, c.à.d. le calcul de la plus faible précondition garantissant que la substitution *subst* conduise au prédicat P.

Pour s'assurer de la cohérence d'une machine B, des schémas d'obligations de preuves, à la base de la méthode B, sont donnés dans [Abr96a]. Il s'agit de confronter la description statique de la machine – l'invariant *inv* – avec la description opérationnelle donnée par l'initialisation *init* et les opérations ope_1, ope_2, etc. Informellement,

- on doit s'assurer de la correction de l'initialisation. Autrement dit, l'invariant de la machine doit être valide après l'application des substitutions de l'initialisation, c.à.d. $[init]inv$;
- on doit aussi s'assurer que les opérations préservent l'invariant. Autrement dit, en prenant comme hypothèse que la machine est dans un état satisfaisant l'invariant avant l'appel de chaque opération *ope*, de substitution généralisée *subst* et de précondition *pre*, on doit prouver que l'invariant est maintenu après l'application de *subst*, c.à.d. que $pre \wedge inv \Rightarrow pre \wedge [subst]inv$.

1.3.2 Le B événementiel

Une extension du langage B proposée par Abrial dans [Abr96b] concerne la définition de *systèmes d'événements*. Le B événementiel est plus adapté à la modélisation de systèmes fermés réactifs que le B classique.

La notion d'*événement gardé* est introduite : chaque événement est composé d'une garde $grd(subst)$, calculée à partir de la substitution généralisée *subst* de l'événement en question. A tout moment, le système réagit avec l'environnement. Chaque événement dont la garde est vérifiée se déclenche spontanément en modifiant les variables concernées par son action. Si plusieurs gardes sont vraies, un des événements, choisi de manière non-déterministe, se produit nécessairement. Le modèle est bloqué si aucune garde d'aucun événement n'est vraie.

En B événementiel, la clause **Machine** est remplacée par la clause **Event System** et la clause **Operations** par la clause **Events** qui donne la liste des événements du système. Des règles de transformation permettent de se ramener d'une spécification en B événementiel à une spécification en B classique.

> **Exemple.** La figure 1.17 donne une spécification en B événementiel qui correspond à l'exemple du bras mobile robotisé *Robot* présenté dans la section 1.1.4 sous la forme d'un ST2E.
>
> Trois ensembles énumérés CLIP = {open, close}, ARM = {front, back} et ELEVATOR = {up, mid, down} sont définis et trois variables *cl*, *ar* et *el* sont

Event System
Robot
Sets
CLIP = {open, close} ;
ARM = {front, back} ;
ELEVATOR = {up, mid, down}
Variables
cl, ar, el
Invariant
cl ∈ CLIP ∧ ar ∈ ARM ∧ el ∈ ELEVATOR
Initialisation
cl := close || ar := back || el := down
Events
O ≙ **select**
cl = close ∧ ((ar = back ∧ el = up)
∨ (ar = front ∧ el = down))
then
cl := open
end ;
C ≙ **select**
cl = open ∧ ((ar = back ∧ el = down)
∨ (ar = front ∧ el = up))
then
cl := close
end ;
F ≙ **select**
ar = back ∧ ((cl = close ∧ el = down)
∨ (cl = open ∧ el = up))
then
ar := front
end ;
B ≙ **select**
ar = front ∧ ((cl = close ∧ el = up)
∨ (cl = open ∧ el = down))
then
ar := back
end ;
BC ≙ **select**
ar = front ∧ cl = open ∧ el = down
then
ar := back || cl := close
end ;
OF ≙ **select**
ar = back ∧ cl = close ∧ el = up
then
ar := front || cl := open
end ;
BCU1 ≙ **select**
ar = front ∧ cl = open ∧ el = down
then
ar := back || cl := close || el := mid
end ;
U1 ≙ **select**
el = down ∧ ar = back ∧ cl=close
then
el := mid
end ;
D2 ≙ **select**
el = mid ∧ ar = back ∧ cl=close
then
el := down
end ;
U2 ≙ **select**
el = mid ∧ ar = back ∧ cl=close
then
el := up
end ;
D1 ≙ **select**
el = up ∧ ar = back ∧ cl=close
then
el := mid
end
End

Fig. 1.17: Système d'événements SE_{Robot} correspondant à *Robot*

déclarées qui correspondent aux variables du ST2E *Robot*. Dans cet exemple, l'invariant indique uniquement le typage des variables.

Chacun des événements de la spécification B proposée correspond à une ou plusieurs des transitions dans le ST2E *Robot*. Le nom de l'événement B correspond à l'étiquette de la transition correspondante dans le ST2E. Pour chacun des événements, il est activable uniquement si le prédicat de la clause **select** est vérifié. Dans ce cas, la ou les substitutions de la clause **then** sont appliquées et les variables du système sont modifiées en conséquence, ce qui correspond bien à un changement d'état dans le ST2E *Robot*.

Plusieurs outils aussi bien industriels qu'académiques permettent de spécifier des systèmes à l'aide du langage B puis de vérifier la cohérence de la spécification, c.à.d de prouver (automatiquement ou non) les obligations de preuve garantissant cette cohérence : l'*Atelier B* [Ate], *B4free* [B4f] sa version académique, le *B-Toolkit* [Bto] ou encore *Barvey* [CDGR04b, CDGR04a]. *Click'n'Prove* [AC] est une interface d'aide à la preuve interactive pour l'Atelier B ou B4free. *ProB* [LB03, LBa] est un outil de model-checking de spécifications B.

Il a été montré qu'il était possible d'associer à un système d'événements B SE un système de transitions étiquetées S [BC00, CG04, PS04]. Un outil, *GeneSyst* a été développé au LSR-IMAG et permet, dans certains cas, de générer à partir d'une spécification B, le ST2E associé [MPS04].

1.3.3 Composition de spécifications B

La composition n'est pas le mode de développement privilégié de la méthode B. Néanmoins, certains travaux s'attachent à introduire une notion de "compositionnalité" au sein de B, afin de pouvoir spécifier une machine ou un système d'événements B sous la forme de machines ou de systèmes d'événements composants [But00, TS99, ST02, Abr02, Att02].

- Une approche proposée par Butler [But00] et Treharne et Schneider [TS99, ST02] consiste à utiliser une *algèbre de processus* pour contrôler les interactions entre plusieurs machines B. L'algèbre de processus CSP utilisée dans [TS99, ST02] est un langage de haut niveau pour décrire les différents processus d'un système concurrent, ainsi que leurs interactions sous la forme d'échanges de messages. Les processus peuvent être synchronisés et la communication s'effectue par l'intermédiaire de canaux de communication.
  Dans [ST02] un sous-ensemble du langage CSP est utilisé pour décrire des *contrôleurs* pour chacune des machines B. Chaque machine B MA_i du modèle est associée à un contrôleur CSP C_i : on parle alors de *machines contrôlées* $MA_i \| C_i$. Une machine donnée ne peut communiquer qu'avec son contrôleur. Les échanges entre machines s'effectuent donc forcément par l'intermédiaire des contrôleurs.

Le système global correspond à la combinaison de toutes les machines contrôlées $MA_i \| C_i$. Pour s'assurer de la consistance du système complet,
- il faut s'assurer que chaque contrôleur C_i est consistant avec sa machine B MA_i. Pour cela, il suffit de s'assurer que la machine contrôlée $MA_i \| CS_i$ ne comporte pas de divergence ;
- il faut vérifier l'absence de blocage de la combinaison de tous les contrôleurs CS_i.

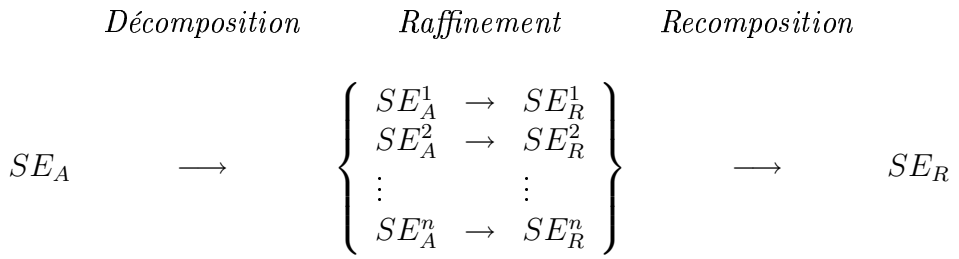


Fig. 1.18: Décomposition d'un système d'événements B

- Abrial propose [Abr02] de spécifier un système abstrait, puis de réaliser sa *décomposition*. Cette méthode s'inscrit dans le cadre d'une démarche de raffinement (voir la section 2.1.3 à propos du raffinement de systèmes d'événements B). Les sous-systèmes issus de la décomposition du système d'événements "abstrait" seront raffinés indépendamment les uns des autres (voir figure 1.18). Si la décomposition est correcte, on pourra conclure à propos du raffinement du système complet.

- Dans [Att02], une démarche de composition de systèmes d'événements B est proposée. La composition est traitée comme une composition parallèle avec l'introduction de variables partagées, un peu à la manière de [Abr02]. De manière plus précise, l'espace de variables est partitionné entre les variables *locales* à un système d'événement et les variables globales à l'ensemble des systèmes à composer. Des règles de réécriture sont exprimées pour construire à partir de deux systèmes d'événements SE^1 et SE^2 avec variables partagées, le système d'événements $SE^1 \| SE^2$ résultant : pour cela, une nouvelle obligation de preuve est ajoutée qui permet de s'assurer que les invariants de SE^1 et de SE^2 ne sont pas contradictoires.

1.4 Conclusion

Dans ce chapitre, nous avons présenté un certain nombre de notions préliminaires par rapport aux modèles des systèmes que nous allons manipuler dans la suite de ce document. Nous avons choisi les systèmes de transitions doublement étiquetés (ST2Es) pour caractériser en termes de traces et d'exécutions aussi bien le modèle des systèmes (STE) que celui des propriétés (SK).

Nous avons ensuite introduit des opérateurs de composition classiques au niveau du modèle choisi, celui des ST2Es : produit cartésien, puis produit *synchronisé*. Nous avons

ensuite montré que la majorité des compositions pouvaient être vues comme des extensions du produit synchronisé, qu'il s'agisse du produit synchrone, du produit asynchrone, des compositions par étiquettes de transitions communes ou des systèmes communicants. La composition par variables partagées et les systèmes à états hiérarchiques peuvent être rapprochés du produit synchronisé, ce qui justifie notre choix de modèle et de composition. Dans le chapitre 5, nous étendrons la notion de produit synchronisé pour définir les systèmes à composants synchronisés.

Finalement, nous avons introduit le langage B, et plus particulièrement les systèmes d'événements B, comme un exemple de langage de haut niveau, qui a comme modèle sous-jacent celui des ST2Es, afin d'indiquer que la démarche présentée dans la suite de ce document dépasse le cadre des ST2Es et peut être transposée à d'autres domaines.

L'exemple du bras mobile robotisé introduit dans ce chapitre nous sert de fil conducteur pour illustrer les résultats présentés dans l'ensemble de ce document.

Chapitre 2

Spécification par raffinement

Le raffinement est un processus de développement et de vérification. Il s'agit de remplacer dans une spécification les éléments de haut niveau par des constructions plus proches du système réel, en allant "de l'abstrait vers du concret" selon Abrial [Abr84]. D'une manière générale, le terme "raffinement" est utilisé pour qualifier différentes notions parmi lesquelles :

- *le raffinement de données*, qui consiste à remplacer les structures de données abstraites *mathématiques* (ensembles, relations, fonctions, etc.) par des structures plus concrètes et plus proches de celles de l'implantation (tableaux, scalaires, fichiers, etc.),
- *le raffinement d'algorithmes*, qui consiste à détailler les algorithmes utilisés tout en leur permettant de manipuler les données raffinées,
- *le raffinement temporel*, qui consiste à réduire la "granularité" du temps. Alors qu'au niveau abstrait, on n'observait qu'une seule action atomique, en "raccourcissant" les unités de temps, on rend observable de nouvelles actions qui détaillent le comportement de notre système.

Le raffinement est un processus de conception qui propose de passer d'une spécification abstraite SA à une spécification plus concrète SR en ajoutant des détails, mais c'est aussi un processus de vérification. En effet, il est nécessaire de s'assurer de la "correction" du raffinement, pour affirmer qu'une spécification concrète donnée est bien une vue plus détaillée de la spécification abstraite de départ.

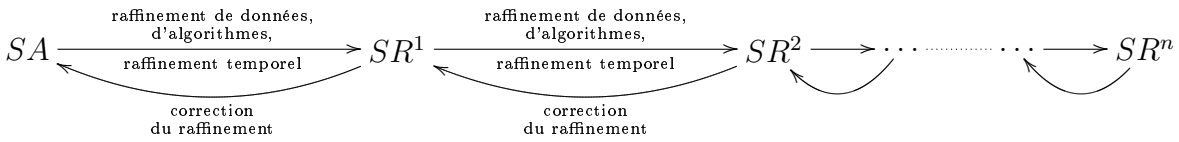


Fig. 2.1: Raffinement : processus de conception et de vérification

La section 2.1 sera l'occasion d'étudier certaines notions classiques de raffinement afin d'illustrer la diversité des définitions possibles du raffinement. Dans la section 2.2, nous présentons de manière détaillée certains travaux antérieurs du LIFC à propos de la définition du raffinement en termes d'une *relation de τ-simulation* particulière entre deux systèmes de transitions finis abstrait SA et raffiné SR. L'intérêt de cette définition pour des systèmes de transitions finis est qu'il est possible de s'assurer de la correction du raffinement, et cela, de manière complètement automatique en parcourant conjointement SR et SA. Cette relation de raffinement est à la base des travaux présentés dans les chapitres 4, 5 à propos de raffinements (dé)compositionnels.

2.1 Différentes notions de raffinement

Nous allons présenter dans cette section certaines notions classiques de raffinement afin d'illustrer la diversité des concepts manipulés ainsi que des méthodes de vérification du

raffinement.

Classiquement, ce sont les travaux de Dijkstra [Dij72, Dij75, Dij76] à propos de la corrections des programmes qui sont souvent cités comme point de départ du raffinement. La notion de raffinement a ensuite été formalisée par Back [Bac78, Bac88, BVW98] pour être ensuite adaptée et intégrée à différents cadres de spécification et/ou de vérification : citons, à titre d'illustration, les travaux d'Abadi et Lamport [AL88, AL91, Lam90, Lam94, Lam96] à propos du raffinement TLA, ceux de Morris [Mor87] ou de Morgan [Mor90], ceux de Pnueli [Pnu92] à propos du raffinement LTL, ceux de Butler [But96] sur le raffinement dans les systèmes d'actions, ceux d'Abrial [Abr96a, Abr96b] pour le raffinement de systèmes B ou encore ceux de Davies, Derrick et Boiten pour le raffinement de spécifications Z [DW96, DB01].

Dans la suite de cette section, nous présenterons de manière plus détaillée certains des travaux cités précédemment.

2.1.1 Raffinement wp de Dijkstra

Comme nous l'avons déjà précisé, c'est Dijkstra qui le premier a parlé de raffinement. Pour s'assurer de la correction d'un programme, il a défini le *langage des commandes gardées* et a introduit la notion de *plus faible précondition* dans [Dij76]. Cette notion s'appuie sur la logique de Hoare, appelée aussi logique des programmes. Une instruction annotée d'un programme en logique de Hoare est un triplet $< pre > instr < post >$, où $instr$ est une instruction, pre une précondition et $post$ une postcondition.

L'opérateur $wp(instr, post)$ pour *weakest precondition* permet de calculer la plus faible des conditions garantissant qu'une instruction $instr$ va se terminer, afin de laisser le système dans un état satisfaisant la postcondition $post$. Ainsi, une instruction annotée $< pre > instr < post >$ est correcte si $pre \Rightarrow wp(instr, post)$.

Le raffinement wp de Dijkstra s'exprime en termes de plus faible précondition de la manière suivante.

Définition 2.1 (Raffinement wp)
Une instruction $instr_A$ sera raffinée par une instruction $instr_R$ si pour tout prédicat P, $wp(instr_A, P)$ implique $wp(instr_R, P)$.

La sémantique du raffinement en termes de plus faible précondition a servi de base pour de nombreux raffinements, en autres pour le raffinement des systèmes d'actions [But96], le raffinement Z [DW96, DB01] et le raffinement B [Abr96a].

2.1.2 Raffinement dans les systèmes d'actions

Les systèmes d'actions de Back [BKS83, BVW98] puis Butler [But96] sont une version étendue du langage des commandes gardées de Dijkstra, fondés sur le calcul des plus faibles préconditions wp.

Informellement, un système d'actions est un quadruplet $AS = (E, v, A_i, A)$ où E est un ensemble d'étiquettes d'actions, v est une variable d'état (ou un ensemble de variables), A_i est l'action d'initialisation et A est un ensemble d'actions gardées. Une action $A_e \in A$ est de la forme $e : - \ grd(A_e) \rightarrow subst(A_e)$ et elle est activable si l'état courant du système satisfait la garde $grd(A_e)$ de l'action. Dans ce cas, la substitution $subst(A_e)$ est appliquée.

$$AS_{Clip} \hat{=} \left(\begin{array}{ll} \textbf{var } cl : \{close, open\} & \\ \textbf{initially } cl := close & \\ \textbf{action } O : - & cl = close \rightarrow cl := open \\ \textbf{action } C : - & cl = open \rightarrow cl := close \end{array} \right)$$

FIG. 2.2: Système d'actions correspondant au ST2E *Clip*

> **Exemple.** Nous donnons pour illustrer les systèmes d'actions, l'exemple d'un système d'actions AS_{Clip} spécifiant une pince. Ce système d'actions correspond au ST2E *Clip* donné Figure 1.7(b), section 1.2. AS_{Clip} est constitué de deux actions O et C, et d'une variable d'état cl indiquant l'état de la pince. Initialement, cl est dans l'état *close*, donc l'action O est la seule action activable. L'exécution de O place cl à *open* et seule l'action C peut être activée. L'exécution de C ramène dans l'état initial.

Soient P et P' deux prédicats. On écrira $P \Rrightarrow P'$, si tout état satisfaisant P satisfait également P'. Le raffinement d'actions est défini dans [But96] de la manière suivante : une action R_e raffine une action A_e, noté $A_e \preceq R_e$, si pour tout prédicat P, $wp(subst(A_e), P) \Rrightarrow wp(subst(R_e), P)$. Notons que cette expression du raffinement est similaire au raffinement wp de Dijkstra.

Dans une démarche de raffinement, les variables abstraites sont habituellement remplacées par des variables plus concrètes. Un invariant d'abstraction *abs* exprime le lien qu'il y a entre une variable d'état v_A d'un système d'actions abstrait AS_A et une variable d'état v_R d'un système d'actions plus concret AS_R. Le raffinement d'actions s'exprime alors de façon un peu plus complexe, de la manière suivante :
une action R_e raffine, sous invariant d'abstraction *abs*, une action A_e, noté $A_e \preceq_{abs} A_R$, si pour tout prédicat P indépendant de v_R, il existe v_A tel que $abs \wedge wp(subst(A_e), P) \Rrightarrow wp(subst(R_e), \exists v_A.\ (abs \wedge P))$.

Le raffinement d'un système d'actions complet est alors défini en termes de simulation sous invariant d'abstraction entre le système d'actions abstrait et son homologue raffiné [But96]. Une relation de simulation sous invariant d'abstraction *abs* est une relation entre deux systèmes d'actions AS_A et AS_R telle que, pour tout $e \in E$,
– $A_e \preceq_{abs} R_e$,
– $\exists v_A.\ (abs \wedge grd(A_e)) \Rrightarrow grd(R_e)$.
La première condition assure que chaque action de AS_R raffine son homologue dans AS_A. La deuxième condition assure quant-à elle que AS_R n'introduit pas de nouveaux blocages.

Remarque. Remarquons qu'on ne peut pas introduire de nouvelles actions lors du raffinement d'un système d'actions.

2.1.3 Raffinement B

Comme nous l'avons indiqué dans la section 1.3, la méthode B s'accompagne d'un processus de développement par raffinement [Abr96a, Abr96b, AM97].

Pour écrire une spécification B raffinée, la syntaxe est quasiment identique à celle donnée section 1.3 à propos des spécifications abstraites. Quelques clauses sont ajoutées comme la clause **Refinement** qui remplace la clause **Event System** ou la clause **Refines** qui indique le système dont on spécifie un raffinement.

Considérons deux systèmes d'événements SE_A et SE_R tels que SE_A soit une spécification abstraite et SE_R une spécification plus concrète. Leurs ensembles de variables V_A et V_R doivent être disjoints. L'invariant inv_R du système raffiné doit décrire – entre autres choses – le lien qu'il doit y avoir entre les variables abstraites V_A et les variables plus concrètes V_R. Cet invariant est appelé *invariant de collage* dans la terminologie B.

Le raffinement d'une substitution B est similaire au raffinement d'actions sous invariant d'abstraction de Butler (voir section précédente) : on dira qu'une substitution $subst_A$ est raffinée par une substitution $subst_R$ si $inv_A \wedge inv_R \Rightarrow [subst_R]\neg[subst_A]\neg(inv_A \wedge inv_R)$. L'application de la substitution *subst* au prédicat P, c.à.d. le calcul de la plus faible précondition $wp(subst, P)$ garantissant que la substitution *subst* conduise au prédicat P se note $[subst]P$ en B.

Un système d'événements raffiné SE_R peut introduire des événements qui n'existaient pas au niveau abstrait. Ces événements qualifiés de "nouveaux" doivent raffiner la substitution identité, notée *skip*. Le raffinement de systèmes d'événements B peut alors être vu comme une sorte de *stuttering refinement* au même titre que le raffinement TLA [Lam96].

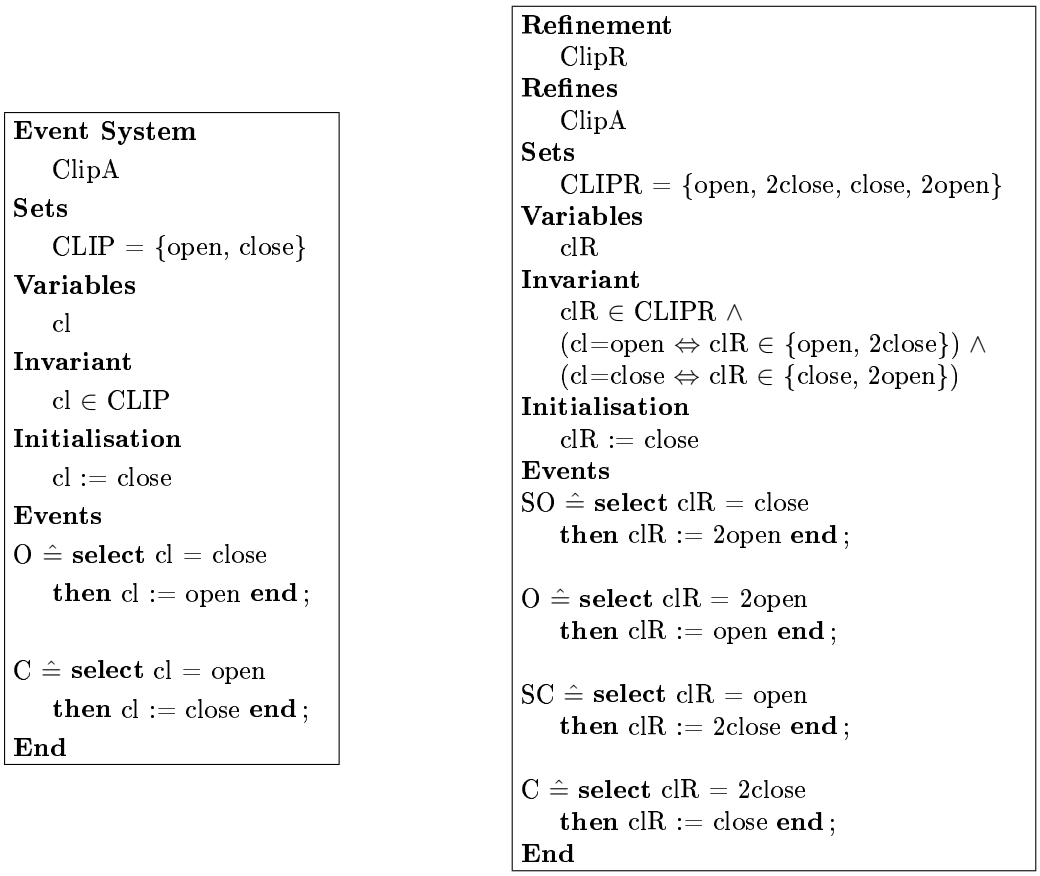
La méthode B indique des obligations de preuve à vérifier afin de s'assurer conjointement de la cohérence du système raffiné SE_R et de la validité du raffinement de SE_A par SE_R :

1. L'initialisation abstraite se raffine par l'initialisation concrète : $[init_R]\neg[init_A]\neg inv_R$.
2. Chaque événement abstrait ev_A est raffiné par son homologue concret ev_R : $inv_A \wedge inv_R \Rightarrow [ev_R]\neg[ev_A]\neg Inv_R$

Dans le cas des "nouveaux" événements new_R, la deuxième obligation de preuve se simplifie, puisque new_R doit raffiner *skip* : $Inv_A \wedge Inv_R \Rightarrow [new_R]Inv_R$,

Abrial et Mussat ajoutent deux autres conditions aux précédentes pour le raffinement d'un système d'événements afin de garantir la préservation des propriétés dynamiques introduites dans [AM98] (voir section 3.1.3) :

– Un nouvel événement new_R ne doit pas "prendre indéfiniment la main" afin de garantir que le système raffiné ne diverge pas plus que le système abstrait. Il est nécessaire



Event System
ClipA
Sets
CLIP = {open, close}
Variables
cl
Invariant
cl ∈ CLIP
Initialisation
cl := close
Events
O ≙ **select** cl = close
then cl := open **end**;

C ≙ **select** cl = open
then cl := close **end**;
End

Refinement
ClipR
Refines
ClipA
Sets
CLIPR = {open, 2close, close, 2open}
Variables
clR
Invariant
clR ∈ CLIPR ∧
(cl=open ⇔ clR ∈ {open, 2close}) ∧
(cl=close ⇔ clR ∈ {close, 2open})
Initialisation
clR := close
Events
SO ≙ **select** clR = close
then clR := 2open **end**;

O ≙ **select** clR = 2open
then clR := open **end**;

SC ≙ **select** clR = open
then clR := 2close **end**;

C ≙ **select** clR = 2close
then clR := close **end**;
End

Fig. 2.3: Systèmes d'événements SE_{Clip_A} et SE_{Clip_R} correspondant à *Clip*

de spécifier un *variant*, qui doit décroître strictement à chaque activation de chaque nouvel événement afin de garantir l'absence de cycles formés uniquement de nouveaux événements.
- Le système raffiné ne doit pas se bloquer plus souvent que le système abstrait afin de garantir que le raffinement n'introduit pas de nouveaux comportements bloquants.

> **Exemple.** A titre d'exemple de raffinement d'un système d'événements B, nous donnons deux spécifications correspondant à une pince. Le système d'événements SE_{Clip_A} décrit une vue abstraite d'une pince et correspond au ST2E *Clip*, donné figure 1.7(b), section 1.2. Le système d'événements SE_{Clip_R} représente un raffinement de SE_{Clip_A} où deux nouveaux événements SO et SC viennent s'intercaler entre O et C.
>
> Grâce à des outils comme l'Atelier B, on peut facilement montrer que le raffinement est correct en vérifiant toutes les obligations de preuve nécessaires.

Remarque. Le raffinement de systèmes d'événements B a inspiré les travaux à propos du raffinement de systèmes de transitions finis défini comme une relation de τ-simulation que nous présentons dans la section 2.2 (voir aussi [BJM99, BJK00]).

Raffinement B et compositionnalité

Abrial propose dans [Abr02] une méthode basée sur la *décomposition* de systèmes d'événements B (voir aussi la section 1.3.3). Soit SE_A, un système d'événements B abstrait. La démarche proposée consiste à décomposer SE_A pour obtenir plusieurs sous-systèmes abstraits SE_A^1, SE_A^2, ..., SE_A^n. On pourra ensuite raffiner indépendamment chacun des sous-systèmes obtenus. Finalement, il sera possible de procéder à la recomposition des sous-systèmes raffinés (voir figure 2.4).

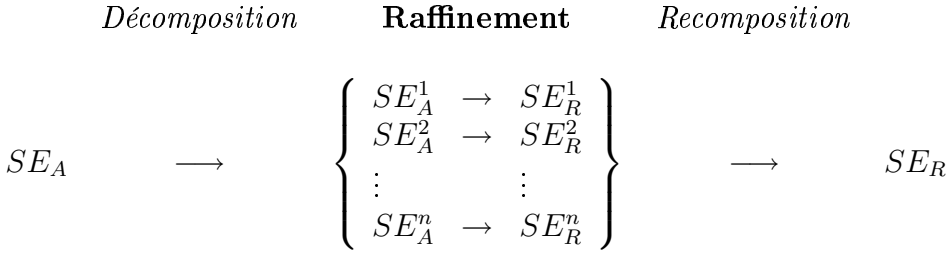


Fig. 2.4: Raffinement par décomposition d'un système d'événements B

La principale difficulté de la démarche de décomposition consiste, bien entendu, à établir une "bonne" décomposition du système abstrait SE_A : il s'agit en particulier de répartir les différentes variables et les différents événements de SE_A entre les sous-systèmes SE_A^1, SE_A^2, ..., SE_A^n. Afin de résoudre ces deux difficultés, Abrial indique qu'il est nécessaire d'étendre le langage B. Il introduit pour cela des *variables partagées* afin de pouvoir répliquer une même variable dans plusieurs sous-systèmes. De la même manière, afin de pouvoir faire apparaître un événement dans plusieurs sous-systèmes, on pourra rencontrer des événements dits *externes*.

Il est ensuite possible de raffiner chacun des sous-systèmes, avec quelques restrictions : les variables partagées ne doivent pas être raffinées et les événements externes doivent forcément être raffinés dans l'un des sous-systèmes. Néanmoins, la décomposition d'un système d'événements demande une grande expertise, d'autant plus qu'il faut que la décomposition aboutisse à des sous-systèmes dont on est capable d'établir un raffinement. La compositionnalité concerne ici uniquement la spécification et non la vérification.

Les travaux de [Att02] sur la composition de modèles B en utilisant des variables partagées reposent sur les même contraintes de raffinement que celles d'Abrial : les variables partagées ne doivent pas être raffinées afin de garder un lien cohérent entre les composants.

2.1.4 Raffinement TLA

La logique temporelle des actions TLA (pour Temporal Logic of Actions), proposée par Lamport [Lam90, Lam94] permet de décrire des systèmes réactifs. Afin de décrire un système, on spécifie en TLA une formule caractérisant l'ensemble des traces pour lesquelles elle est vraie. TLA est la base de TLA+, un langage de spécification de systèmes réactifs structurés en modules [Lam].

Dans TLA, l'implication tient compte du bégaiement, c.à.d. qu'une spécification est satisfaite par un comportement σ si elle est satisfaite par tout comportement obtenu à partir de σ en ajoutant ou en enlevant des étapes qui laissent l'état inchangé. Le raffinement s'exprime alors directement en termes d'implication. Il consiste à "sélectionner" certaines traces, parmi toutes les traces de la spécification abstraite, modulo le bégaiement. Il s'agit donc d'un *stuttering refinement* [Lam96].

Notons, par ailleurs que l'outil TLC (TLA+ Model checker) peut être utilisé pour vérifier des spécifications avec différents niveaux de raffinements écrites en TLA+.

2.1.5 Raffinement LTL

Dans [Pnu92], Pnueli montre qu'il est possible d'exprimer la sémantique temporelle d'un système de transitions par une formule de Logique Temporelle Linéaire (LTL). La LTL est présentée en détail dans la section 3.1.2. On spécifiera le système ainsi que ses propriétés dans le même formalisme.

Le raffinement s'exprime alors très simplement : soient ϕ_A et ϕ_R deux formules LTL. Une spécification ϕ_R raffine une spécification ϕ_A si toutes les exécutions qui satisfont ϕ_R satisfont également ϕ_A, autrement dit, quand $\phi_R \Rightarrow \phi_A$.

L'avantage principal du raffinement LTL est la préservation des propriétés LTL par transitivité de l'implication. Si une spécification abstraite ϕ_A satisfait une propriété ψ (c.à.d. $\phi_A \Rightarrow \psi$)et que ϕ_A est raffinée par ϕ_R, alors ϕ_R satisfait aussi ψ. La plupart du temps, il est par contre difficile de vérifier l'implication $\phi_R \Rightarrow \phi_A$.

2.1.6 Raffinement CSP

Le langage CSP permet de spécifier le comportement de systèmes grâce à une algèbre de processus, c'est-à-dire une description de processus communiquant les uns avec les autres [Hoa85]. Comme il est indiqué dans [Ros97], la sémantique d'une algèbre de processus repose sur l'observation du comportement des processus. Trois modèles d'observation sont donnés :

- Le modèle des *traces* associe à un processus C l'ensemble des séquences finies d'actions que le processus peut effectuer, noté $traces(C)$. Ce modèle permet de représenter tous les comportements possibles du processus C.
- Le modèle des *échecs stables* associe à un processus C, des couples (σ, A), où σ est une trace finie admise par C et A l'ensemble des actions que C ne peut pas exécuter après avoir exécuté les actions de σ. L'ensemble de tous ces couples est noté $failures(C)$. Ce modèle permet de caractériser les blocages de C.
- Le modèle des *échecs-divergences* associe à un processus C l'ensemble de ses échecs stables et l'ensemble de ses divergences. L'ensemble des divergences, noté $divergences(C)$ est l'ensemble des traces σ telles que C se retrouve, après avoir exécuté les actions de σ, dans un état où il ne peut plus exécuter que des événements internes et inobservables.

Le raffinement dépend alors du modèle considéré, et se définit en termes d'inclusion de modèles. Soient C_A et C_R deux processus,
- dans le cas du modèle des traces, on dira que C_R raffine C_A si $traces(C_R) \subseteq traces(C_A)$,
- dans le cas du modèle des échecs stables, on dira que C_R raffine C_A si $failures(C_R) \subseteq failures(C_A)$,
- dans le cas du modèle des échecs-divergences, on dit que C_R raffine C_A si $failures(C_R) \subseteq failures(C_A)$ et $divergences(C_R) \subseteq divergences(C_A)$.

La vérification de chacune des formes du raffinement CSP est implémentée dans FDR [FDR93, FDR97], bien qu'elle soit limitée par la complexité des processus à analyser. Calculer l'ensemble $divergences(C)$ pour un processus C complexe conduit souvent à une explosion du nombre d'états du système.

Dans [Jos88], Josephs propose de ramener la sémantique des processus à celle des systèmes de transitions étiquetés. Deux relations de simulation consistantes par rapport au raffinement CSP sont définies. Considérons C_A et C_R, deux processus.
- C_A étant en relation avec C_R, la relation *forward* de Josephs exprime que C_A peut simuler le comportement de C_R tant que C_A et C_R sont dans des états correspondants,
- C_R étant en relation avec C_A, la relation *backward* exprime que si C_R atteint un certain état, alors C_A est capable de retracer la séquence d'actions jusqu'à trouver un état à partir duquel C_A peut simuler le comportement de C_R.

C_R raffine C_A si l'une des deux relations de simulation est vérifiée.

2.2 Raffinement des systèmes de transitions

Dans [BJM99, BJK00], les auteurs ont cherché à exprimer le raffinement des systèmes d'événements B en termes de relation de simulation entre deux systèmes de transitions finis. Soient $SA = (Q_A, Q_{0A}, E_A, T_A, V_A, l_A)$ et $SR = (Q_R, Q_{0R}, E_R, T_R, V_R, l_R)$ deux ST2E.

Syntaxiquement le raffinement de systèmes de transitions finis se traduit par les conditions suivantes :

1. le raffinement peut introduire de nouvelles actions donc $E_A \subseteq E_R$,
2. le raffinement peut introduire de nouvelles variables, quant aux anciennes variables, elles sont renommées. On doit avoir $V_A \cap V_R = \varnothing$,
3. un *prédicat de collage*[1] *gp* exprime le lien entre les variables du système abstrait et celles du système raffiné.

2.2.1 Définition de la relation de raffinement

Avant de définir le raffinement de systèmes de transitions en termes de relation de simulation, il faut, dans un premier temps, définir formellement le lien qu'il y a entre variables

[1] Ce prédicat de collage est aussi appelé *invariant de collage* comme en B.

abstraites et variables raffinées. Soit $AP_{V_A \cup V_R}$ l'ensemble des propositions atomiques définies sur $V_A \cup V_R$. Un prédicat de collage gp est une formule d'état[2] définie sur $SP_{V_A \cup V_R}$, l'ensemble des formules d'état de $V_A \cup V_R$. Dans ces conditions, le lien entre les variables des deux systèmes se transpose au niveau des états des deux systèmes : nous définissons une relation binaire, dite "de collage", $\mu \subseteq Q_R \times Q_A$ qui lie les états des deux systèmes.

Définition 2.2 (Relation de collage)
Soient SA et SR deux ST2E. Soit $gp \in SP_{V_A \cup V_R}$ un prédicat de collage. Un état $q_R \in Q_R$ du système raffiné est "collé" à un état $q_A \in Q_A$ du système abstrait, noté $q_R \ \mu \ q_A$, si

$$\bigwedge_{ap_i \in l_A(q_A) \cup l_R(q_R)} ap_i \Rightarrow gp$$

Afin d'identifier plus facilement les *nouvelles* transitions introduites par le raffinement, nous les renommerons par τ, et nous les considérerons comme des transitions "non-observables", i.e., des τ-transitions. Nous définissons formellement le renommage par τ des nouvelles transitions dans le ST2E raffiné de la manière suivante.

Définition 2.3 (Renommage par τ)
Soient SA et SR deux ST2E. Le ST2E SR dans lequel le renommage par τ a été réalisé est défini ainsi :

$$\mathcal{T}_{SA}(SR) = \mathcal{R}(SR, E_R \setminus E_A, \tau)$$

Remarque. Nous nous contenterons de noter $\mathcal{T}(SR)$ quand il sera clair que le renommage par τ des nouvelles transitions du ST2E SR est effectué dans le contexte d'un ST2E abstrait SA.

Nous allons définir le raffinement entre SR et SA, en introduisant successivement différentes conditions syntaxiques et en définissant, à chacune des étapes, la plus grande relation binaire R incluse dans μ qui satisfait la nouvelle condition introduite (ainsi que les précédentes).

Remarque. Nous travaillons sur des ensembles d'états Q_R et Q_A finis. Considérons une fonction $\mathcal{F}$ de $\mathbb{P}(Q_R \times Q_A)$ définie par $R \subseteq Q_R \times Q_A$. La forme des conditions utilisées pour définir R garantit que $\mathcal{F}$ est *monotone* et qu'elle admet donc un *plus grand point fixe*. Nous sommes donc assurés de l'existence de la plus grande relation binaire R vérifiant la condition considérée.

Les conditions que nous allons introduire pour le raffinement sont les suivantes :

1. les transitions de SR étiquetées dans E_A, c.à.d. les *anciennes* transitions, sont aussi observables dans SA,
2. les nouvelles transitions introduites par le raffinement et renommées par τ sont considérées comme des transitions non-observables raffinant "skip",
3. les nouvelles transitions ne doivent pas introduire plus de comportements bloquants qu'il n'y en avait dans le système abstrait,

[2]Voir aussi la section 3.1.1 à propos de la satisfaction d'une formule d'état.

4. les nouvelles transitions ne doivent pas prendre le contrôle indéfiniment. On interdit donc tous les cycles formés uniquement de τ-transitions, i.e., τ-cycles.

Nous souhaitons en premier lieu que tous les comportements observables dans le système raffiné SR, soient aussi des comportements observables dans SA. Nous définissons une relation de *simulation* entre SR et SA, adaptée à notre contexte (voir figure 2.5).

Définition 2.4 (Relation de simulation)
Soient SA et SR deux ST2E et μ la relation de collage. Soit $e \in E_A$. La relation de simulation γ est la plus grande relation binaire incluse dans μ qui vérifie :

$$\forall q_R \in Q_R, \forall q_A \in Q_A.\ \ (q_R\ \gamma\ q_A\ \wedge\ q_R \xrightarrow{e} q'_R) \Rightarrow \exists q'_A.\ (q_A \xrightarrow{e} q'_A\ \wedge\ q'_R\ \gamma\ q'_A)$$

Nous dirons que SA simule SR, noté $SR \preceq_\gamma SA$, si $\forall q_R.(q_R \in Q_R \Rightarrow \exists q_A.(q_A \in Q_A \wedge q_R\ \gamma\ q_A))$.
La relation de simulation définie précédemment est comparable, modulo la relation de collage, à celles définies par Milner [Mil71], Park [Par81] ou Clarke [CGP00].

Le raffinement introduit de nouvelles actions que nous rendrons inobservables en les masquant par τ. Nous étendons donc la relation de simulation précédemment définie afin de tenir compte des τ-transitions et nous définissons une τ-simulation (voir figure 2.6). Intuitivement, une transition $q_A \xrightarrow{e} q'_A$ sera raffinée par une partie finie de chemin dont la trace est $\tau^*.e$.

Définition 2.5 (Relation de τ-simulation)
Soient SA et SR deux ST2E et γ la relation de simulation. La relation de τ-simulation ϱ est la plus grande relation binaire incluse dans γ qui vérifie :

$$\forall q_R \in Q_R, \forall q_A \in Q_A.\ \ (q_R\ \varrho\ q_A\ \wedge\ q_R \xrightarrow{\tau} q'_R) \Rightarrow (q'_R\ \varrho\ q_A)$$

Nous dirons que SA τ-simule SR, noté $SR \preceq_\varrho SA$, si $\forall q_R.(q_R \in Q_R \Rightarrow \exists q_A.(q_A \in Q_A \wedge q_R\ \varrho\ q_A))$.
La relation ϱ définit une τ-simulation particulière qui diffère légèrement de la *simulation observationnelle* de Milner [Mil89] : chez Milner, la trace raffinée $\tau^*.e$ est remplacé par $\tau^*.\alpha.\tau^*$ avec $\alpha \in E_A \cup \{\epsilon\}$ où ϵ caractérise l'action vide. La τ-simulation ϱ correspond également à la $\tau^*.a$-bisimulation définie par Fernandez [Fer90]. La relation de τ-simulation ϱ nous permet de définir un *stuttering refinement* (raffinement par bégaiement) comme dans le cas du raffinement TLA ou celui des systèmes d'événements B.

Nous étendons encore la relation de τ-simulation ϱ afin de la rendre "non-bloquante", c.à.d. que les nouvelles transitions, renommées par τ ne doivent pas introduire plus de blocages qu'il n'y en avait dans le système abstrait (voir figure 2.7). L'objectif est de garantir que le système raffiné n'est pas plus bloquant que ne l'est le système abstrait. Néanmoins, il faut noter que le système abstrait pouvant être bloquant, alors le système raffiné aussi.

Définition 2.6 (Relation de τ-simulation non-bloquante)
Soient SA et SR deux ST2E et ϱ la relation de τ-simulation. La relation de τ-simulation non-bloquante ρ est la plus grande relation binaire incluse dans ϱ qui vérifie :

$$\forall q_R \in Q_R, \forall q_A \in Q_A.\ \ (q_R\ \rho\ q_A\ \wedge\ q_R \nrightarrow) \Rightarrow (q_A \nrightarrow)$$

Nous dirons que SA τ-simule SR de manière non-bloquante, noté $SR \preceq_\rho SA$, si $\forall q_R.(q_R \in Q_R \Rightarrow \exists q_A.(q_A \in Q_A \wedge q_R\ \rho\ q_A))$.

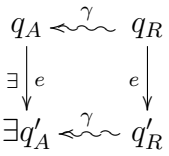


Fig. 2.5: Relation de simulation

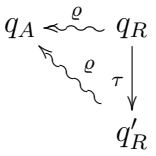


Fig. 2.6: Relation de τ-simulation

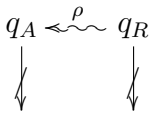


Fig. 2.7: Relation de τ-simulation non-bloquante

Nous pouvons maintenant définir une relation de raffinement entre deux ST2E finis. Nous ajoutons à la relation de τ-simulation non-bloquante une dernière clause qui garantit l'absence de τ-divergence : les nouvelles transitions introduites par le raffinement ne prendront pas le contrôle indéfiniment.

Définition 2.7 (Relation de raffinement [BJK00])
Soient SA et SR deux ST2E et ρ la relation de τ-simulation non-bloquante. La relation η est la plus grande relation binaire incluse dans ρ qui vérifie :

$$\forall q_R \in Q_R, \forall q_A \in Q_A.\ \ (q_R\ \eta\ q_A) \Rightarrow \forall \sigma.(\sigma \in \Sigma(q_R) \Rightarrow tr(\sigma) \neq \tau^\omega)$$

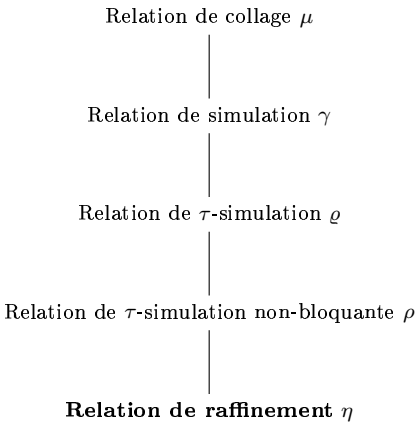


Fig. 2.8: Diagramme d'inclusion des relations définissant le raffinement

Remarque. La figure 2.8 donne le diagramme d'inclusion des différentes relations définissant le raffinement.

La relation de raffinement précédente notée η Est présentée dans [BJK00] comme la plus grande relation binaire η_0 incluse dans la relation de collage μ qui vérifie quatre conditions correspondant aux différentes relations de simulation présentées ici :
- *Raffinement strict* : $(q_R\ \eta\ q_A\ \wedge\ q_R \xrightarrow{e} q'_R) \Rightarrow \exists q'_A.\ (q_A \xrightarrow{e} q'_A\ \wedge\ q'_R\ \eta\ q'_A)$
- *Raffinement avec bégaiement* : $(q_R\ \eta\ q_A\ \wedge\ q_R \xrightarrow{\tau} q'_R) \Rightarrow (q'_R\ \eta\ q_A)$
- *Absence de nouveaux blocages* : $(q_R\ \eta\ q_A\ \wedge\ q_R \nrightarrow) \Rightarrow (q_A \nrightarrow)$
- *Absence de τ-divergence* : $(q_R\ \eta\ q_A) \Rightarrow \forall \sigma.(\sigma \in \Sigma(q_R) \Rightarrow tr(\sigma) \neq \tau^\omega)$

Les relations η et η_0 sont équivalentes.

Définition 2.8 (Raffinement [BJK00])
Soient SA et SR deux ST2E. Nous dirons que SR raffine SA, noté $\mathcal{T}_{SA}(SR) \sqsubseteq_\eta SA$ si

$$\forall q_R.\ (q_R \in Q_R \Rightarrow \exists q_A.\ (q_A \in Q_A\ \wedge\ q_R\ \eta\ q_A))$$

La relation de raffinement η peut aussi être vue dans le spectre de Van Glabbeek [vG90] comme une simulation observationnelle préservant la divergence et la stabilité : *divergence stability respecting completed simulation.*

Remarque. Le lecteur intéressé par cette section trouvera plus de détails dans [Kou04].

2.2.2 Propriétés de la relation de raffinement

Raffinement de chemins

La relation de raffinement η peut être étendue aux chemins des systèmes considérés. Soient $\Sigma(SA)$ et $\Sigma(SR)$ les ensembles de chemins des deux systèmes de transitions SA et $\mathcal{T}_{SA}(SR)$. Si $\mathcal{T}_{SA}(SR) \sqsubseteq_\eta SA$, alors chaque chemin σ_R de $\Sigma(SR)$ raffine un chemin σ_A de $\Sigma(SA)$ (voir figure 2.9).

Proposition 2.1 (Raffinement de chemins)
Soient σ_A un chemin de SA et σ_R un chemin de $\mathcal{T}_{SA}(SR)$. On dira que σ_R raffine σ_A, noté $\sigma_R \sqsubseteq_\eta \sigma_A$, si $\sigma_R(0)\ \eta\ \sigma_A(0)$.

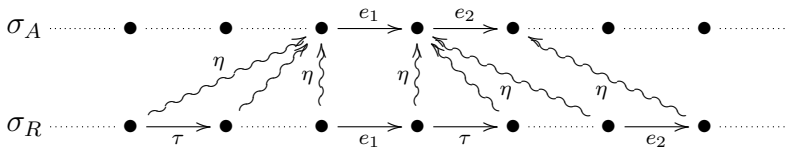


Fig. 2.9: Raffinement de chemins

Transitivité du raffinement

Pour la relation de raffinement η, le résultat de transitivité du raffinement a été établi et une preuve détaillée peut être trouvée dans [Cha03a, Kou04].

Théorème 2.1 (Transitivité du raffinement [Cha03a])
Soient S^1, S^2 et S^3 trois ST2E.

$$\frac{\mathcal{T}_{S^1}(S^2) \sqsubseteq_{\eta_1} S^1, \quad \mathcal{T}_{S^2}(S^3) \sqsubseteq_{\eta_2} S^2}{\mathcal{T}_{S^1}(S^3) \sqsubseteq_{\eta_1 \eta_2} S^1}$$

Puisque la relation de raffinement η est réflexive et transitive, il s'agit donc d'un pré-ordre et, puisqu'elle est antisymétrique, c'est donc un ordre partiel.

2.2.3 Vérification du raffinement

Il est connu que les relations de simulation et de τ-simulation sont calculables sur les systèmes de transitions finis. Le principe des algorithmes par analyse locale consiste à parcourir *en profondeur d'abord* un graphe composé, produit synchrone des deux systèmes de transitions à comparer ; ceci sans mémoriser les systèmes avant la vérification [Hol85]. Pour la relation de simulation, l'algorithme par analyse locale calcule un point fixe en dénombrant les relations binaires sur les ensembles d'états finis. Dans [FM90, FM91a], des algorithmes pour le calcul de relations d'équivalence et de pré-ordre sont proposés.

Une deuxième méthode pour comparer deux ST2E consiste à calculer une partition stable à partir d'une partition initiale, puisqu'une relation d'équivalence sur un ensemble peut être caractérisée comme une partition de cet ensemble compatible avec la relation de transition. Dans [PT87], un algorithme est proposé : celui-ci procède de manière similaire au calcul de l'automate minimal pour un automate déterministe (voir aussi [Fer89a, Fer90, FKM93]).

Par exemple, ALDEBARAN [Fer89a, Fer89b, FM91b] – qui fait partie de la boîte à outils CADP [FGM+92, GJM+97, GLM02, GL02a, VAS] – propose de calculer de manière efficace un certain nombre de relations de simulation entre STE. BISIMULATOR [Mat03, BDJM05] – qui fait aussi partie de la boite à outils CADP – propose aussi de calculer plusieurs relations d'équivalence, dont la $\tau^*.a$-bisimulation [Fer90].

La relation de raffinement η peut être calculée en adaptant l'algorithme de vérification par analyse locale proposé dans [FM90]. La vérification du raffinement consiste alors à une énumération conjointe des deux systèmes de transitions, afin de dénombrer les relations binaires sur l'ensemble $Q_R \times Q_A$, et chercher, parmi elles, une relation de raffinement vérifiant toutes les conditions et contenant les paires $Q_{0R} \times Q_{0A}$.

Une procédure de vérification du raffinement est implantée dans SynCo [KL, Lan04b] (voir chapitre 6).

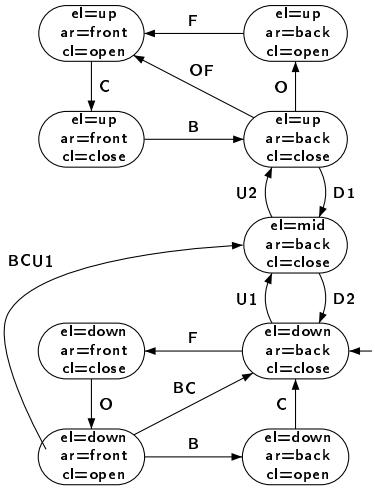
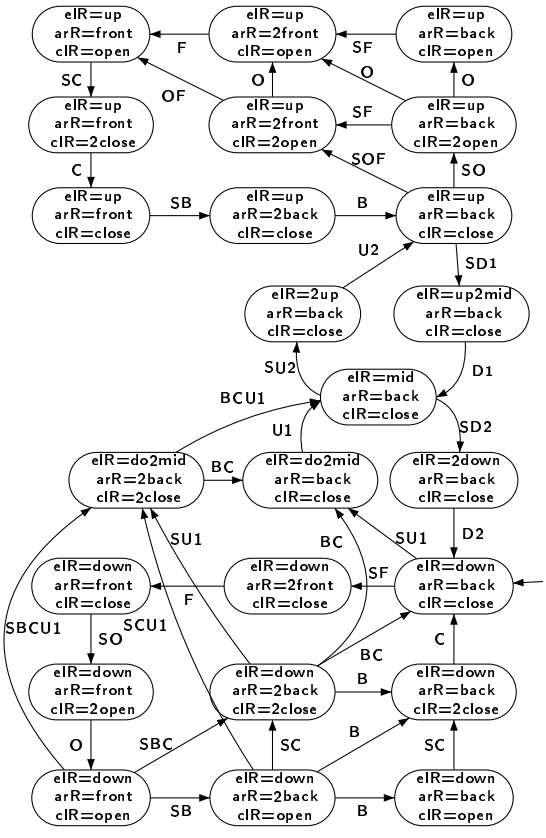
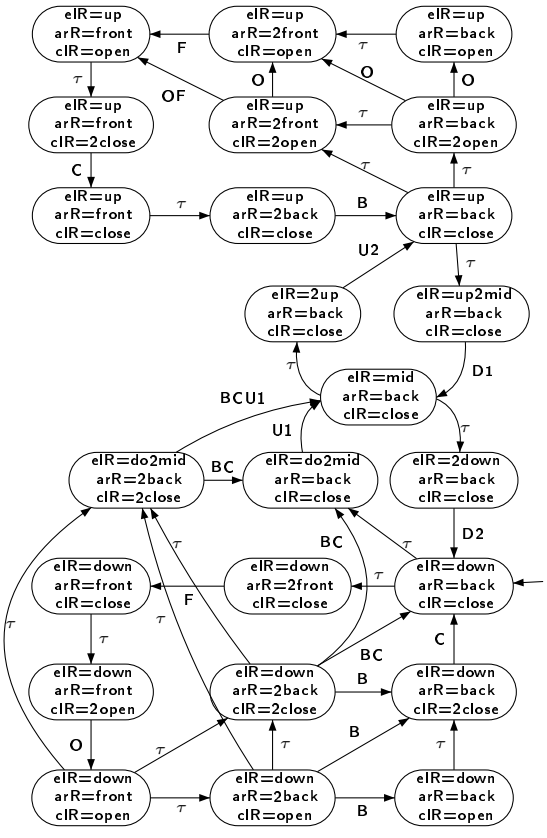


FIG. 2.10: Système abstrait $Robot_A$



(a) $Robot_R$



(b) $\mathcal{T}(Robot_R)$

FIG. 2.11: Système raffiné $Robot_R$

2.2.4 Exemple de raffinement

Dans cette section, nous nous intéressons à l'étude d'un raffinement du bras mobile robotisé présenté dans la section 1.1.4. La figure 2.10 illustre le ST2E abstrait $Robot_A$. Notons que $Robot_A$ modélise exactement le même dispositif que le ST2E *Robot* donné section 1.1.4.

Nous souhaitons détailler les comportements modélisés dans $Robot_A$ afin d'observer le point de départ de chaque transition. Considérons maintenant le ST2E $Robot_R$, illustré par la figure 2.11(a). De nouvelles étiquettes de transitions sont aussi introduites : SC, SB, SBC, SO, $SBCU1$, $SCU1$, SF, $SU1$, $SU2$, $SD1$, $SD2$ et SOF. Par exemple, l'ouverture de la pince du robot représenté dans $Robot_A$ par les transitions étiquetées O (pour "ouverture"), sera modélisée dans $Robot_R$ par les successions de deux transitions étiquetées SO (pour "commencer l'ouverture") puis O (pour "ouverture terminée"). Nous observons donc plus finement les comportements du bras mobile robotisé afin de distinguer les points de départs des différents mouvements. Les variables el, ar et cl sont raffinées par les variables elR, arR et clR, dont les domaines ont été étendus :
- $\mathbb{D}_{elR} = \{down, do2mid, mid, 2up, up, up2mid, 2down\}$,
- $\mathbb{D}_{arR} = \{back, 2front, front, 2back\}$ et
- $\mathbb{D}_{clR} = \{close, 2open, open, 2close\}$.

Nous pouvons bien entendu renommer par τ toutes les nouvelles transitions introduites précédemment, en appliquant la définition 2.3. Nous obtenons ainsi le ST2E $\mathcal{T}(Robot_R)$ illustré par la figure 2.11(b).

Le prédicat de collage donné figure 2.12 permet d'établir la relation de collage μ (voir définition 2.2) qui lie les états des systèmes $Robot_R$ et $Robot_A$. Nous pouvons ensuite vérifier que toutes les conditions de correction de la relation de raffinement η sont vérifiées entre $\mathcal{T}(Robot_R)$ et $Robot_A$ (définition 2.7). La figure 2.13 illustre la relation de raffinement η pour la bras mobile robotisé : $\mathcal{T}(Robot_R) \sqsubseteq_\eta Robot_A$. Nous pouvons conclure que $Robot_R$ est bien un raffinement de $Robot_A$.

2.3 Conclusion

Dans ce chapitre, nous avons présenté différentes notions classiques du raffinement. Nous avons commencé par rappeler la notion de raffinement *wp* de Dijkstra [Dij76], comme étant le point de départ de la notion de raffinement. Nous avons exposé deux notions de raffinement qui découlent du raffinement *wp* : le raffinement dans les systèmes d'actions [But96] et le raffinement des systèmes d'événements B [AM98] combiné avec la notion de (dé)composition. Nous avons aussi présenté la notion de raffinement dans le cadre de TLA [Lam96], de LTL [Pnu92] et de CSP. Constatons que, généralement, le raffinement est considéré comme un moyen de construire de manière progressive des systèmes et des programmes corrects.

Nous avons ensuite exposé de manière détaillée une sémantique du raffinement au niveau des systèmes de transitions doublement étiquetés. Cette définition est issue de travaux

$$
\begin{array}{lll}
 & el = up & \Leftrightarrow \quad elR \in \{up, up2mid\} \\
\wedge & el = mid & \Leftrightarrow \quad elR \in \{2up, mid, 2down\} \\
\wedge & el = down & \Leftrightarrow \quad elR \in \{down, do2mid\} \\
\wedge & ar = front & \Leftrightarrow \quad arR \in \{front, 2back\} \\
\wedge & ar = back & \Leftrightarrow \quad arR \in \{back, 2front\} \\
\wedge & cl = open & \Leftrightarrow \quad clR \in \{open, 2close\} \\
\wedge & cl = close & \Leftrightarrow \quad clR \in \{close, 2open\}
\end{array}
$$

Fig. 2.12: Prédicat de collage gp

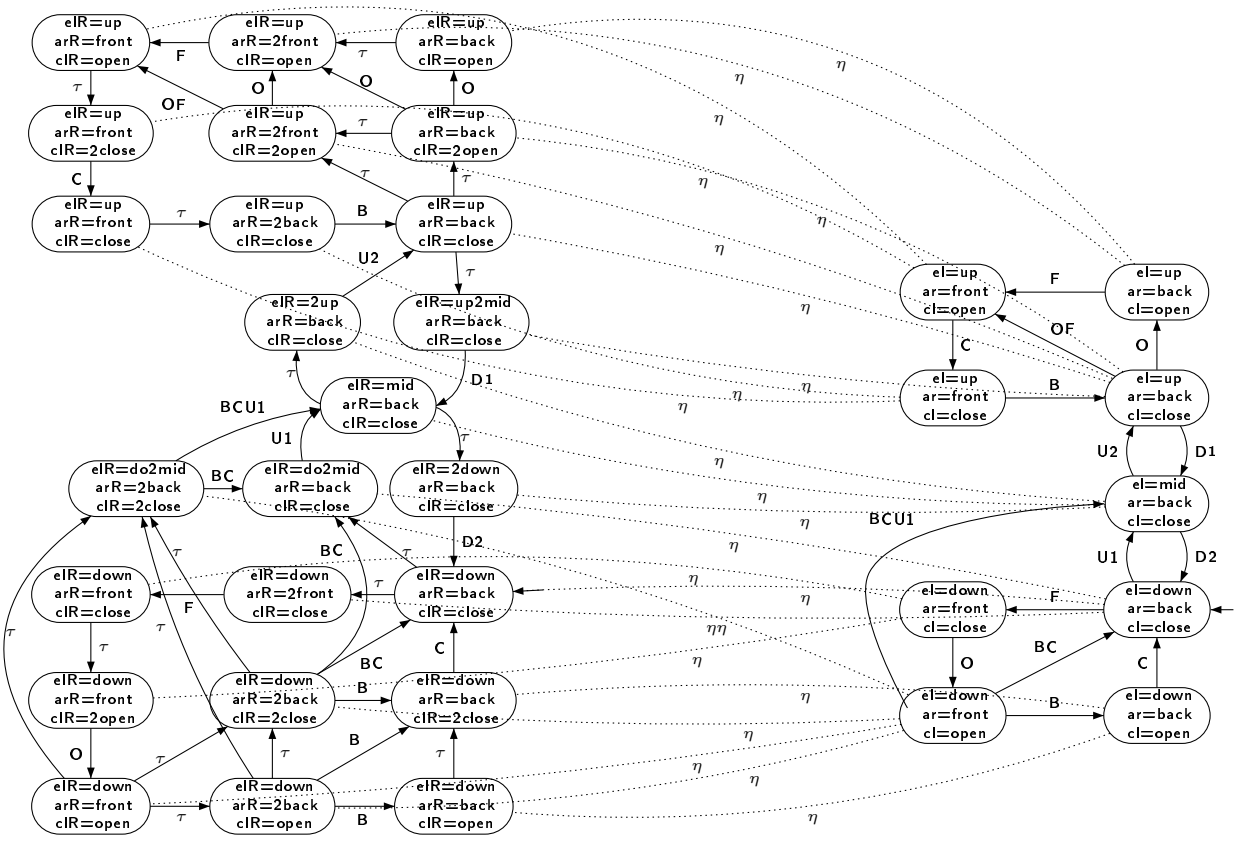


Fig. 2.13: Relation de raffinement entre $\mathcal{T}(Robot_R)$ et $Robot_A$

précédents [BJM99, BJK00] et consiste à définir une relation de raffinement en termes de τ-simulation entre un système concret SR et un système abstrait SA. Cette relation respecte le prédicat de collage qui lie les variables abstraites et concrètes, le raffinement strict des transitions d'étiquettes abstraites, le raffinement par bégaiement des transitions d'étiquettes nouvelles (τ-transitions) ainsi que la non-introduction de blocage et de τ-cycle par les nouvelles transitions.

Nous verrons dans le chapitre suivant que cette définition du raffinement permet la préservation des propriétés LTL d'un système abstrait par un système raffiné. Ainsi, ce concept est à la base de la vérification par raffinement des propriétés d'un système. Pour que le raffinement et la préservation aient un intérêt, il est nécessaire que la vérification de la relation de raffinement soit possible, c.à.d. que la taille du système raffiné n'atteigne pas les limites. C'est pourquoi les travaux des chapitres 4 et 5 s'attachent à établir des méthodes de vérification par (dé)composition du raffinement.

Chapitre 3

Propriétés dynamiques des systèmes

Le chapitre 3 concerne les propriétés des systèmes, qu'il s'agisse de leur spécification ou de leur vérification. Nous présentons aussi le lien qu'il y a entre
– vérification de propriétés et composition de systèmes, et
– vérification de propriétés et raffinement.
La section 3.1 concerne principalement la spécification des propriétés d'un système : dans un premier temps, nous nous intéressons dans la section 3.1.1 aux propriétés *statiques* spécifiées par des *invariants*. Les sections 3.1.2, 3.1.3 et 3.1.4 concernent les *logiques temporelles* et particulièrement la Logique Temporelle Linéaire (LTL). Il est connu qu'il s'agit d'un des formalismes les plus adaptés pour énoncer des propriétés *dynamiques* à propos d'un système. La section 3.2 rappelle le principe de vérification par "model-checking" d'une propriété LTL. Nous donnons ensuite dans la section 3.3 quelques résultats classiques à propos de la vérification compositionnelle des propriétés d'un système.

Dans la section 3.4, nous présentons l'un des avantages principaux du raffinement : supposons un système abstrait SA ainsi qu'un raffinement SR de SA. Certaines propriétés dynamiques vérifiées par SA seront automatiquement *préservées* au niveau de SR. C'est en particulier le cas du raffinement de systèmes de transitions finis (présenté section 2.2), qui préserve, modulo le collage, la totalité des propriétés LTL.

3.1 Spécification des propriétés d'un système

3.1.1 Propriétés d'invariance

Les propriétés d'invariance permettent d'exprimer des propriétés *statiques* qui devront être vérifiées quoi qu'il arrive. Plus précisément, tous les états du système modélisé devront satisfaire la propriété d'invariance énoncée à propos du système.

Soit $SP_V \stackrel{\text{def}}{=} \{sp, sp', \ldots\}$ l'ensemble de formules d'états formées à partir d'un ensemble de propositions atomiques AP_V. SP_V est formellement défini par la grammaire suivante :

$$sp, sp' \quad ::= \quad ap \mid sp \vee sp' \mid \neg sp \quad \text{avec } ap \in AP_V$$

La satisfaction d'une formule d'état sp par un état q d'un ST2E S, que nous noterons $q \models sp$, est définie inductivement sur la structure de sp. Classiquement, la satisfaction d'une formule d'état est définie pour une structure de Kripke. Ici, la définition est similaire. Si S est un ST2E, alors $SK(S)$ est la SK associée à S.

Définition 3.1 (Satisfaction d'une formule d'état sp)
Nous dirons que $sp \in SP_V$ est satisfaite par un état q, noté $q \models sp$, si
– *$q \models ap$ si $ap \in l(q)$,*
– *$q \models \neg sp$ s'il n'est pas vrai que $q \models sp$, et*
– *$q \models sp \vee sp'$ si $q \models sp$ ou si $q \models sp'$*

Nous appellerons invariant *une formule d'état $sp \in SP_V$ satisfaite par un ST2E S, i.e., $\forall q.\ (q \in Q \Rightarrow q \models sp)$, noté $S \models sp$.*

De manière classique, nous utiliserons aussi les règles de réécriture suivantes pour donner une sémantique aux opérateurs que nous n'avons pas formellement définis : $sp \wedge sp' \equiv \neg(\neg sp \vee \neg sp')$ et $sp \Rightarrow sp' \equiv \neg sp \vee sp'$ et $sp \Leftrightarrow sp' \equiv (sp \Rightarrow sp') \wedge (sp' \Rightarrow sp)$.

> **Exemple.** Donnons maintenant un exemple de propriétés d'invariance à propos de l'exemple du bras mobile robotisé présenté section 1.1.4 et section 2.2.4. Un invariant de *Robot* pourrait être que quand le bras est en train de monter ou de descendre (c.à.d. en position médiane), alors la pince est toujours fermée et le bras toujours reculé :
>
> $$(el = mid) \Rightarrow (ar = back \wedge cl = close) \tag{3.1}$$

Nous aurons besoin dans la suite de ce document de parler de la satisfaction par un couple d'états (q_1, q_2) d'une formule d'état sp définie sur un ensemble de variables $V_1 \cup V_2$. Pour ce faire, nous étendons la définition 3.1 précédente aux couples d'états.

Définition 3.2 (Satisfaction d'une formule d'état sp par (q_1, q_2))
Nous dirons que $sp \in SP_{V_1 \cup V_2}$ est satisfaite par un couple d'états $(q_1, q_2) \in Q_1 \times Q_2$, noté $(q_1, q_2) \models sp$, si
– $(q_1, q_2) \models ap$ *si* $ap \in l_1(q_1) \cup l_2(q_2)$
– $(q_1, q_2) \models \neg sp$ *s'il n'est pas vrai que* $(q_1, q_2) \models sp$ *et*
– $(q_1, q_2) \models sp \vee sp'$ *si* $(q_1, q_2) \models sp$ *ou si* $(q_1, q_2) \models sp'$

3.1.2 Logique temporelle linéaire (LTL)

Pour énoncer des propriétés à propos du comportement dynamique d'un système, les propriétés d'invariance ne sont pas suffisantes. Pour ce faire, nous introduisons ici la *Logique Temporelle Linéaire* (LTL : Linear Temporal Logic). La LTL, définie par Pnueli [Pnu77, Pnu81, MP92], est une logique adaptée à la description de propriétés à propos de l'ordonnancement des états dans le temps. Mais, la notion de "temps qui passe" est ici implicite, et est exprimée en terme d'exécution du système, c.à.d. de succession d'états.

Une formule LTL s'exprime par la grammaire suivante :

$$\begin{array}{rcll} \phi, \phi' & ::= & ap & \text{avec } ap \in AP_V \\ & | & \phi \vee \phi' \ | \ \neg\phi & \\ & | & \bigcirc\phi \ | \ \phi \,\mathcal{U}\, \phi' & \end{array}$$

D'autres opérateurs LTL existent : $\Diamond$, $\Box$ et $\mathcal{W}$. Néanmoins, ils peuvent être ré-exprimés grâce aux opérateurs LTL $\bigcirc$ et $\mathcal{U}$: $\Diamond\phi \equiv true \,\mathcal{U}\, \phi$, $\Box\phi \equiv \neg(\Diamond\neg\phi)$ et $\phi \,\mathcal{W}\, \phi' \equiv (\Box\phi) \vee (\phi \,\mathcal{U}\, \phi')$. Nous désignons par LTL l'ensemble des formules LTL. $LTL_{-\bigcirc}$ désigne le sous-ensemble des formules LTL sans l'opérateur $\bigcirc$.

Soient $\phi \in LTL$ et π une exécution. La signification des opérateurs LTL précédents est la suivante.
– $\bigcirc\phi$ indique que le successeur direct de l'état courant le long de π doit vérifier ϕ,

- $\Diamond\phi$ indique qu'il existe, à partir de l'état courant, un état le long de π qui vérifie ϕ,
- $\Box\phi$ indique que tous les états de π, à partir de l'état courant, doivent vérifier ϕ,
- $\phi\,\mathcal{U}\,\phi'$ indique que, à partir de l'état courant, tous les états de π doivent vérifier ϕ jusqu'à ce qu'on atteigne un état vérifiant ϕ', l'atteignabilité étant obligatoire, et,
- $\phi\,\mathcal{W}\,\phi'$ indique la même chose que $\phi\,\mathcal{U}\,\phi'$ sans l'obligation d'atteignabilité.

Classiquement, la satisfaction d'une formule LTL est donnée pour une structure de Kripke. Nous pouvons considérer un ST2E S, puisque $SK(S)$ est la SK associée à S par projection. Formellement, la satisfaction d'une formule $\phi \in LTL$ est exprimée par induction sur la structure de ϕ (voir aussi figure 3.1).

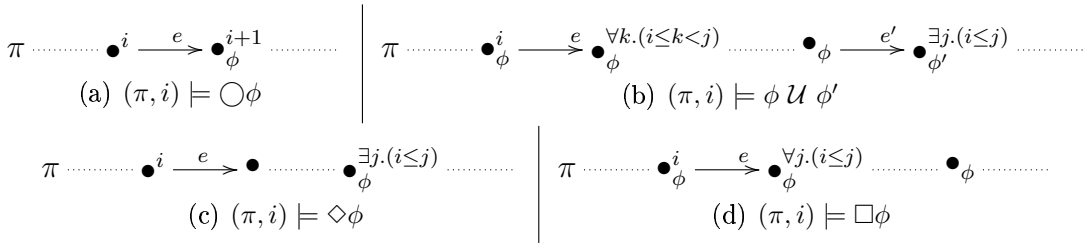


Fig. 3.1: Sémantique des différents opérateurs LTL

Définition 3.3 (Satisfaction d'une formule LTL ϕ)
Soient S un ST2E et π une exécution de S. Nous dirons que $\phi \in LTL$ est satisfaite à l'étape i de l'exécution π ($i \geq 0$), noté $(\pi, i) \models \phi$, si
- $(\pi, i) \models ap$ *si* $ap \in l(\pi(i))$,
- $(\pi, i) \models \neg\phi$ *s'il n'est pas vrai que* $(\pi, i) \models \phi$,
- $(\pi, i) \models \phi \vee \phi'$ *si* $(\pi, i) \models \phi$ *ou* $(\pi, i) \models \phi'$,
- $(\pi, i) \models \bigcirc\phi$ *si* $(\pi, i+1) \models \phi$
- $(\pi, i) \models \phi\,\mathcal{U}\,\phi'$ *s'il existe j tel que $i \leq j \leq |\pi|$ et $(\pi, j) \models \phi'$, et pour tout k tel que $i \leq k < j$, on a $(\pi, k) \models \phi$.*

Nous dirons qu'une formule $\phi \in LTL$ est satisfaite par S, noté $S \models \phi$, si

$$\forall \pi.\ (\pi \in \Pi(S) \Rightarrow (\pi, 0) \models \phi)$$

Etant donné les règles d'équivalence, la définition formelle de la sémantique des opérateurs $\bigcirc$ et $\mathcal{U}$ suffit. Nous indiquons la sémantique des opérateurs LTL $\Diamond$ et $\Box$ pour simplifier la lecture de preuves ultérieures :
- $(\pi, i) \models \Diamond\phi$ *s'il existe j tel que $i \leq j \leq |\pi|$ et $(\pi, j) \models \phi$,*
- $(\pi, i) \models \Box\phi$ *si pour tout j tel que $i \leq j \leq |\pi|$, on a $(\pi, j) \models \phi$.*

Exemple. Considérons l'exemple du bras mobile robotisé étudié dans les sections 1.1.4 et 2.2.4. Nous rappelons qu'un modèle de ce système est donné par le ST2E *Robot* (voir figures 1.5 et 1.6). Nous pouvons exprimer certaines propriétés du bras mobile robotisé à l'aide de formules LTL :

– La propriété d'invariance (3.1) s'exprime ainsi en LTL :

$$\Box((el = mid) \Rightarrow (ar = back \wedge cl = close)) \tag{3.2}$$

– La pince ne reste pas indéfiniment ouverte :

$$\begin{array}{c} \neg\Diamond\Box(cl = open) \quad \text{ou} \\ \Box((cl = open) \Rightarrow \Diamond(cl = close)) \end{array} \tag{3.3}$$

– Si le dispositif est avancé, en position basse et que la pince est ouverte, alors le dispositif recule :

$$\Box((el = down \wedge ar = front \wedge cl = open) \Rightarrow \bigcirc(ar = back)) \tag{3.4}$$

– En position haute, si le dispositif est avancé et que la pince est ouverte, alors la pince se referme avant tout autre mouvement :

$$\Box((el = up \wedge ar = front \wedge cl = open) \Rightarrow \bigcirc(cl = close)) \tag{3.5}$$

– Quand le dispositif est en train de monter ou de descendre (c.à.d. en position médiane), les seules actions autorisées sont la montée ou la descente du dispositif :

$$\Box((el = mid) \Rightarrow \bigcirc(el \in \{up, down\})) \tag{3.6}$$

– La pince ne peut pas être bras avancé et en train de monter ou de descendre :

$$\begin{array}{c} \neg\Diamond(el = mid \wedge ar = front) \quad \text{ou} \\ \Box(ar = back \vee el \in \{up, down\}) \end{array} \tag{3.7}$$

3.1.3 Autres logiques temporelles

Il existe un large éventail de logiques temporelles étudiées dans la littérature. Nous n'essayerons pas ici de dresser un état de l'art exhaustif quant à l'expression des propriétés dynamiques d'un système. Une classification peut, par exemple, être trouvée dans [CGP00]. Nous nous contenterons uniquement de présenter quelques autres formalismes, proches de LTL, parce qu'il s'agit encore de logiques basées sur les *états* du système étudié. Citons entre autre CTL* et CTL, les modalités B ou encore Past+LTL, c.à.d. LTL avec des opérateurs du passé. Nous aurions encore pu faire le choix de présenter d'autres logiques temporelles basées quant-à elles sur les *actions* du système considéré, citons entre autres le μ-calcul modal [Koz83], le μ-calcul linéaire [Var88] ou encore TLA [Lam94].

CTL* et CTL

Les logiques temporelles CTL* (pour *Computation Tree Logic*) [EH86] et CTL [CE81, EH82, CES86], permettent d'exprimer des propriétés portant sur les arbres d'exécutions

du modèle considéré, contrairement à la logique temporelle LTL : une formule LTL considérant une exécution donnée ne pourra pas examiner les exécutions alternatives qui s'en distinguent. Par exemple, la LTL ne permettra pas d'exprimer que dans un certain état le long d'une certaine exécution, il sera possible de prolonger cette exécution de telle ou telle manière, alors qu'une formule CTL* le permettrait.

La syntaxe de CTL* est donnée par la grammaire suivante :

$$\begin{array}{rcll} \phi, \phi' & ::= & ap & \text{avec } ap \in AP_V \\ & | & \phi \vee \phi' \ | \ \neg\phi & \\ & | & \mathrm{X}\phi \ | \ \phi \mathrm{U} \phi' & \\ & | & \mathrm{A}\phi & \end{array}$$

où les opérateurs X et U ont respectivement la même signification que les opérateurs LTL $\bigcirc$ et $\mathcal{U}$ (on notera G pour $\Box$ et F pour $\Diamond$), et où l'opérateur A indique que toutes les exécutions partant de l'état courant doivent satisfaire ϕ.

On définit grâce à l'égalité suivante $\neg \mathrm{A}\phi \equiv \mathrm{E}\neg\phi$ un autre opérateur $\mathrm{E}\phi$ qui indique qu'à partir de l'état courant, il existe une exécution qui satisfait ϕ. Les opérateurs A et E sont appelés *quantificateurs de chemins*.
CTL^* désigne l'ensemble des formules de CTL*.

La logique temporelle CTL est le fragment de CTL* où l'on exige que chaque opérateur temporel (X, G, F, U ou W) soit directement sous la portée d'un des quantificateurs A ou E, c.à.d. que l'on peut considérer que les opérateurs CTL sont AX, EX, A_U_, E_U_, etc. Par définition, $CTL \subset CTL^*$.

Dans certains cas, on souhaitera uniquement parler des "chemins universellement quantifiés". La restriction de CTL* au quantificateur A est appelée ACTL* ($ACTL^* \subset CTL^*$). De la même manière, on définira $ECTL*$. Puisque les opérateurs LTL correspondent à une partie des opérateurs CTL*, on peut dire que LTL est le fragment de CTL^* lorsque l'on retire les opérateurs A et E : en fait, LTL correspond à une partie de CTL^* dans laquelle toutes les formules sont quantifiées universellement par un A implicite. Il a été montré que l'expressivité de la LTL était incluse dans celle de ACTL* : $LTL \subset ACTL^* \subset CTL^*$ [CGP00]. La figure 3.2 représente un diagramme d'inclusion pour les différentes logiques temporelles (ou sous-ensemble) présentées (voir aussi section 3.1.4).

Past+LTL : LTL avec opérateurs du passé

Les opérateurs LTL classiques $\bigcirc$, $\mathcal{U}$, $\Box$ ou $\Diamond$ font référence au "futur" dans la formule LTL dans laquelle ils interviennent. Dans certains cas, on trouvera plus simple d'exprimer certaines propriétés à propos de ce qui s'est passé avant. On définit alors la Past+LTL (Past Linear Temporal Logic) comme une extension de LTL avec de nouveaux opérateurs parlant du "passé" :
– $\bigcirc^{-1}\phi$ indique que l'état précédent directement l'état courant vérifiait ϕ,
– $\Diamond^{-1}\phi$ indique que ϕ a été vérifiée dans un état précédent l'état courant,

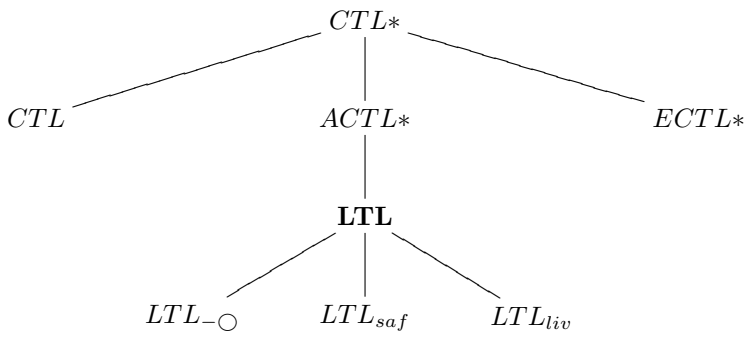


Fig. 3.2: Diagramme d'inclusion des différentes logiques temporelles

- $\Box^{-1}\phi$ indique que tous les états précédents l'état courant vérifiaient ϕ,
- $\phi \mathcal{S} \phi'$ indique d'une part que ϕ' a été vérifiée dans un état précédent, et que, d'autre part, depuis cet état, ϕ est vérifiée.

Ces opérateurs peuvent être vus comme une version miroir des opérateurs LTL correspondants. On a les règles de réécriture suivantes : $\Diamond^{-1}\phi \equiv true\ \mathcal{S}\ \phi$ et $\Box^{-1}\phi \equiv \neg(\Diamond^{-1}\neg\phi)$. LTL^{-1} désigne l'ensemble de toutes formules de Past+LTL et LTL^{-1}_{only}, l'ensemble des formules formées uniquement d'opérateurs du passé.

Nous indiquons maintenant la sémantique des opérateurs du passé $\bigcirc^{-1}$ et $\mathcal{S}$. Cette sémantique vient étendre (pour $i > 0$) la sémantique donnée définition 3.3 aux opérateurs LTL classiques :

- $(\pi, i) \models \bigcirc^{-1}\phi$ *si* $(\pi, i-1) \models \phi$ *et* $i > 0$,
- $(\pi, i) \models \phi\ \mathcal{S}\ \phi'$ *s'il existe* j *tel que* $0 \leq j \leq i$ *et* $(\pi, j) \models \phi'$, *et pour tout* k *tel que* $j < k \leq i$, *on a* $(\pi, k) \models \phi$.

Kamp [Kam68], puis Gabbay [GPSJ80, Gab89] énoncent que toute formule de Past+LTL peut être ré-exprimée par une formule *équivalente* exprimée uniquement en LTL (voir aussi [GB92]). En d'autres termes, la Past+LTL n'est pas plus expressive que la LTL. Il a par contre été montré dans [LMS02] que toute formule Past+LTL était exponentiellement plus petite que toute formule purement LTL équivalente.

Exemple. Considérons l'exemple du bras mobile robotisé étudié dans les sections 1.1.4 et 2.2.4. Grâce à la Past+LTL, nous pouvons exprimer certaines propriétés dynamiques :

- Si le bras est en position haute, cela signifie qu'il est passé par la position médiane :

$$\Box((el = up) \Rightarrow \bigcirc^{-1}(el = mid)) \tag{3.8}$$

- Pour que le bras mobile soit pince ouverte, en position basse et avancée, il est nécessaire qu'il ait été pince fermée, en position basse et reculée :

$$\begin{array}{l}\Box(((el = down \wedge ar = front \wedge cl = open) \\ \Rightarrow \Diamond^{-1}((el = down \wedge ar = back \wedge cl = close))\end{array} \tag{3.9}$$

Modalités B

Dans [AM98], Abrial et Mussat introduisent trois schémas de propriétés dynamiques, afin d'exprimer des contraintes temporelles liées aux systèmes d'événements B. Il s'agit des *invariants dynamiques* et des modalités *leads to* et *until*.

Soit $SE = (E, V, Inv, Init, EV)$ un système d'événements B.

- Un invariant dynamique se note

  Dynamics $P(V, V')$

  où $P(V, V')$ est un prédicat *avant-après* ; V' désigne l'ensemble des variables après l'application d'un événement.
- Une modalité *leads to* se note

  Select sp **Leadsto** sp' [**Invariant** inv **Variant** var] **End**

  où sp et sp' sont deux formules propositionnelles définies sur V et où l'invariant local inv et le variant var servent à prouver la maintenance de la modalité par le système SE.
- Une modalité *until* se note

  Select sp **Until** sp' [**Invariant** inv **Variant**] var **End**

Dans [BDJK01], les auteurs indiquent que les contraintes temporelles B d'Abrial et Mussat peuvent toutes être ré-exprimées en LTL. Considérons sp et sp', deux formules propositionnelles. Un invariant dynamique pourra s'exprimer en LTL par une formule de la forme $\Box(sp \Rightarrow \bigcirc sp')$. En ce qui concerne les modalités, elles s'exprimeront par les formules LTL $\Box(sp \Rightarrow \Diamond sp')$, pour la modalité *leads to*, et $\Box(sp \Rightarrow sp\ \mathcal{U}\ sp')$, pour la modalité *until*.

> **Exemple.** On pourrait ajouter au système d'événements B correspondant au bras mobile robotisé donné figure 1.17, l'invariant dynamique suivant
>
> **Dynamics** $el = mid \wedge el' \in \{up, down\}$ (3.10)
>
> correspondant à la propriété LTL (3.6) ou bien encore la modalité suivante
>
> **Select** $cl = open$ **Leadsto** $cl = close$ **end** (3.11)
>
> correspondant quant-à-elle à la propriété LTL (3.3).

3.1.4 Classification des propriétés temporelles

Les propriétés temporelles, qu'elles soient exprimées en LTL, en CTL* ou dans d'autres formalismes sont souvent classifiées en fonction des objectifs de vérification qu'une certaine propriété doit énoncer : atteignabilité, sûreté, vivacité, etc. Nous reprenons ici quelques définitions informelles données dans [S+99] :

- Les *propriétés de sûreté* énoncent que, sous certaines conditions, quelque chose ne se produit jamais. Les propriétés d'invariance ou d'inatteignabilité font partie des propriétés de sûreté. En LTL, ces propriétés ont souvent la forme $\Box\phi$.
- Les *propriétés de vivacité* énoncent que, sous certaines conditions, quelque chose finira par avoir lieu. Les opérateurs LTL utilisés pour spécifier ce type de propriétés sont les opérateurs $\Diamond$ ou $\mathcal{U}$.
- Les *propriétés d'atteignabilité* énoncent qu'une certaine situation peut être atteinte. La LTL ne permet pas d'exprimer l'atteignabilité. On peut par contre exprimer en LTL que quelque chose n'est jamais atteignable et on parle alors d'inatteignablité. On se situe dans la classe des propriétés de sûreté.
- Les *propriétés d'absence de blocages* énoncent que le système ne se trouve pas dans une situation où il lui est impossible de progresser. Dans certains cas, l'absence de blocages est assimilée à une propriété de sûreté.
- Les *propriétés d'équité* énoncent que, sous certaines conditions, quelque chose aura lieu un nombre infini de fois. Une propriété d'équité est quelquefois vue comme une propriété de vivacité *répétée*, par opposition aux propriétés de vivacité simple, que l'on qualifie aussi de *progrès*.

Pour ce qui concerne la LTL, Lamport a proposé dans [Lam73] une distinction entre propriétés de sûreté et propriétés de vivacité. Par la suite, des caractérisations purement syntaxiques [Sis85, Sis94] – où bien en termes d'automates [AS87, AS89] – ont été énoncées. Dans [LPZ85, LS95], la caractérisation des propriétés de sûreté utilise la Past+LTL. Nous noterons $LTL_{saf} \subset LTL$, l'ensemble des formules LTL exprimant des propriétés de sûreté et $LTL_{liv} \subset LTL$, l'ensemble exprimant des propriétés de vivacité. Toute propriété de sûreté LTL est exprimable sous la forme $\Box\phi_p$ où ϕ_p est une formule qui utilise uniquement des opérateurs du passé.

Lemme 3.1 (Propriété de sûreté [LPZ85])
Soit $\phi \in LTL$ une formule LTL quelconque. S'il existe $\phi_p \in LTL^{-1}_{only}$ telle que $\phi \equiv \Box\phi_p$ alors $\phi \in LTL_{saf}$.

> **Exemple.** Les propriétés LTL (3.2), (3.5) et (3.7) exprimées section 3.1.2 sont des propriétés de sûreté. Les propriétés (3.2) et (3.7) sont déjà sous la forme $\Box\phi_p$. La propriété (3.5) est équivalente à
>
> $$\Box(suivant^{-1}(el = up \wedge ar = front \wedge cl = open) \Rightarrow (cl = close))$$

Dans [AS85], il est démontré qu'une formule LTL quelconque pouvait toujours se décomposer en une formule LTL de sûreté et une formule LTL de vivacité.

Lemme 3.2 (Décomposition d'une formule LTL [AS85])
Soit $\phi \in LTL$. Alors, il existe $\phi_s \in LTL_{saf}$ et $\phi_l \in LTL_{liv}$ telles que $\phi \equiv \phi_s \wedge \phi_l$.

Chang, Manna et Pnueli ont proposé dans [MP90, CMP92] une caractérisation syntaxique beaucoup plus précise des formules LTL appelée *hiérarchie de Borel*. La figure 3.3 illustre

cette classification, sous la forme d'un diagramme d'inclusion. Elle identifie six classes de propriétés en fonction de leur forme purement syntaxique.

Dans [DAC98, DAC99], Dwyer et al. proposent une classification plus méthodologique des propriétés temporelles en huit catégories, basée sur des schémas de propriétés : *Précédence*, *Absence*, *Universalité*, *Existence*, *Existence bornée*, *Réponse*, *Précédences en cascade* et *Réponse en cascade*. Ils ont aussi réalisé une étude basée sur plus de 500 spécifications afin de déterminer la fréquence d'apparition des différents schémas identifiés.

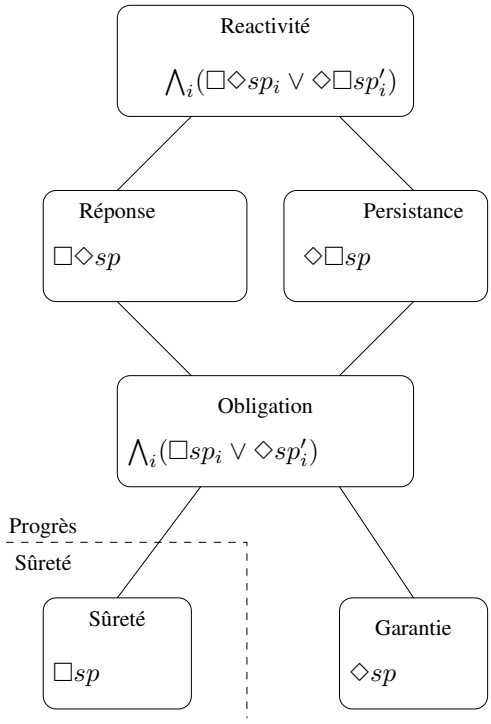


Fig. 3.3: Hiérarchie de Borel [CMP92]

3.2 Vérification des propriétés LTL d'un système

La vérification des propriétés LTL repose sur une technique de *model-checking* (littéralement, vérification de modèles) permettant de vérifier la cohérence entre deux modèles : celui du système exprimé sous la forme d'une SK ou d'un ST2E et celui de la propriété à vérifier, exprimé sous la forme d'un *automate de Büchi* [LP85, VW86, CGP00].

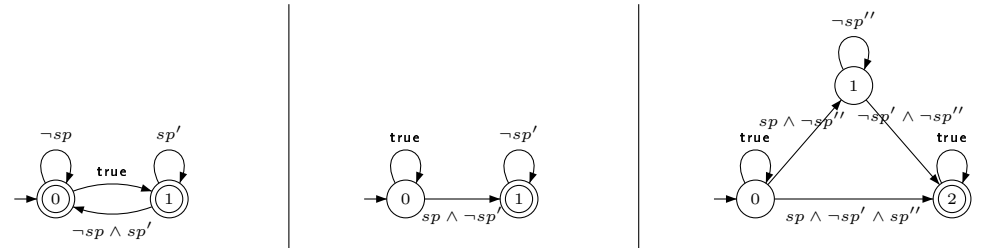
Considérons un ST2E S. Un automate de Büchi AB est un automate d'une classe particulière qui donne une représentation de la formule LTL. Les automates de Büchi reconnaissent des langages ω-réguliers. L'alphabet utilisé est l'ensemble des propositions d'états SP_V. Une exécution $\pi \in \Pi(S)$ peut être vue comme un mot du langage reconnu par AB.

Définition 3.4 (Automate de Büchi)
Un automate de Büchi est un quintuplet $AB = (Q, Q_O, E, T, F)$ tel que :
– *Q est un ensemble fini d'états,*
– *$Q_O \subseteq Q$ est un ensemble des états initiaux,*
– *E est un ensemble d'étiquettes de transitions,*
– *$T \subseteq Q \times E \times Q$ est une relation de transition étiquetée, et*
– *$F \subseteq Q$ est un ensemble d'états d'acceptation de l'automate.*

Si l'on considère une formule $\phi \in LTL$ exprimée sur l'ensemble de variables V de S, alors il est toujours possible de construire un automate de Büchi AB tel que AB reconnaisse l'ensemble des exécutions $\pi \in \Pi(S)$ qui satisfont la propriété ϕ. C'est ce qu'exprime le théorème suivant :

Théorème 3.1 (Propriété LTL et automate de Büchi [Var94, VW94])
Etant donnée une formule $\phi \in LTL$, on peut construire un automate AB tel que le langage reconnu par AB soit exactement l'ensemble des exécutions satisfaisant la formule ϕ.

Il existe de nombreux algorithmes permettant de construire automatiquement l'automate de Büchi correspondant à une formule LTL donnée [WVS83, VW94, GPVW95, EH00, SB00, GO01, GL02b]. La figure 3.4 donne à titre d'exemple quelques automates de Büchi reconnaissant les formules LTL suivantes : $\Box(sp \Rightarrow \bigcirc sp')$, $\neg\Box(sp \Rightarrow \Diamond sp')$ et $\neg\Box(sp \Rightarrow sp' \mathcal{W} sp'')$.



(a) $\Box(sp \Rightarrow \bigcirc sp')$ (b) $\neg\Box(sp \Rightarrow \Diamond sp')$ (c) $\neg\Box(sp \Rightarrow sp' \mathcal{W} sp'')$

Fig. 3.4: Automates de Büchi reconnaissant certaines formules LTL

Pour répondre à la question "est-ce que S satisfait ϕ ?", l'algorithme classique du model-checking LTL suit la démarche suivante [LP85, VW86, CGP00].

1. Dans un premier temps, on va réaliser le produit synchronisé du ST2E S modélisant le système avec l'automate de Büchi AB modélisant la négation de la propriété ϕ à vérifier.
2. On va ensuite chercher à détecter dans l'automate produit $S \times AB$ les composantes fortement connexes : la détection de composantes fortement connexes peut, par exemple, être réalisée grâce à l'algorithme de Tarjan [Tar83].
   – Si l'une des composantes fortement connexes contient un état d'acceptation cela signifie qu'il existe une exécution $\pi \in \Pi(S)$ acceptée par l'automate AB, c.à.d. qui satisfait $\neg\phi$.

– Si ce n'est pas le cas, cela signifie que S ne satisfait pas la négation de ϕ, et donc que S satisfait ϕ.

Des algorithmes pour la vérification par model-checking de propriétés CTL ou CTL* pourraient aussi être présentés : voir [QS82, CES86, CGP00], pour ce qui concerne la vérification de propriétés CTL et [EL85, CES86, CGP00] pour ce qui concerne les propriétés CTL*.

Le model-checker SPIN

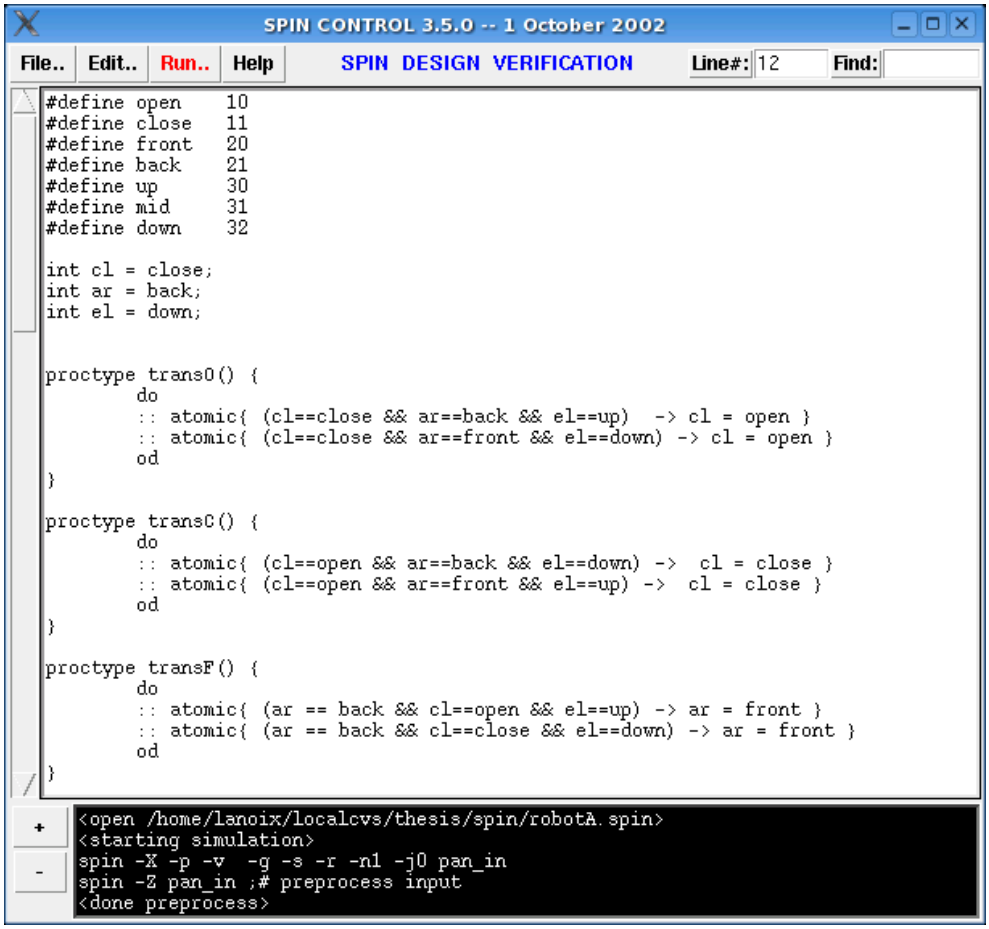


Fig. 3.5: Spécification Promela de $Robot_A$ sous SPIN

SPIN [Hol97, Hol03, SPI] est l'un des model-checkers LTL les plus utilisés. Il propose de vérifier la satisfaction de propriétés LTL pour des systèmes communicants à nombre d'états fini.

SPIN utilise un langage de spécification appelé Promela[1], basé sur le langage des commandes gardées de Dijkstra. En Promela, on peut spécifier différents types de communica-

[1]Pour PROcess MEta LAngage.

tions entre processus : variables partagées et canaux de communication, entre autres.

Les propriétés à vérifier peuvent être exprimées de plusieurs façons : soit sous forme d'assertions locales, soit sous forme de formules LTL, soit directement sous la forme d'un automate de Büchi.

Dans tous les cas, SPIN transforme la propriété à vérifier en automate de Büchi et applique ensuite la démarche de vérification classique présentée précédemment. La vérification est effectuée *à la volée*, c.à.d. que SPIN ne construit pas l'automate complet représentant le système pour vérifier la propriété. De plus, SPIN utilise plusieurs techniques d'implémentation pour combattre le problème de l'explosion combinatoire : réduction d'ordres partiels, BDD, compression mémoire, tables de hashage, etc.

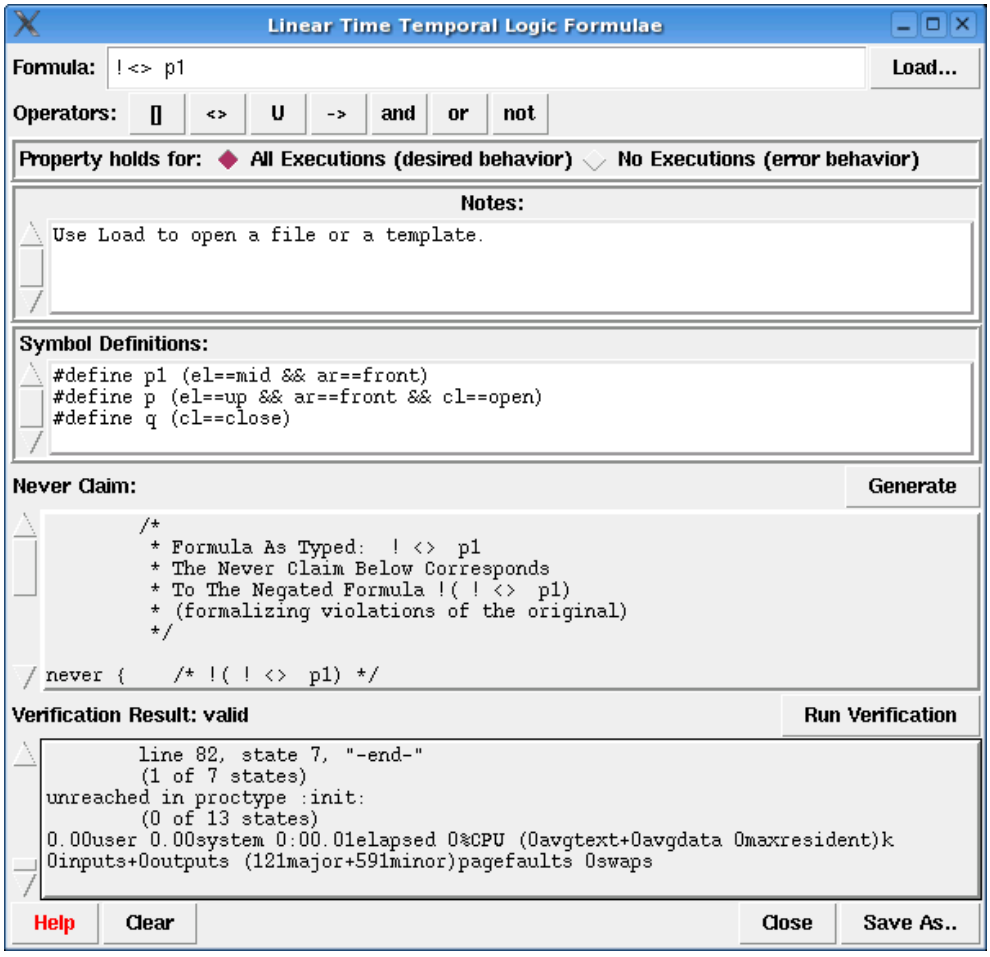


Fig. 3.6: Vérification de la propriété LTL (3.7) avec SPIN

Exemple. Grâce à SPIN, nous pouvons vérifier que les propriétés LTL (3.2) à (3.7) énoncées dans la section 3.1.2 sont satisfaites par $Robot_A$. Nous commençons par donner figure 3.5 une spécification en Promela qui correspond à

$Robot_A$. Ensuite, nous vérifions grâce à SPIN chacune des propriétés : la figure 3.6 montre la vérification de la propriété (3.7).

3.3 Propriétés et compositionnalité

Le principal problème de la vérification compositionnelle de propriétés consiste à réduire la vérification d'une propriété globale sur le système pris dans sa globalité à la vérification de propriétés locales sur différents "morceaux" du système. Exprimé autrement, il s'agit de déduire de la satisfaction de certaines propriétés $\phi_1, \phi_2, \ldots, \phi_n$ par les composants $S^1, S^2, \ldots S^n$ d'un système, la satisfaction d'une propriété ϕ par le système complet $S^1 \| S^2 \| \ldots \| S^n$.

$$\frac{S^1 \models \phi_1, S^2 \models \phi_2, \ldots S^n \models \phi_n}{S^1 \| S^2 \| \ldots \| S^n \models \phi}$$

Dans le cas des propriétés d'invariance, il est bien connu (voir par exemple [BLS96, Cha03b]) que les invariants satisfaits par les composants d'un système, ne peuvent être composés que si l'opérateur de composition $\|$ considéré n'autorise pas les variables partagées.

- Soient $I_1 \in SP_{V^1}$ et $I_2 \in SP_{V^2}$ tels que $S^1 \models I_1$ et $S^2 \models I_2$. Alors, on peut conclure que $S^1 \| S^2$ satisfait le prédicat $I_1 \wedge I_2$.
- Dans le cas d'une composition à variables partagées, les invariants I_1 et I_2 sont définis sur un ensemble de formules d'états SP_V. Dans le cas où $S^1 \models I_1$ et $S^2 \models I_2$, ils ne peuvent pas être composés. On peut uniquement conclure à propos de leur disjonction : $S^1 \| S^2 \models I_1 \vee I_2$.

Le principe d'Hypothèse/Garantie. Afin de *garantir* qu'un composant S^i satisfera une certaine propriété temporelle ϕ_i, il est souvent nécessaire que son "environnement" (i.e. les autres composants du système) satisfasse certaines *hypothèses*. Cette démarche appelée *Hypothèse/Garantie*[2] a été introduite, entre autres, par Misra et Chandy [MC81], Lamport [Lam83] et Jones [Jon83].

Nous noterons $< \phi > S < \psi >$ pour indiquer que, si l'environnement de S satisfait ϕ, alors, S, dans cet environnement, satisfait ψ. D'une manière générale, le principe d'Hypothèse/Garantie s'exprime ainsi :

$$\frac{\begin{array}{l} < \quad > S^1 < \phi >, \\ < \phi > S^2 < \psi > \end{array}}{< \quad > S^1 \| S^2 < \psi >}$$

Ce qui signifie que si S^1 satisfait ϕ et que S^2, pris dans un environnement satisfaisant ϕ, satisfait ψ, alors $S^1 \| S^2$ satisfait ψ.

[2]En anglais, on parle indifféremment d'*Assumption/Guarantee*, de *Assume/Guarantee*, de *Rely/Guarantee* ou encore d'*Assumption/Commitment*

Dans [Pnu85], Pnueli a, le premier, combiné Hypothèse/Garantie et LTL. Dans le cas où ϕ et ψ sont deux formules LTL, la notation $< \phi > S < \psi >$ devient $S \models \phi \Rightarrow \psi$, où $\models$ est l'opérateur de satisfaction classique défini section 3.1.2. De la même manière, le principe d'Hypothèse/Garantie s'exprime alors

$$\frac{\begin{array}{l} S^1 \models \phi, \\ S^2 \models \phi \Rightarrow \psi \end{array}}{S^1 \| S^2 \models \psi}$$

Remarque. Pour plus de détails, voir aussi [MP95, JT96].

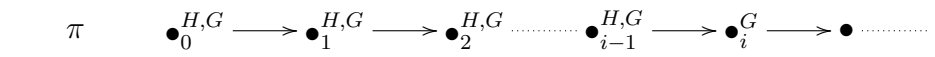


FIG. 3.7: La formule $H \rhd G$

Dans [Tsa00, TT03], le principe d'Hypothèse/Garantie est énoncé de manière plus précise, en utilisant la Past+LTL. Soient $H = \Box\phi$ et $G = \Box\psi$ deux formules, telles que $\phi \in LTL^{-1}_{only}$ et $\psi \in LTL^{-1}_{only}$. Considérons maintenant la formule $H \rhd G = \Box(\bigcirc^{-1}\Box^{-1}\phi \Rightarrow \psi)$. Cette formule indique que si, le long d'une exécution π, G est satisfaite à l'état $\pi(0)$, et que H est satisfaite pour $i-1$ états successifs, alors G est satisfaite pour i états successifs (voir figure 3.7). L'une des Hypothèses/Garanties proposées dans [Tsa00] est la suivante :

$$\frac{\begin{array}{l} S^1 \models H_1 \rhd G_1, \\ S^2 \models H_2 \rhd G_2, \\ (H_1 \rhd G_1) \wedge (H_2 \rhd G_2) \Rightarrow G \end{array}}{S^1 \| S^2 \models G}$$

Dans [NT00], d'autres Hypothèses/Garanties utilisant elles aussi des formules de la forme $H \rhd G$ sont proposées.

Kupferman et Vardi proposent dans [KV96, KV97] des adaptations des algorithmes classiques de model-checking au cas des systèmes à variables partagées : on parle alors de *module-checking*. Dans [KV98, KV00], ils étudient le principe d'Hypothèse/Garantie dans les cas où l'hypothèse est exprimée en LTL ou en CTL et la garantie en ACTL ou en ACTL*, et énoncent plusieurs règles de vérification compositionnelle, adaptées au module-checking. Considérons le cas où les hypothèses sont exprimées en LTL et les garanties en CTL*, que nous noterons $[\ \phi\ ]\ S < \psi >$. On a alors

$$\frac{\begin{array}{l} [\ \phi_2\ ]\ S^1 < \psi_1 >, \\ [\quad]\ S^1 < br(\phi_1) >, \\ [\ \phi_1\ ]\ S^2 < \psi_2 >, \\ [\quad]\ S^2 < br(\phi_2) > \end{array}}{[\quad]\ S^1 \| S^2 < \psi_1 \wedge \psi_2 >}$$

où ϕ_1 et ϕ_2 sont des formules LTL et ψ_1, ψ_2, $br(\phi_1)$ et $br(\phi_2)$ des formules CTL*, telles que $br(\phi_1)$ (resp. $br(\phi_2)$) soit la version arborescente de ϕ_1 (resp. ϕ_2). Afin de vérifier

$[\ \phi\ ]\ S < \psi >$, il est proposé de considérer le composant *étendu* $[S, \phi]$, puis de vérifier $[S, \phi] \models \psi$ (voir [KV98, KV00] pour plus de détails).

Dans [CGP03], une méthode incrémentale pour construire l'hypothèse ϕ la plus simple permettant de satisfaire la garantie ψ est proposée. Cette méthode repose sur un algorithme d'apprentissage qui construit graduellement l'hypothèse ϕ, par approximation successive.

Remarque. On pourrait aussi citer les travaux suivants [HQR98, PDH99, McM00, HMP01, FQ03, GPC04, CIP04, FFQS05] comme des applications du principe d'Hypothèse/Garantie.

Extension des composants. Dans [AS02] les composants sont spécifiés par des Structures de Kripke, et la composition est complètement asynchrone. Les auteurs proposent d'*étendre* chacun des composants avec l'ensemble de variables de l'autre composant. Pour deux composants, S^1 et S^2, on notera (S^1, V^2), l'extension de S^1. Il s'agit ensuite de vérifier la satisfaction de certaines propriétés CTL sur les extensions des composants, plutôt que sur les composants eux-mêmes, et d'en déduire leur satisfaction sur le système complet. La démarche proposée ici repose aussi sur des Hypothèses/Garanties : l'environnement d'un des composants (i.e. l'autre composant) est pris en compte dans l'extension du composant, puis les propriétés sont vérifiées sur des composants sous hypothèses.

Atteignabilité compositionnelle. Une méthode d'analyse d'atteignabilité compositionnelle est proposée dans [CGK97, CK99]. La méthode consiste à construire à partir des composants, une abstraction du système complet en masquant les étiquettes de transitions internes que l'on ne souhaite pas observer. L'automate de Büchi représentant la propriété est inclus dans la construction. Des propriétés de sûreté [CK99], mais aussi de vivacité [CGK97], sont vérifiées de cette manière.

Symétries. De nombreux travaux [ES93, CFJ93, ES96, AHI98] proposent d'exploiter des symétries entre les composants, mais aussi dans l'automate de Büchi représentant la propriété à vérifier. L'objectif est de réduire la taille de l'automate produit. Dans [And95, LL95, KLL+97], l'approche proposée (méthode du quotient) consiste à réduire progressivement le système en supprimant des composants par des opérations de masquage et de restriction.

Systèmes hiérarchiques. Plusieurs travaux concernent la vérification de systèmes à états hiérarchiques.

Dans [BLA+99, LNAH+01], une méthode d'analyse de dépendances entre les différents sous-systèmes d'un système à états hiérarchiques est proposée. Cette analyse de dépendances permet d'améliorer la vérification de certaines propriétés d'atteignabilité ou d'absence de blocages, entre autre parce qu'elle permet de considérer uniquement les sous-systèmes concernés par la propriété.

Plusieurs algorithmes à propos d'atteignabilité et de détection de cycles dans les systèmes à états hiérarchiques sont proposés dans [AY01] et le model-checking LTL est étudié.

Dans [MLSH97], des statecharts sont traduits en Promela, afin de vérifier des propriétés LTL grâce à SPIN. La même démarche est employée dans [LMM99a, LMM99b] et dans [LP99, LS99], à propos de diagrammes d'état-transition UML.

3.4 Préservation des propriétés LTL par le raffinement

Nous avons déjà indiqué que l'avantage du raffinement LTL était la préservation des propriétés abstraites : si une spécification abstraite ϕ_A satisfait une propriété ψ et qu'une spécification ϕ_R raffine ϕ_A, alors ϕ_R satisfait aussi ψ [Pnu92] (voir la section 2.1.5 à propos du raffinement LTL). Un résultat analogue est proposé dans [DS04] pour des spécifications Z [Spi92]. Le raffinement d'un modèle Z est défini en termes de simulation [DW96, DB01]. Dans [SW03, DS04], il est indiqué que pour toute propriété LTL ϕ satisfaite par une spécification abstraite Z_A, si une spécification Z_R raffine Z_A sous une relation de collage R alors ϕ est aussi satisfaite par Z_R modulo R.

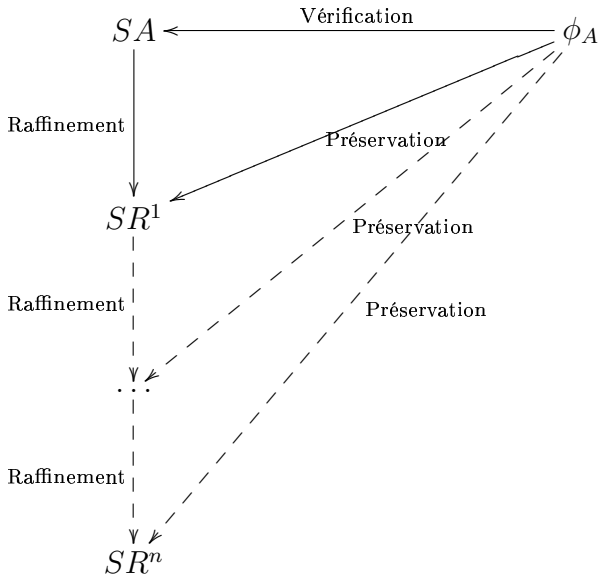


Fig. 3.8: Préservation des propriétés par le raffinement

Le raffinement des systèmes d'événements B est défini par Abrial dans [AM98] pour préserver les modalités B, qui peuvent être vues comme un sous ensemble de LTL (voir section 3.1.3. Dans [Sie97], une extension du raffinement défini par Abadi et Lamport [AL91] est proposée. Le raffinement est paramétré par une propriété LTL ϕ afin de garantir la préservation de cette propriété uniquement.

Par contre, dans [LMA01], il est montré que le raffinement de traces et le raffinement des échecs stables de CSP ne préservaient pas la LTL (voir la section 2.1.6 à propos du

raffinement CSP).

D'une manière générale, il est communément admis [Mil71, Par81, Mil89, CGP00] que les relations de simulation et de τ-simulation préservent uniquement les propriétés de sûreté $\phi_A \in LTL_{saf}$ ou, tout du moins, la partie sûreté d'une formule LTL quelconque. C'est en particulier le cas pour la relation de simulation γ et pour les τ-simulations ϱ et ρ, toutes formellement définies dans la section 2.2. Le raffinement des systèmes de transitions finis présenté dans la section 2.2 a une propriété de préservation similaire à celle de [Pnu92] : si une propriété LTL ϕ_A est satisfaite par un ST2E abstrait SA, alors elle est également satisfaite par les ST2E raffinés associés SR, SR_1... [Dar02, DJK03]. La suite de cette section présente en détail cette propriété de préservation, ainsi qu'une extension à la *reformulation* de certaines propriétés.

3.4.1 Préservation de la LTL par le raffinement de systèmes de transitions finis

Considérons deux systèmes abstraits et raffinés SA et SR. Dans notre contexte, ils sont définis sur des ensembles de variables distincts. Considérons une propriété abstraite ϕ_A, exprimée par conséquent uniquement avec les variables de SA. Afin d'établir formellement la satisfaction de ϕ_A par SR, il est nécessaire d'établir un lien entre les variables de SA en jeu dans ϕ_A et celles de SR. La satisfaction par préservation s'exprime modulo la relation de collage donnée par le prédicat de collage gp nécessaire pour le raffinement (voir définition 2.2, section 2.2). Dans un premier temps, la satisfaction d'une proposition d'état $sp_A \in SP_{V_A}$ modulo un prédicat de collage gp est définie.

Définition 3.5 (Satisfaction d'une proposition d'état modulo le collage)
Soient SA et SR deux ST2E et $gp \in SP_{V_A \cup V_R}$ leur prédicat de collage. Soit $q_R \in Q_R$. Nous dirons que $sp_A \in SP_{V_A}$ est satisfaite par q_R modulo gp, noté $q_R \models_{gp} sp_A$, si

$$\left(\bigwedge_{ap \in l_R(q_R)} (ap) \ \wedge \ gp \right) \ \Rightarrow \ sp_A$$

Il faut noter que les formules LTL contenant l'opérateur $\bigcirc$ ne seront pas préservées par le raffinement de manière "immédiate". Des transitions nouvelles introduites par le raffinement peuvent venir "s'intercaler" entre deux états qui vérifiaient la formule. Soit $\phi_A = \bigcirc sp_A$ une propriété vérifiée sur SA. On pourrait remplacer syntaxiquement dans ϕ_A, l'opérateur $\bigcirc$ par un opérateur $\Diamond$ afin de montrer sa préservation dans SR. La figure 3.9 illustre la préservation de $\bigcirc sp_A$ pour une exécution π_R où de nouvelles transitions viennent s'intercaler.

La satisfaction d'une formule LTL par préservation est formellement définie dans [DJK03]. Cette satisfaction est définie inductivement sur la structure de la formule ϕ_A, comme une extension de la définition 3.3 qui tient compte de la relation de collage liant les variables de SA à celles de SR, et de l'opérateur $\bigcirc$ vu comme comme un $\Diamond$ strict.

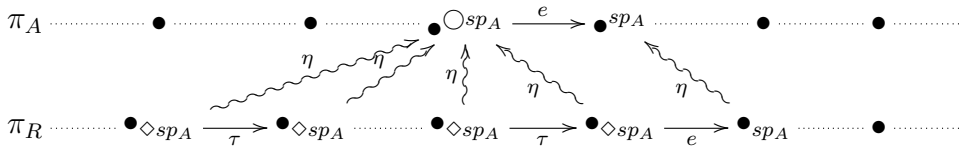


FIG. 3.9: Préservation de $\bigcirc sp_A$

Définition 3.6 (Satisfaction d'une formule LTL par préservation [DJK03])
Soient SA et SR deux ST2E et gp leur prédicat de collage. Soit $\pi \in \Pi(SR)$. Nous dirons que ϕ_A est satisfaite par préservation au temps i de l'exécution π ($i \geq 0$), noté $(\pi, i) \models_{pre} \phi_A$, si
- $(\pi, i) \models_{pre} ap_A$ *si* $\pi(i) \models_{gp} ap_A$,
- $(\pi, i) \models_{pre} \neg\phi_A$ *s'il n'est pas vrai que* $(\pi, i) \models_{pre} \phi_A$,
- $(\pi, i) \models_{pre} \phi_A \vee \phi'_A$ *si* $(\pi, i) \models_{pre} \phi_A$ *ou* $(\pi, i) \models_{pre} \phi'_A$,
- $(\pi, i) \models_{pre} \bigcirc\phi_A$ *s'il existe* j *tel que* $i < j \leq |\pi|$ *et* $(\pi, j) \models_{pre} \phi_A$
- $(\pi, i) \models_{pre} \phi_A \,\mathcal{U}\, \phi'_A$ *s'il existe* j *tel que* $i \leq j \leq |\pi|$ *et* $(\pi, j) \models_{pre} \phi'_A$, *et pour tout* k *tel que* $i \leq k < j$, *on a* $(\pi, k) \models_{pre} \phi_A$.

Un théorème à propos de la préservation d'une formule LTL quelconque par le raffinement de systèmes de transitions finis est donné dans [Dar02, DJK03]. Ce théorème généralise la définition 3.6 grâce à la proposition 2.1 à propos du raffinement de chemins.

Théorème 3.2 (Préservation d'une propriété LTL abstraite [DJK03])
Soient SA et SR deux ST2E. Soit $\phi_A \in LTL$ une formule LTL quelconque. Nous dirons que ϕ_A est préservée sur SR, noté $SR \models_{pre} \phi_A$, si $\mathcal{T}_{SA}(SR) \sqsubseteq_\eta SA$ et $SA \models \phi_A$.

La tableau 3.10 récapitule les résultats par rapport à la préservation des propriétés LTL en fonction de la relation de simulation utilisée.

	Type de simulation	Propriétés préservées
$SR \preceq_\gamma SA$	simulation forte (définition 2.4)	LTL_{saf}
$SR \preceq_\varrho SA$	τ-simulation (définition 2.5)	LTL_{saf}
$SR \preceq_\rho SA$	τ-simulation non bloquante (définition 2.6)	LTL_{saf}
$SR \sqsubseteq_\eta SA$	raffinement (définition 2.7)	LTL

FIG. 3.10: Simulations et préservation LTL

Exemple. Nous avons montré dans la section 2.2.4 que le ST2E $Robot_R$, illustré par la figure 2.11(a) était bien un raffinement de $Robot_A$.
Les propriétés LTL (3.2) à (3.7) énoncées dans la section 3.1.2 sont toutes satisfaites par $Robot_A$. Le théorème 3.2 nous donne immédiatement que toutes ces propriétés sont aussi satisfaites par préservation par $Robot_R$.

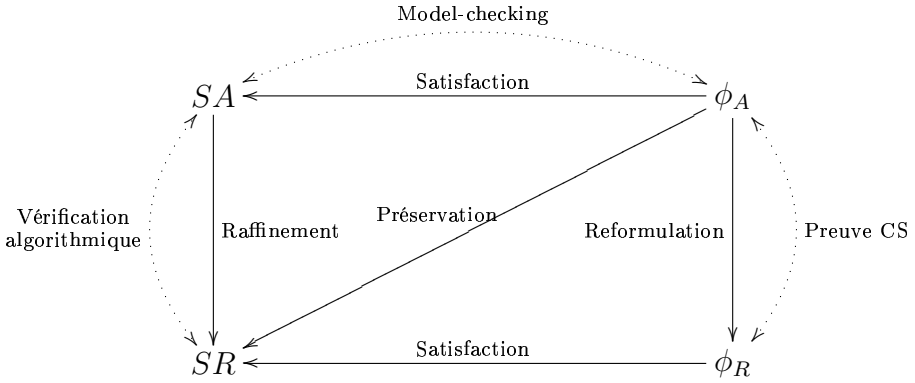


FIG. 3.11: Préservation et reformulation de propriétés

3.4.2 Reformulation de propriétés LTL

Dans la section précédente, nous avons indiqué que le raffinement de systèmes de transitions finis présenté dans la section 2.2 préservait au niveau du ST2E raffiné les propriétés LTL vérifiées par le ST2E abstrait. Néanmoins, la propriété LTL abstraite et le système raffiné sont exprimés sur des ensembles de variables distincts et l'on peut souhaiter ré-exprimer plus finement cette propriété au niveau du système raffiné. La démarche de *reformulation* ou de *raffinement* de propriétés LTL présentée dans [BDJK00, BDJK01, Dar02] consiste à ré-exprimer une propriété LTL abstraite ϕ_A en une nouvelle propriété ϕ_R du système raffiné (voir figure 3.11). Des schémas de reformulation de propriétés permettent de guider l'utilisateur lors de la ré-expression de la propriété LTL au niveau du système raffiné.

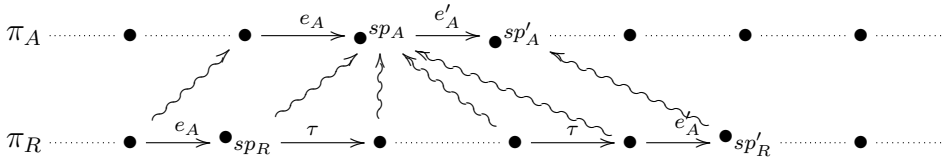


FIG. 3.12: Le schéma $\bigcirc\Diamond$

On appelle *schéma de reformulation* un couple $\#_A\#_R$ d'opérateurs temporels tels que $\#_i \in \{\bigcirc, \Diamond, \mathcal{U}, \mathcal{W}\}$, où $\#_A\#_R$ représente les opérateurs temporels intervenant dans ϕ_A et dans ϕ_R tels que ϕ_A et ϕ_R soient de la forme $\Box(sp \Rightarrow \#sp')$ (ou $\Box(sp \Rightarrow sp''\#sp')$). Considérons par exemple, un propriété abstraite $\Box(sp_A \Rightarrow \bigcirc sp'_A)$. Elle peut être raffinée par $\Box(sp_R \Rightarrow \Diamond sp'_R)$ (schéma $\bigcirc\Diamond$) ou par $\Box(sp_R \Rightarrow sp''_R \mathcal{U} sp'_R)$ (schéma $\bigcirc\mathcal{U}$). Voir les figures 3.12 et 3.13.

Considérons deux ST2E SA et SR ainsi qu'un prédicat de collage gp tels que $SR \sqsubseteq_\eta SA$. Pour certains schémas de reformulation, si ϕ_A est satisfaite par SA, alors des conditions suffisantes $CS(\phi_A, \phi_R)$ ne faisant intervenir que des propositions d'états permettent de garantir la satisfaction de ϕ_R par SR. Ces conditions sont une *condition de départ* $sp_R \wedge gp \Rightarrow sp_A$, une *condition de maintenance* $sp_A \wedge gp \Rightarrow sp''_R$ et une *condition de fin* $sp'_A \wedge gp \Rightarrow$

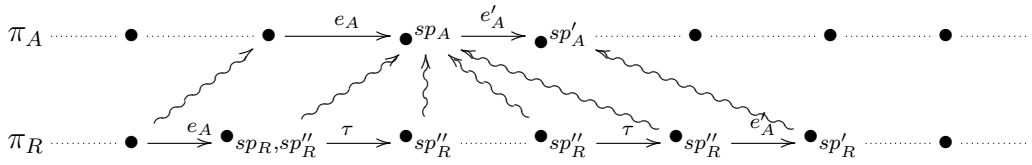


FIG. 3.13: Le schéma $\bigcirc\mathcal{U}$

sp'_R [BDJK00, BDJK01, Dar02].

Les conditions $CS(\phi_A, \phi_R)$ sont assez intuitives et relativement simples. Elles peuvent facilement être vérifiées de manière automatique : la vérification de $SR \models \phi_R$, dans le cadre d'un développement par raffinement, s'en trouve simplifiée, puisqu'il suffit alors de vérifier $SA \models \phi_A$ et $CS(\phi_A, \phi_R)$ au lieu de vérifier directement ϕ_R.

Remarque. En pratique, les conditions $CS(\phi_A, \phi_R)$ sont souvent trop restrictives. Des conditions affaiblies ont aussi été proposées dans [BDJK01, Dar02]. Ces conditions sont plus complexes, puisqu'elles font intervenir les *gardes* des transitions, mais permettent de vérifier, par reformulation, une classe plus large de propriétés.

Dans les sections 4.3 et 5.3, on souhaite réutiliser la démarche de reformulation pour des propriétés d'invariance $\Box sp_A$ / $\Box sp_R$. La proposition suivante définit la reformulation d'une propriété d'invariance.

Proposition 3.1 (Reformulation d'une propriété d'invariance)
Soient SA et SR deux ST2E et $gp \in SP_{V_A \cup V_R}$ leur prédicat de collage. Soient $\Box sp_A$ et $\Box sp_R$ deux propriétés LTL telles que $sp_A \in SP_{V_A}$ et $sp_R \in SP_{V_R}$.

$$Si \begin{cases} \mathcal{T}_{SA}(SR) \sqsubseteq_\eta SA \\ SA \models \Box sp_A \\ sp_A \wedge gp \Rightarrow sp_R \end{cases} \quad alors\ SR \models \Box sp_R$$

PREUVE. Comme $\mathcal{T}_{SA}(SR) \sqsubseteq_\eta SA$ et $SA \models \Box sp_A$, nous avons $SR \models_{pre} \Box sp_A$,
c.à.d. $\forall \pi, i.(\pi \in \Pi(SR) \wedge 0 \leq i \leq |\pi| \Rightarrow (\pi, i) \models_{pre} sp_A)$,
ce qui est équivalent à $\forall \pi, i.(\pi \in \Pi(SR) \wedge 0 \leq i \leq |\pi| \Rightarrow \pi(i) \models_{gp} sp_A)$,
c.à.d. $\forall \pi, i.(\pi \in \Pi(SR) \wedge 0 \leq i \leq |\pi| \Rightarrow (l_R(\pi(i)) \wedge gp \Rightarrow sp_A))$,
qui implique $\forall \pi, i.(\pi \in \Pi(SR) \wedge 0 \leq i \leq |\pi| \Rightarrow (l_R(\pi(i)) \wedge gp \Rightarrow sp_A \wedge gp))$.
Nous savons par hypothèse que $sp_A \wedge gp \Rightarrow sp_R$. Par transitivité de l'implication, nous obtenons $\forall \pi, i.(\pi \in \Pi(SR) \wedge 0 \leq i \leq |\pi| \Rightarrow (l_R(\pi(i)) \wedge gp \Rightarrow sp_R))$.
Comme gp n'intervient pas pour établir sp_R, nous pouvons le supprimer de la partie gauche de l'implication. Nous obtenons alors $\forall \pi, i.(\pi \in \Pi(SR) \wedge 0 \leq i \leq |\pi| \Rightarrow (l_R(\pi(i)) \Rightarrow sp_R))$,
c.à.d. $\forall \pi, i.(\pi \in \Pi(SR) \wedge 0 \leq i \leq |\pi| \Rightarrow (\pi, i) \models sp_R)$.
On a donc bien $SR \models \Box sp_R$.

□

3.5 Conclusion

Ce chapitre rappelle la Logique Temporelle Linéaire (LTL) comme l'un des moyens d'exprimer des propriétés sur les comportements dynamiques des systèmes. Les propriétés d'invariance sont vues comme des formes simplifiées de formules LTL qui expriment des propriétés statiques à propos du système à vérifier. Nous situons ensuite la LTL par rapport à d'autres logiques temporelles (CTL*, CTL et Past+LTL), avant d'identifier différentes classes de propriétés exprimables (ou non) en LTL en fonction de leur signification : nous insistons en particulier sur la distinction propriétés de sûreté / propriétés de vivacité. Après un rappel à propos de l'algorithme classique de model-checking LTL, nous donnons quelques idées à propos de la vérification compositionnelle de propriétés et nous présentons en particulier la démarche d'*Hypothèse/Garantie*.

Le choix de la LTL pour spécifier les propriétés des systèmes se justifie avant tout par rapport à la définition du raffinement avec laquelle nous avons choisi de travailler. Comme nous l'indiquons dans ce chapitre, la relation de raffinement η préserve au niveau du système raffiné les propriétés vérifiées sur le système abstrait : la vérification algorithmique des propriétés LTL peut être utilisée pour s'assurer que le modèle le plus abstrait satisfait une formule temporelle, c'est en effet à ce niveau que le nombre d'états à explorer est le plus petit. La satisfaction de la propriété est garantie sur les systèmes raffinés, plus complexes et plus importants.

Néanmoins, le théorème de compositionnalité de [Rab01] nous a réconforté dans notre choix de la LTL : il a été montré que la vérification compositionnelle de propriétés dynamiques échouait pour des opérateurs de composition même très simples dès lors qu'il est possible d'énoncer une propriété exprimant "qu'il existe un chemin tel que tous les états du chemin vérifient une propriétés p" [Rab01], ce qui n'est pas le cas en LTL : nous verrons d'ailleurs dans les chapitres 4 et 5 que nous pourrons vérifier de manière compositionnelle un certain nombre de propriétés LTL, en particulier les propriétés d'invariance et de sûreté et que ces vérifications compositionnelles sont conciliables avec le raffinement.

Chapitre 4

Raffinement par décomposition

Dans ce chapitre, nous allons présenter une méthode générale pour la vérification par décomposition du raffinement d'un système. Cette méthode s'appuie avant tout sur la *décomposition* d'un système en plusieurs sous-systèmes permettant l'étude séparée de chacun des sous-systèmes : un raffinement *affaibli* pour chaque sous-système permet de conclure à propos du raffinement du système complet.

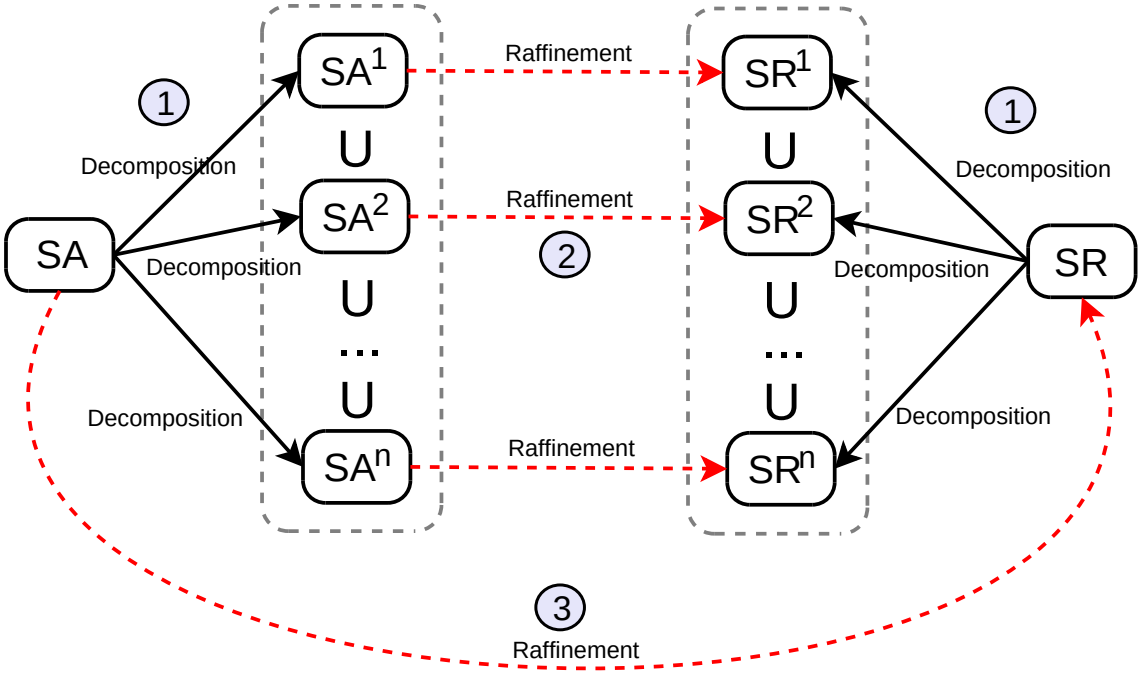


Fig. 4.1: Raffinement par décomposition

Dans la section 4.1, nous présentons la notion de *décomposition* d'un ST2E en plusieurs sous-systèmes. Cette décomposition peut être vue comme une sorte d'"union" très générale de plusieurs sous-systèmes permettant de ré-obtenir le système complet.

La section 4.2 présente une étude à propos de la compositionnalité de la relation de raffinement η vis à vis de l'opération de décomposition. Considérons un système abstrait SA décomposé en SA^1, SA^2,..., SA^n ainsi qu'un système raffiné SR décomposé en SR^1, SR^2,..., SR^n. Etudier la relation de raffinement η entre les sous-systèmes SR^1 et SA^1, SR^2 et SA^2, ..., et SR^n et SA^n ne suffit pas. La décomposition fait apparaître de "faux" états de blocages dans les sous-systèmes raffinés et masque l'entrelacement des "nouvelles" transitions. Finalement, nous montrons qu'il suffit d'étudier une relation de raffinement *plus faible* entre chaque sous-système issu de la décomposition pour pouvoir conclure à propos du raffinement de SA par SR. Décomposition et raffinement sont donc rendus compatibles.

Nous bénéficions toujours de la préservation pour le système complet raffiné des propriétés LTL vérifiées au niveau abstrait, comme il est précisé dans la section 3.4. Nous verrons qu'il est quelquefois possible de combiner décomposition, raffinement et vérification de propriétés LTL. Plus précisément, nous verrons dans la section 4.3 que la vérification de certaines propriétés du système complet peut être effectuée de manière compositionnelle, en

réutilisant les sous-systèmes issus de la décomposition du système complet : de manière plus précise, il s'agit de vérifier compositionnellement des propriétés d'invariance, des invariants dynamiques, des propriétés d'(in)atteignabilité et la non satisfaction des propriétés LTL de sûreté en général.

Remarque. La démarche présentée dans ce chapitre est "similaire" à celle proposée, au niveau des systèmes d'événements B, par Abrial dans [Abr02].

4.1 Décomposition d'un système

Supposons que nous souhaitions étudier un système de transitions S de très grande taille, que nous imaginons ne pas pouvoir appréhender de manière complète. Une solution consiste alors à décomposer ce système en plusieurs sous-systèmes, et à étudier séparément chacun d'entre eux. Nous parlons alors de la décomposition d'un système.

Définissons d'abord la notion de sous-système.

Définition 4.1 (Sous-système)
Soit $S = (Q, Q_0, E, T, V, l)$ un ST2E. $S' = (Q', Q'_0, E', T', V', l')$ est un sous-système de S si :
- $Q' \subseteq Q$,
- $Q'_0 \subseteq Q_0$,
- $E' \subseteq E$,
- $T' \subseteq T$,
- $V' = V$,
- $\forall q \in Q', l'(q) = l(q)$

Notons que dans un sous-système, on ne considérera pas que l'ensemble des états Q' est atteignable à partir des états initiaux de Q'_0, via T'. Nous définissons maintenant la décomposition d'un ST2E en deux sous-systèmes, comme l'opération qui à partir de deux sous-systèmes, nous redonne le système complet.

Définition 4.2 (Décomposition d'un ST2E)
Soit $S = (Q, Q_0, E, T, V, l)$ un ST2E. Soient $S^1 = (Q^1, Q^1_0, E^1, T^1, V, l^1)$ et $S^2 = (Q^2, Q^2_0, E^2, T^2, V, l^2)$ deux sous-systèmes de S. S^1 et S^2 forment une décomposition de S, notée $S = S^1 \cup S^2$ si :
- $Q = Q^1 \cup Q^2$,
- $Q_0 = Q^1_0 \cup Q^2_0$,
- $E = E^1 \cup E^2$,
- $T = T^1 \cup T^2$,
- $\forall q \in Q,\ l(q) = \begin{cases} l^1(q) \text{ si } q \in Q^1 \setminus Q^2 \\ l^2(q) \text{ si } q \in Q^2 \setminus Q^1 \\ l^1(q) = l^2(q) \text{ si } q \in Q^1 \cap Q^2 \end{cases}$

Remarque. L'opérateur de décomposition $\cup$ défini précédemment est un opérateur très générique qui autorise des décompositions plus particulières comme par exemple une *partition* de l'espace des transitions.

Les systèmes de transitions séparables [BQ96] présentés section 1.2.4 peuvent, par exemple, être vus comme un cas particulier de l'opérateur de décomposition présenté ici.

La décomposition de S en deux sous-systèmes S^1 et S^2 peut aussi être vue comme le résultat d'une opération de masquage. En effet, on peut considérer que le système S^1 est le résultat d'une opération qui consiste à cacher certains comportements de S, et plus précisément, à enlever de S les transitions étiquetées par des éléments de E^2. On a alors

$$S \;=\; \mathcal{H}(S, E^2) \;\cup\; \mathcal{H}(S, E^1) \text{ si } E^1 \cup E^2 = E$$

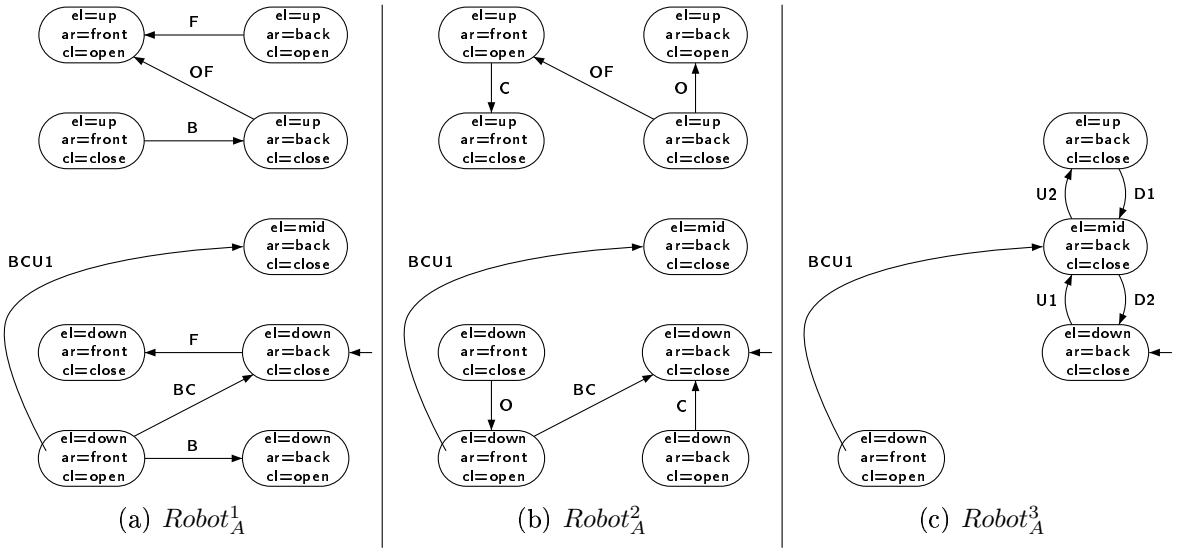
La décomposition d'un ST2E en deux sous-systèmes peut être généralisée à n sous-systèmes de manière évidente, l'opérateur $\cup$ de deux sous-systèmes étant commutatif et associatif : on parlera alors de *décomposition généralisée*.

Propriété 4.1 (Associativité de la décomposition)
Soit $S = (Q, Q_0, E, T, V, l)$ un ST2E. Soient S^1, S^2 et S^3 trois sous-systèmes de S. On a

$$(S^1 \cup S^2) \cup S^3 = S^1 \cup (S^2 \cup S^3)$$

PREUVE. Evidente grâce à l'associativité de l'union ensembliste.

□



(a) $Robot_A^1$ (b) $Robot_A^2$ (c) $Robot_A^3$

FIG. 4.2: Décomposition de $Robot_A$

Exemple. Considérons l'exemple du bras mobile robotisé, présenté dans les sections 1.1.4 et 2.2.4. On peut envisager de décomposer $Robot_A$ de la manière

illustrée par la figure 4.2. Les trois ST2E $Robot_A^1$, $Robot_A^2$ et $Robot_A^3$ sont définis sur le même ensemble de variables que le système complet $Robot_A$ et l'union de leurs relations de transitions est bien égale à la relation de transition de $Robot_A$. Il s'agit donc bien de sous-systèmes de $Robot_A$ et ils en forment une décomposition : $Robot_A = Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$.

Nous verrons dans le chapitre 5 une manière systématique pour obtenir une décomposition d'un produit synchronisé qui nous permettra d'étudier le raffinement du produit à partir de celui de ses composants.

4.2 Raffinement par décomposition

Supposons que l'on ait un système abstrait $SA = SA^1 \cup SA^2$ qui se décompose ainsi qu'un système raffiné $SR = SR^1 \cup SR^2$ qui se décompose lui aussi. Serait-il possible d'étudier le raffinement des sous-systèmes issus de la décomposition et de conclure à propos du raffinement du système complet ? Autrement dit, est-ce que $SR^1 \sqsubseteq_\eta SA^1$ et $SR^2 \sqsubseteq_\eta SA^2$ suffisent à conclure $SR \sqsubseteq_\eta SA$?

Nous verrons que cette démarche n'est pas suffisante, mais qu'il s'agit toutefois d'une bonne intuition dans de nombreux cas. Pour cela, nous étudierons différents exemples, avant de formaliser notre démarche.

4.2.1 Renommage par τ et décomposition

La première étape pour la vérification du raffinement consiste à renommer par τ les étiquettes des transitions introduites par le raffinement afin de les rendre inobservables. Montrons que le renommage par τ présenté dans la définition 2.3 est compatible avec la décomposition.

Lemme 4.1 (Décomposition vs. renommage par τ)
Soient $SA^1 \cup SA^2$ et $SR^1 \cup SR^2$ deux ST2E. On a l'égalité suivante :

$$\mathcal{T}_{SA^1 \cup SA^2}(SR^1 \cup SR^2) = \mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2)$$

PREUVE. Montrons que quelque soit la transition de $SR^1 \cup SR^2$, elle est présente dans l'un des sous-systèmes et elle est renommée de la même manière dans $\mathcal{T}_{SA^1 \cup SA^2}(SR^1 \cup SR^2)$ et dans $\mathcal{T}_{SA^1}(SR^1)$ ou $\mathcal{T}_{SA^2}(SR^2)$. Les différents cas dépendent de l'étiquette de la transition $q_R \xrightarrow{e} q'_R$ considérée :

- $e \in E_A^1 \cap E_A^2$. La transition est présente dans SR^1 et SR^2. Elle n'est pas renommée dans $\mathcal{T}_{SA^1 \cup SA^2}(SR^1 \cup SR^2)$, ni dans $\mathcal{T}_{SA^1}(SR^1)$, ni dans $\mathcal{T}_{SA^2}(SR^2)$.
- $e \in E_A^1$. La transition est présente dans SR^1. Elle n'est pas renommée dans $\mathcal{T}_{SA^1 \cup SA^2}(SR^1 \cup SR^2)$, ni dans $\mathcal{T}_{SA^1}(SR^1)$.
- $e \in E_A^2$. La transition est présente dans SR^2. Elle n'est pas renommée dans $\mathcal{T}_{SA^1 \cup SA^2}(SR^1 \cup SR^2)$, ni dans $\mathcal{T}_{SA^2}(SR^2)$.

- $e \in (E_R^1 \cap E_R^2) \setminus (E_A^1 \cap E_A^2)$. La transition est présente dans SR^1 et SR^2. Elle est renommée par τ dans $\mathcal{T}_{SA^1 \cup SA^2}(SR^1 \cup SR^2)$, dans $\mathcal{T}_{SA^1}(SR^1)$ et dans $\mathcal{T}_{SA^2}(SR^2)$.
- $e \in E_R^1 \setminus E_A^1$. La transition est présente dans SR^1. Elle est renommée par τ dans $\mathcal{T}_{SA^1 \cup SA^2}(SR^1 \cup SR^2)$ et dans $\mathcal{T}_{SA^1}(SR^1)$.
- $e \in E_R^2 \setminus E_A^2$. La transition est présente dans SR^2. Elle est renommée par τ dans $\mathcal{T}_{SA^1 \cup SA^2}(SR^1 \cup SR^2)$ et dans $\mathcal{T}_{SA^2}(SR^2)$.

□

Puisque nous avons montré que la décomposition et le renommage par τ étaient compatibles, nous pouvons maintenant vérifier si l'étude du raffinement de sous-systèmes suffit à conclure à propos du raffinement du système complet, en commençant par étudier quelques exemples.

4.2.2 Exemple de raffinement par décomposition

Pour cet exemple, nous considérons toujours le cas du bras mobile robotisé présenté dans les sections 1.1.4 et 2.2.4. Une décomposition du ST2E abstrait $Robot_A = Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$ a été donnée figure 4.2. Nous étudions le ST2E $Robot_B$, un raffinement de $Robot_A$ tel que $\mathcal{T}(Robot_B) \sqsubseteq_\eta Robot_A$. Le ST2E $Robot_B$ est illustré par la figure 4.3.

Remarque. Il s'agit d'une version simplifiée du raffinement $Robot_R$ présenté figure 2.11(a), section 2.2.4.

Considérons par exemple la décomposition suivante de $Robot_B = Robot_B^1 \cup Robot_B^2 \cup Robot_B^3$, illustrée par la figure 4.4. Nous pouvons étudier successivement si la relation de raffinement η définie section 2.2 est correcte pour chacun des sous-systèmes. On remarque facilement qu'elle peut être établie pour chacun des sous-systèmes : $\mathcal{T}(Robot_B^1) \sqsubseteq_\eta Robot_A^1$, $\mathcal{T}(Robot_B^2) \sqsubseteq_\eta Robot_A^2$ et $\mathcal{T}(Robot_B^3) \sqsubseteq_\eta Robot_A^3$.

Sur cet exemple, il semblerait que l'on puisse conclure à propos du raffinement d'un système complet uniquement en étudiant le raffinement de chacun de ses sous-systèmes, étant donné que l'étude séparée du raffinement de chacun des sous-systèmes recouvre l'ensemble des transitions du système complet, en vertu de la définition 4.2.

4.2.3 Autre exemple de raffinement par décomposition : le problème des états de blocage

Nous considérons toujours l'exemple du bras mobile robotisé. Le ST2E abstrait $Robot_A$ est donné section 2.2.4 et une décomposition $Robot_A = Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$ est donnée section 4.1. Dans la section 2.2.4, nous avons introduit le ST2E $Robot_R$ comme un raffinement de $Robot_A$, figure 2.11(a) et nous avons montré que $\mathcal{T}(Robot_R) \sqsubseteq_\eta Robot_A$ est correct.

La figure 4.5 illustre une décomposition possible de $Robot_R = Robot_R^1 \cup Robot_R^2 \cup Robot_R^3$. Etudions le raffinement pour chacun des sous-systèmes, comme dans l'exemple précédent. La relation de raffinement η peut être établie pour les sous-systèmes $Robot_R^2$ et $Robot_R^3$, et

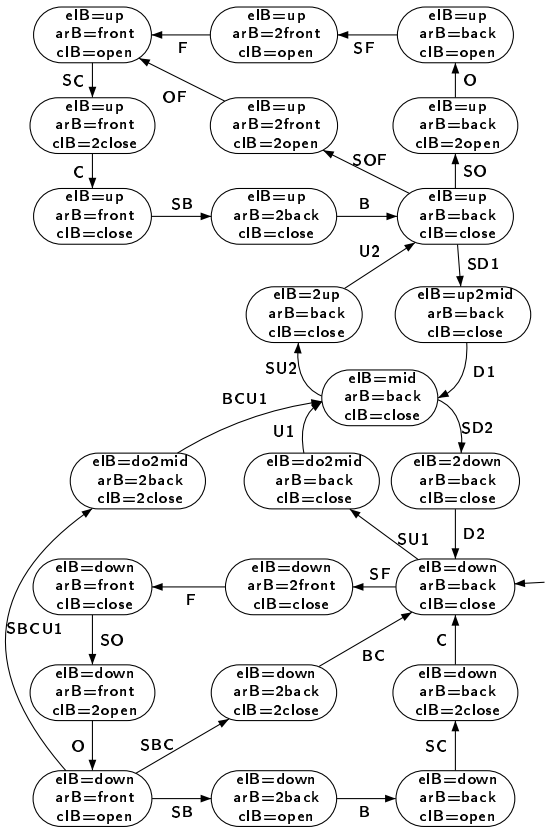
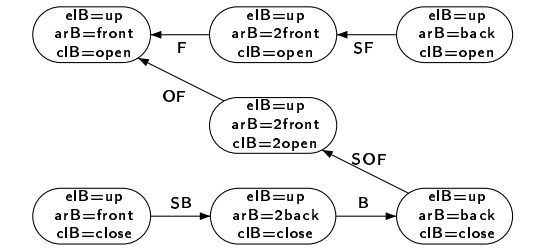
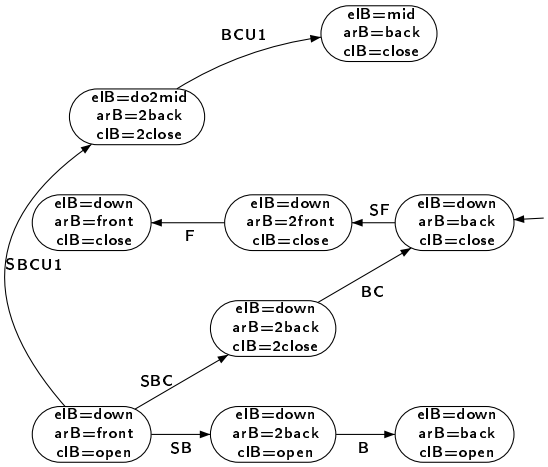
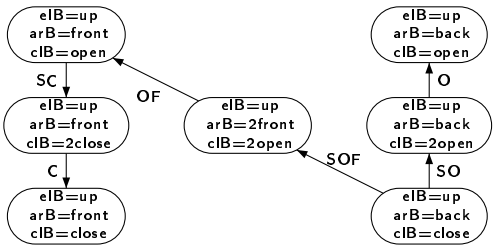


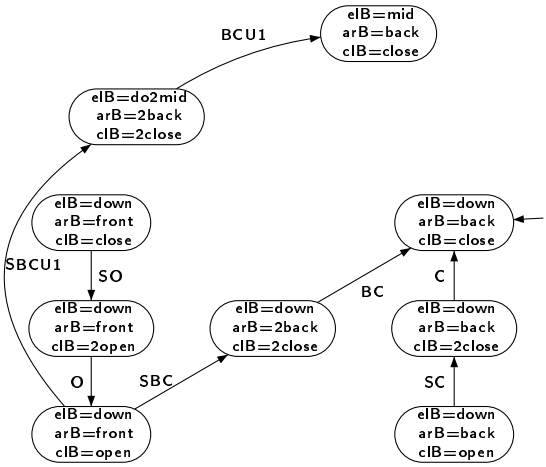
Fig. 4.3: Système raffiné $Robot_B$



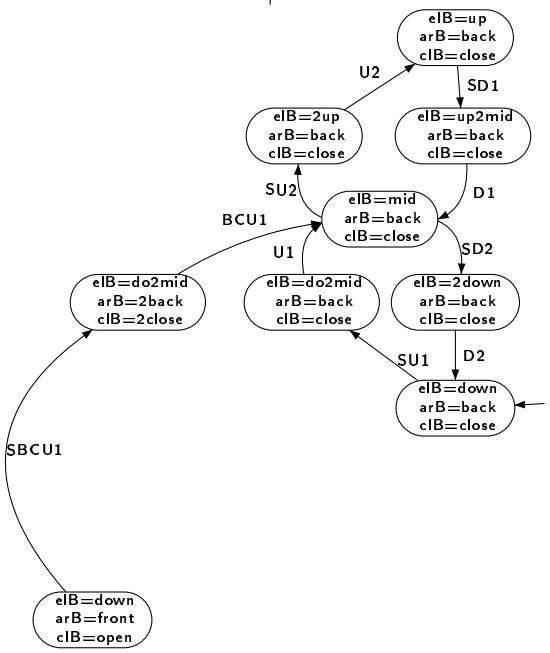


(a) $Robot^1_B$





(b) $Robot^2_B$



(c) $Robot^3_B$

FIG. 4.4: Décomposition de $Robot_B$

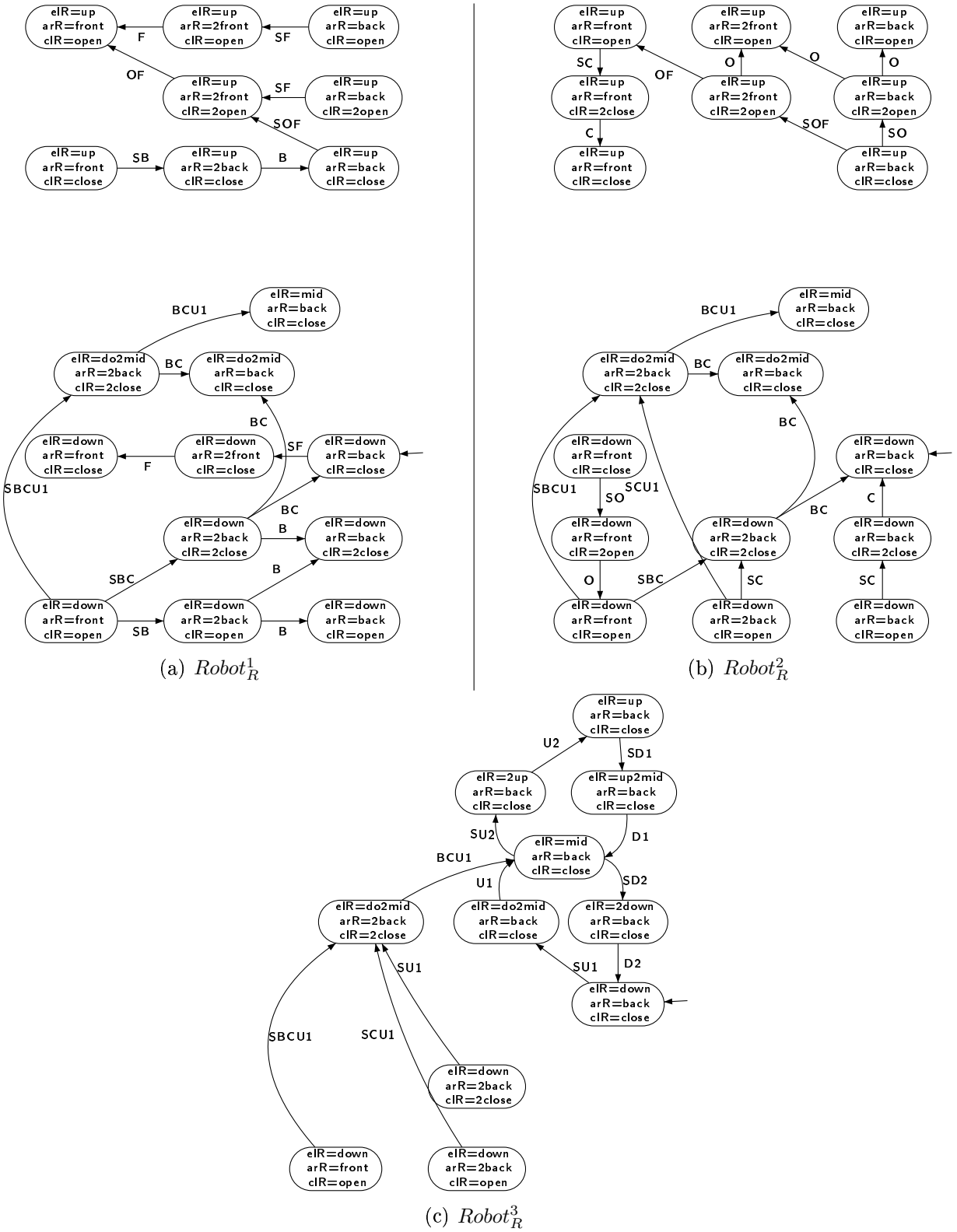


Fig. 4.5: Décomposition de $Robot_R$

l'on a $\mathcal{T}(Robot_R^2) \sqsubseteq_\eta Robot_A^2$ et $\mathcal{T}(Robot_R^3) \sqsubseteq_\eta Robot_A^3$.
Dans le cas du sous-système $Robot_R^1$, l'absence de nouveaux blocages de la définition 2.6 de la relation de τ-simulation non-bloquante n'est pas vérifiée. La figure 4.6 illustre la raison de cet échec : l'état q_e du système $\mathcal{T}(Robot_R^1)$ est un état de blocage alors qu'il est lié par la relation de collage à un état non-bloquant du système abstrait $Robot_A^1$: On ne peut donc pas établir que $\mathcal{T}(Robot_R^1) \preceq_\rho Robot_A^1$, et donc, encore moins que $\mathcal{T}(Robot_R^1) \sqsubseteq_\eta Robot_A^1$.

L'étude de la relation de raffinement η pour chacun des sous-systèmes – issus de la décomposition du système complet – n'est pas toujours suffisante pour conclure à propos du raffinement du système complet, même dans des cas où le raffinement du système complet est avéré. Sur l'exemple précédent, l'état de blocage q_e qui fait échouer la vérification du raffinement du sous-système $\mathcal{T}(Robot_R^1)$, n'est pas un état de blocage dans les autres sous-systèmes ni dans le système complet d'ailleurs.

La τ-simulation non-bloquante ρ est trop restrictive quand il s'agit d'étudier le raffinement de sous-systèmes issus d'une décomposition. L'opération de décomposition décompose l'ensemble de transitions du système complet et répartit donc le comportement du système complet entre les différents sous-systèmes. Elle introduit par conséquent, ce que nous appellerons, de *"faux"* états de blocage, c.à.d., des états qui font échouer la vérification de la τ-simulation non-bloquante dans un des sous-systèmes, mais qui ne sont pas des états de blocage dans le système complet. Une idée pour contourner ce problème est de redéfinir une τ-simulation non-bloquante moins stricte que ρ, et qui n'échouerait pas lorsque l'on rencontre un des ces "faux" états de blocage. Cependant, elle doit aussi permettre de s'assurer qu'il s'agit bien de "faux" blocages et non de "vrai" blocages.

4.2.4 Entrelacement des τ-transitions

Un autre problème peut faire échouer la vérification du raffinement par décomposition : la décomposition en sous-systèmes peut cacher l'entrelacement des τ-transitions dans le système complet et masquer certains τ-cycles présents dans le système complet.

Considérons des ST2E très simples $SA = SA^1 \cup SA^2$ et $SR = SR^1 \cup SR^2$, illustrés par les figures 4.7 et 4.8.

En considérant la relation de collage suivante $\mu = \{(a0, r0), (a1, r1), (a1, r2), (a1, r3)\}$, nous pourrions montrer que SR^1 raffine SA^1 et que SR^2 raffine SA^2 alors qu'il est évident que SR ne raffine pas SA : la présence du τ-cycle $r_1 \xrightarrow{\tau} r_2 \xrightarrow{\tau} r_3 \xrightarrow{\tau} r_1 \ldots$ dans $\mathcal{T}(SR)$ fait échouer la construction de la relation de raffinement η (définition 2.7).

Nous ne prendrons plus en compte la clause à propos de l'absence de nouveaux τ-cycles. En effet, étudier l'absence de τ-cycles dans les sous-systèmes ne permet aucunement de conclure à propos de l'absence de τ-cycles dans le système complet à cause de l'entrelacement possible des τ-transitions dans les différents sous-systèmes. Nous noterons $div^\tau(\mathcal{T}_{SA}(SR)) \stackrel{\text{def}}{=} \exists q, \sigma, i.\ (q \in Q_R \wedge \sigma \in \Sigma(q) \wedge i \geq 1 \wedge tr(\sigma) = \tau^\omega \wedge (\sigma, i) = q)$ un prédicat qui indique l'existence d'un τ-cycle dans $\mathcal{T}_{SA}(SR)$. Le prédicat $\neg\, div^\tau(\mathcal{T}_{SA}(SR))$

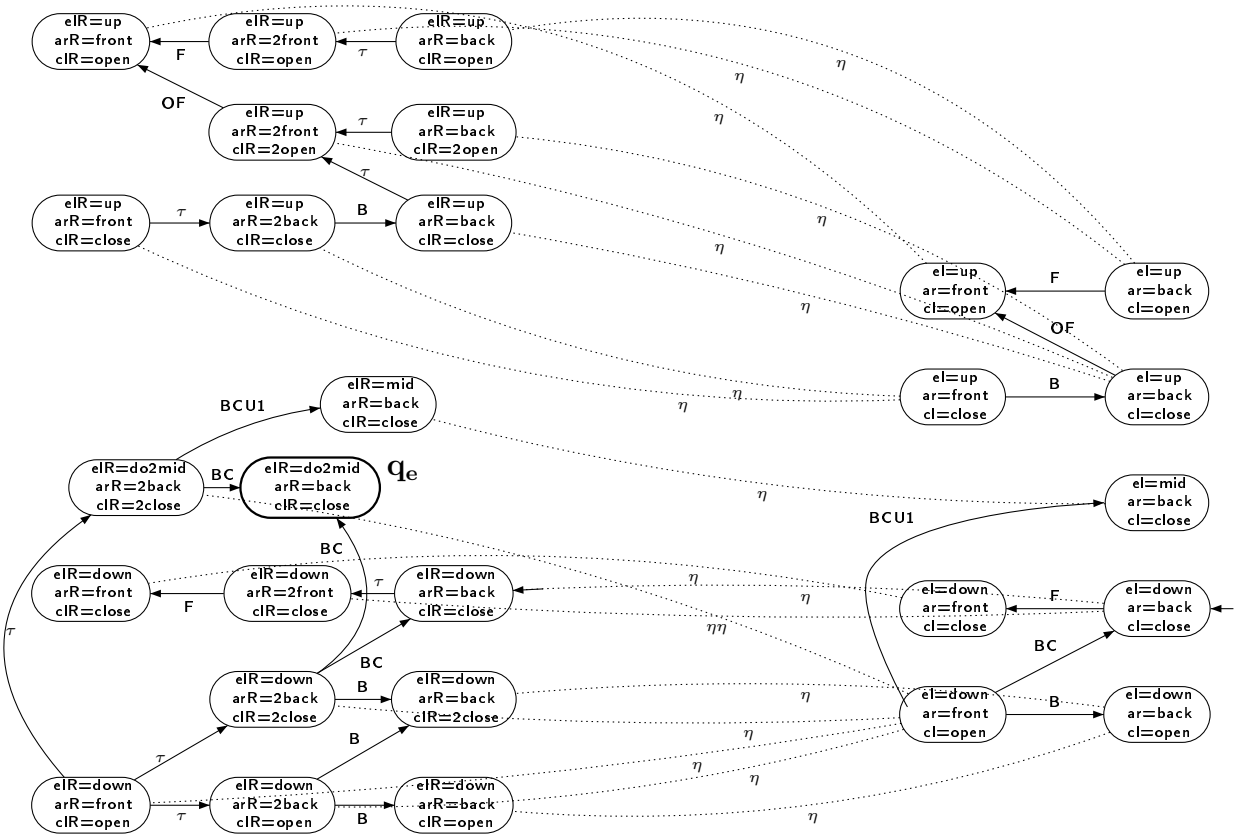
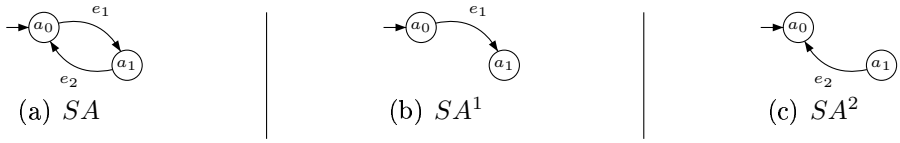
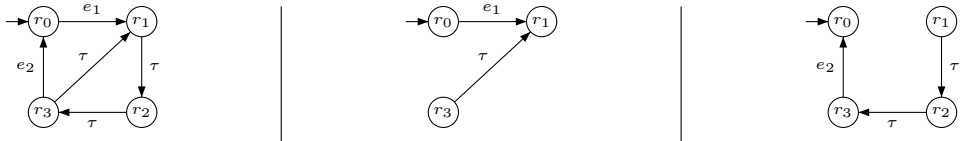


Fig. 4.6: Relation de raffinement entre $\mathcal{T}(Robot^1_R)$ et $Robot^1_A$



(a) SA (b) SA^1 (c) SA^2

Fig. 4.7: Décomposition de $SA = SA^1 \cup SA^2$



(a) $\mathcal{T}(SR)$ (b) $\mathcal{T}(SR^1)$ (c) $\mathcal{T}(SR^2)$

Fig. 4.8: Décomposition de $SR = SR^1 \cup SR^2$

correspond bien à l'absence de τ-cycles – généralisée à tout le système – de la définition 2.7.

4.2.5 Raffinement affaibli

Afin de tenir compte des spécificités de la vérification du raffinement par décomposition, nous allons donner une définition affaiblie du raffinement. Pour cela nous affaiblirons la relation de τ-simulation non-bloquante donnée définition 2.6.

Nous considérerons toujours les relations de simulation γ (déf. 2.4) et de τ-simulation ϱ (déf. 2.5). Nous allons définir une nouvelle relation *affaiblie* de τ-simulation non-bloquante afin de prendre en compte les "faux" états de blocage, c.à.d. les états qui font échouer la vérification de ρ. Pour cela, nous allons considérer l'ensemble $D \subseteq Q_R$ contenant ces "faux" états de blocage.

Définition 4.3 (Relation affaiblie de τ-simulation non-bloquante)
Soient SA et $\mathcal{T}_{SA}(SR)$ deux ST2E. Soit $e \in E_A$. D est le plus petit sous-ensemble de Q_R et ρ_w est la plus grande relation binaire incluse dans ϱ qui vérifient :

$$((q_R\ \rho_w\ q_A \wedge\ q_R \nrightarrow) \Rightarrow ((q_A \nrightarrow \wedge q_R \notin D) \vee (q_A \rightarrow \wedge q_R \in D)))$$

A propos de vérification : si une transition du système raffiné introduit un état de blocage, alors nous regardons s'il s'agit d'un ancien état de blocage (qui peut alors être conservé) ou d'un nouvel état de blocage (introduit par le raffinement). S'il s'agit d'un nouveau blocage, alors l'état est sauvé dans l'ensemble des nouveaux états de blocage D. Nous vérifierons par la suite si les états contenus dans D sont de "vrais" états de blocage introduits par une erreur dans la spécification ou de "faux" états de blocage introduits par la décomposition (voir section 4.2.3).

La relation affaiblie de τ-simulation est généralisée au niveau des ST2E raffiné et abstrait par la définition suivante. Elle définit le *raffinement affaibli* de systèmes de transitions.

Définition 4.4 (Raffinement affaibli)
Soient SA et SR deux ST2E. Nous dirons que SR raffine faiblement SA, noté $\mathcal{T}_{SA}(SR) \sqsubseteq^{D}_{\rho_w} SA$, ssi $\forall q_R.\ (q_R \in Q_R \Rightarrow \exists q_A.\ (q_A \in Q_A\ \wedge\ q_R\ \rho_w\ q_A))$.

Nous pouvons remarquer que la relation affaiblie de τ-simulation non-bloquante ρ_w est moins restrictive que la relation de τ-simulation non-bloquante ρ. En fait, nous pouvons remarquer que ρ_w s'intercale entre ϱ et ρ dans le diagramme d'inclusion des différentes relations définissant le raffinement (voir figure 4.9).

Un lien peut facilement être établi entre le raffinement strict (définition 2.8) et le raffinement affaibli. Ce lien nous permettra de conclure – dans certains cas – à propos du raffinement strict d'un système en étudiant son raffinement affaibli.

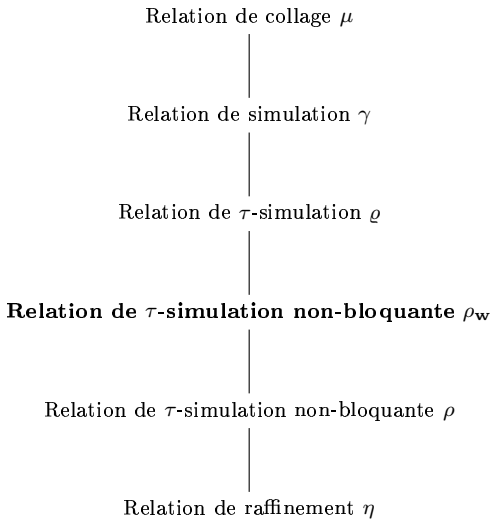


FIG. 4.9: Diagramme d'inclusion des relations définissant le raffinement

Lemme 4.2 (Raffinement strict vs. raffinement affaibli)
Soient SA et SR deux ST2E.

$$\mathcal{T}_{SA}(SR) \sqsubseteq_{\eta} SA \text{ ssi } \mathcal{T}_{SA}(SR) \sqsubseteq^{D}_{\rho_w} SA \text{ et } D = \varnothing \text{ et } \neg\, div^{\tau}(\mathcal{T}_{SA}(SR))$$

PREUVE. Considérons $\mathcal{T}_{SA}(SR)$ et SA tel que $\mathcal{T}_{SA}(SR) \sqsubseteq^{D}_{\rho_f} SA$ et $D = \varnothing$ et $\neg\, div^{\tau}(\mathcal{T}_{SA}(SR))$. Montrons alors que $\mathcal{T}_{SA}(SR) \sqsubseteq_{\eta} SA$.

1. *Simulation γ :* puisque nous avons $\mathcal{T}_{SA}(SR) \sqsubseteq^{D}_{\rho_w} SA$, nous avons γ.
2. *τ-simulation ϱ :* puisque nous avons $\mathcal{T}_{SA}(SR) \sqsubseteq^{D}_{\rho_w} SA$, nous avons ϱ.
3. *τ-simulation non bloquante ρ :* puisque nous avons $\mathcal{T}_{SA}(SR) \sqsubseteq^{D}_{\eta_f} SA$, nous avons ρ_w et ρ_w est équivalente à ρ puisque $D = \varnothing$.
4. *Raffinement η :* par définition, le prédicat $\neg\, div^{\tau}(\mathcal{T}_{SA}(SR))$ est équivalent à la clause définissant la relation η.

La réciproque est aussi évidente.

□

4.2.6 Raffinement affaibli par décomposition

L'étude du raffinement affaibli des sous-systèmes issus d'une décomposition, nous permettra de conclure à propos du raffinement du système complet, et cela, de manière compositionnelle. En d'autres termes, nous allons montrer que les raffinements affaiblis $\mathcal{T}_{SA^1}(SR^1) \sqsubseteq^{D_1}_{\rho_w} SA^1$ et $\mathcal{T}_{SA^2}(SR^2) \sqsubseteq^{D_2}_{\rho_w} SA^2$ sont suffisants pour conclure à propos de la correction du raffinement affaibli $\mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2) \sqsubseteq^{D_1 \triangle D_2}_{\rho_w} SA^1 \cup SA^2$, puis du

raffinement strict $\mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2) \sqsubseteq_\eta SA^1 \cup SA^2$.

Il est nécessaire pour cela de savoir décider si les états de blocage – issus du calcul de la relation de raffinement affaiblie pour les sous-systèmes – sont ou non de nouveaux états de blocage pour le raffinement affaibli du système complet. Considérons les raffinements affaiblis $\mathcal{T}_{SA^1}(SR^1) \sqsubseteq_{\rho_w}^{D_1} SA^1$ et $\mathcal{T}_{SA^2}(SR^2) \sqsubseteq_{\rho_w}^{D_2} SA^2$. Un état $q_R \in D_1$ (resp. $q_R \in D_2$) est un nouvel état de blocage pour le raffinement affaibli du système complet, i.e. $q_R \in D_1 \triangle D_2$, si

- q_R est aussi un état de blocage identifié lors de l'analyse du raffinement affaibli de $\mathcal{T}_{SA^2}(SR^2)$ (resp. de $\mathcal{T}_{SA^1}(SR^1)$), c.à.d. $q_R \in D_2$ (resp. $q_R \in D_1$), ou
- q_R n'est pas un état de $\mathcal{T}_{SA^2}(SR^2)$ (resp. de $\mathcal{T}_{SA^1}(SR^1)$), c.à.d. $q_R \notin Q_R^2$ (resp. $q_R \notin Q_R^1$).

Définition 4.5 (Réduction des ensembles de blocage)
Soient $SA^1 \cup SA^2$ et $SR^1 \cup SR^2$ deux ST2E. Considérons $\mathcal{T}_{SA^1}(SR^1) \sqsubseteq_{\rho_w}^{D_1} SA^1$ et $\mathcal{T}_{SA^2}(SR^2) \sqsubseteq_{\rho_w}^{D_2} SA^2$. L'ensemble des états de blocage pour le raffinement affaibli de $\mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2)$, noté $D_1 \triangle D_2$, est défini de la manière suivante :

$$D_1 \triangle D_2 = (D_1 \cap D_2) \cup (D_1 \setminus Q_R^2) \cup (D_2 \setminus Q_R^1)$$

Propriété 4.2 (Associativité de la réduction des ensembles de blocage)
Soient $SA^1 \cup SA^2 \cup SA^3$ et $SR^1 \cup SR^2 \cup SR^3$ deux ST2E tels que $\mathcal{T}_{SA^1}(SR^1) \sqsubseteq_{\rho_w}^{D_1} SA^1$, $\mathcal{T}_{SA^2}(SR^2) \sqsubseteq_{\rho_w}^{D_2} SA^2$ et $\mathcal{T}_{SA^3}(SR^3) \sqsubseteq_{\rho_w}^{D_3} SA^3$. On a

$$(D_1 \triangle D_2) \triangle D_3 = D_1 \triangle (D_2 \triangle D_3)$$

PREUVE. Montrons que $(D_1 \triangle D_2) \triangle D_3$ est bien égal à $D_1 \triangle (D_2 \triangle D_3)$.
$(D_1 \triangle D_2) \triangle D_3 = ((D_1 \triangle D_2) \cap D_3) \cup ((D_1 \triangle D_2) \setminus Q_R^3) \cup (D_3 \setminus (Q_R^1 \cup Q_R^2)$

1. $(D_1 \triangle D_2) \cap D_3 = (D_1 \cap D_2 \cap D_3) \cup ((D_1 \cap D_3) \setminus Q_R^2) \cup ((D_2 \cap D_3) \setminus Q_R^1)$
2. $(D_1 \triangle D_2) \setminus Q_R^3 = ((D_1 \cap D_2) \setminus Q_R^3) \cup (D_1 \setminus (Q_R^2 \cup Q_R^3)) \cup (D_2 \setminus (Q_R^1 \cup Q_R^3))$
3. $D_3 \setminus (Q_R^1 \cup Q_R^2)$

$= (D_1 \cap D_2 \cap D_3) \cup ((D_1 \cap D_3) \setminus Q_R^2) \cup ((D_2 \cap D_3) \setminus Q_R^1) \cup ((D_1 \cap D_2) \setminus Q_R^3) \cup (D_1 \setminus (Q_R^2 \cup Q_R^3)) \cup (D_2 \setminus (Q_R^1 \cup Q_R^3)) \cup (D_3 \setminus (Q_R^1 \cup Q_R^2))$
$= ((D_1 \cap D_2 \cap D_3) \cup ((D_1 \cap D_2) \setminus Q_R^3) \cup ((D_1 \cap D_3) \setminus Q_R^2)) \cup (D_1 \setminus (Q_R^2 \cup Q_R^3)) \cup (((D_2 \cap D_3) \setminus Q_R^1) \cup (D_2 \setminus (Q_R^1 \cup Q_R^3)) \cup (D_3 \setminus (Q_R^1 \cup Q_R^2)))$

1. $(D_1 \cap D_2 \cap D_3) \cup ((D_1 \cap D_2) \setminus Q_R^3) \cup ((D_1 \cap D_3) \setminus Q_R^2) = D_1 \cap (D_2 \triangle D_3)$
2. $D_1 \setminus (Q_R^2 \cup Q_R^3)$
3. $((D_2 \cap D_3) \setminus Q_R^1) \cup (D_2 \setminus (Q_R^1 \cup Q_R^3)) \cup (D_3 \setminus (Q_R^1 \cup Q_R^2)) = (D_2 \triangle D_3) \setminus Q_R^1$

$= (D_1 \cap (D_2 \triangle D_3)) \cup (D_1 \setminus (Q_R^2 \cup Q_R^3)) \cup ((D_2 \triangle D_3) \setminus Q_R^1)$
$= (D_1 \triangle (D_2 \triangle D_3)$

□

Le lemme 4.1 et la définition 4.5 de $D_1 \triangle D_2$ nous permettent d'énoncer un théorème à propos du raffinement affaibli du système complet. Ce théorème nous permettra par la suite de conclure à propos du raffinement strict d'un système, en étudiant le raffinement affaibli de ses sous-systèmes.

Théorème 4.1 (Raffinement affaibli par décomposition)
Soient $SA^1 \cup SA^2$ et $SR^1 \cup SR^2$ deux ST2E. Alors

$$\frac{\mathcal{T}_{SA^1}(SR^1) \sqsubseteq^{D_1}_{\rho_w} SA^1, \quad \mathcal{T}_{SA^2}(SR^2) \sqsubseteq^{D_2}_{\rho_w} SA^2}{\mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2) \sqsubseteq^{D_1 \triangle D_2}_{\rho_w} SA^1 \cup SA^2}$$

PREUVE. Soit $M \stackrel{\text{def}}{=} \{(q_R, q_A) \mid q_R \in Q^1_R \cup Q^2_R \wedge q_A \in Q^1_A \cup Q^2_A\}$. Montrons que M vérifie toutes les conditions de correction de la définition 4.3 de la relation ρ_w.

1. *Simulation γ :* supposons que $(q_R, e, q'_R) \in T_R$ et $e \in E_A$. Nous devons prouver qu'il existe $q_A \in Q_A$ et $q'_A \in Q_A$ tels que $(q_A, e, q'_A) \in T_A$, $q_R \gamma q_A$ et $q'_R \gamma q'_A$. Par définition, $E_A = E^1_A \cup E^2_A$. La preuve dépend de e.
   - $e \in E^1_A$. Nous avons $(q_R, e, q'_R) \in T^1_R$. Puisque $\mathcal{T}(SR^1) \sqsubseteq^{D_1}_{\rho_w} SA^1$, il existe $q_A \in Q^1_A$ tel que $q_R \gamma q_A$ et il existe $q'_A \in Q^1_A$ tel que $(q_A, e, q'_A) \in T^1_A$ et $q'_R \gamma q'_A$. q_A et q'_A appartiennent à $Q_A = Q^1_A \cup Q^2_A$.
   - $e \in E^2_A$. Nous avons $(q_R, e, q'_R) \in T^2_R$. Puisque $\mathcal{T}(SR^2) \sqsubseteq^{D_2}_{\rho_w} SA^2$, il existe $q_A \in Q^2_A$ tel que $q_R \gamma q_A$ et il existe $q'_A \in Q^2_A$ tel que $(q_A, e, q'_A) \in T^2_A$ et $q'_R \gamma q'_A$. q_A et q'_A appartiennent à $Q_A = Q^1_A \cup Q^2_A$.
2. *τ-simulation ϱ :* supposons que $(q_R, \tau, q'_R) \in T_R$. Nous devons prouver qu'il existe $q_A \in Q_A$ tel que $q_R \varrho q_A$ et $q'_R \varrho q_A$. Par définition, $E_A = E^1_A \cup E^2_A$ et $E_R = E^1_R \cup E^2_R$. La preuve dépend de l'étiquette $e \in E_R \setminus E_A$ masquée par τ dans $(q_R, e \backslash \tau, q'_R)$.
   - $e \in E^1_R \setminus E^1_A$. Nous avons $(q_R, e \backslash \tau, q'_R) \in T^1_R$. Puisque $\mathcal{T}(SR^1) \sqsubseteq^{D_1}_{\rho_w} SA^1$, il existe $q_A \in Q^1_A$ tel que $q_R \varrho q_A$ et $q'_R \varrho q_A$. q_A appartient à $Q_A = Q^1_A \cup Q^2_A$.
   - $e \in E^2_R \setminus E^2_A$. Nous avons $(q_R, e \backslash \tau, q'_R) \in T^2_R$. Puisque $\mathcal{T}(SR^2) \sqsubseteq^{D_2}_{\rho_w} SA^2$, il existe $q_A \in Q^2_A$ tel que $q_R \varrho q_A$ et $q'_R \varrho q_A$. q_A appartient à $Q_A = Q^1_A \cup Q^2_A$.
3. *τ-simulation affaiblie non bloquante ρ_w :* supposons que $q_R \nrightarrow \in T_R$. Nous devons prouver qu'il existe $q_A \in Q_A$, tel que $q_R \rho_w q_A$ et que $q_A \nrightarrow$ ou que $q_R \in D_1 \triangle D_2$. Par définition $Q_R = Q^1_R \cup Q^2_R = (Q^1_R \cap Q^2_R) \cup (Q^1_R \setminus Q^2_R) \cup (Q^1_R \setminus Q^2_R)$. La preuve dépend de q_R.
   - $q_R \in Q^1_R \setminus Q^2_R$. Puisque $\mathcal{T}(SR^1) \sqsubseteq^{D_1}_{\rho_w} SA^1$, soit, il existe $q_A \in Q^1_A$ tel que $q_A \nrightarrow$ et $q_R \rho_w q_A$. q_A appartient à Q_A ; soit, $q_R \in D_1$ et $q_R \rho_w q_A$. Comme $q_R \in Q^1_R \setminus Q^2_R$ et $q_R \in D_1$, $q_R \in D_1 \setminus Q^2_R$, et donc, q_R appartient à $D_1 \triangle D_2$.
   - $q_R \in Q^2_R \setminus Q^1_R$. Puisque $\mathcal{T}(SR^2) \sqsubseteq^{D_2}_{\rho_w} SA^2$, soit, il existe $q_A \in Q^2_A$ tel que $q_A \nrightarrow$ et $q_R \rho_w q_A$. q_A appartient à Q_A ; soit, $q_R \in D_2$ et $q_R \rho_w q_A$. Comme $q_R \in Q^2_R \setminus Q^1_R$ et $q_R \in D_2$, $q_R \in D_2 \setminus Q^1_R$, et donc, q_R appartient à $D_1 \triangle D_2$.
   - $q_R \in Q^1_R \cap Q^2_R$.
     Puisque $\mathcal{T}(SR^1) \sqsubseteq^{D_1}_{\rho_w} SA^1$, soit (1) il existe $q_A \in Q^1_A$ tel que $q_A \nrightarrow$ et $q_R \rho_w q_A$, soit (2) il existe $q_A \in Q^1_A$ tel que $q_R \rho_w q_A$ et $q_R \in D_1$.
     Puisque $\mathcal{T}(SR^2) \sqsubseteq^{D_2}_{\rho_w} SA^2$, soit (3) il existe $q_A \in Q^2_A$ tel que $q_A \nrightarrow$ et $q_R \rho_w q_A$, soit (4) il existe $q_A \in Q^2_A$ tel que $q_R \rho_w q_A$ et $q_R \in D_2$.
     On a quatre combinaisons possibles :

- (1) et (3). Il existe $q_A \in Q_A^1 \cup Q_A^2$ tel que $q_A \nrightarrow$.
- (1) et (4). Il existe $q_A \in Q_A^1$ tel que $q_A \nrightarrow$.
- (2) et (3). Il existe $q_A \in Q_A^2$ tel que $q_A \nrightarrow$.
- (2) et (4). $q_R \in D_1$ et $q_R \in D_2$, d'où $q_R \in D_1 \cap D_2$, et donc, q_R appartient à $D_1 \triangle D_2$.

□

4.2.7 Raffinement strict par décomposition

Le théorème 4.1 précédent, à propos du raffinement affaibli, nous permet d'exprimer un autre théorème – comme une sorte de "conséquence" du théorème précédent – permettant de conclure à propos du raffinement strict d'un système, en analysant les raffinements affaiblis des sous-systèmes issus de sa décomposition, et en étudiant de manière séparée l'absence de τ-cycles dans le système complet raffiné.

Théorème 4.2 (Raffinement par décomposition)
Soient $SA^1 \cup SA^2$ et $SR^1 \cup SR^2$ deux ST2E. Alors

$$\frac{\begin{array}{l}\mathcal{T}_{SA^1}(SR^1) \sqsubseteq_{\rho_w}^{D_1} SA^1,\ \mathcal{T}_{SA^2}(SR^2) \sqsubseteq_{\rho_w}^{D_2} SA^2, \\ D_1 \triangle D_2 = \varnothing,\ \neg\ div^\tau(\mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2))\end{array}}{\mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2) \sqsubseteq_\eta SA^1 \cup SA^2}$$

PREUVE. Le théorème 4.1 donne le raffinement affaibli $\mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2) \sqsubseteq_{\rho_w}^{D_1 \triangle D_2} SA^1 \cup SA^2$. Grâce à $D_1 \triangle D_2 = \varnothing$ et $\neg\ div^\tau(\mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2))$, on peut conclure en utilisant le lemme 4.2.

□

Le théorème précédent se généralise pour une décomposition en n sous-systèmes grâce aux propriétés d'associativité de la décomposition et de la réduction des états de blocage.

Théorème 4.3 (Raffinement et décomposition généralisée)
Soient $SA^1 \cup \cdots \cup SA^n$ et $SR^1 \cup \cdots \cup SR^n$ deux ST2E.

$$\frac{\bigwedge_{i=1}^{n} \left(\mathcal{T}_{SA^i}(SR^i) \sqsubseteq_{\rho_w}^{D_i} SA^i\right),\ \bigwedge_{i=1}^{n} D_i = \varnothing,\ \neg\ div^\tau\left(\bigcup_{i=1}^{n} \mathcal{T}_{SA^i}(SR^i)\right)}{\bigcup_{i=1}^{n} \mathcal{T}_{SA^i}(SR^i) \sqsubseteq_\eta \bigcup_{i=1}^{n} SA^i}$$

PREUVE. Immédiate grâce aux propriétés d'associativité 4.1 et 4.2, au théorème à propos du raffinement affaibli 4.1 et au lemme 4.2.

□

Puisque l'opération de décomposition définie dans ce chapitre est une opération générique, le théorème 4.2 à propos du raffinement par décomposition donne un cadre très général

pour vérifier de manière compositionelle la correction du raffinement. Nous étudierons dans le chapitre 5 une application du théorème 4.2 pour les systèmes à composants.

Remarque. L'étude compositionnelle de la non-τ-divergence n'est pas abordée dans ce chapitre. Notons qu'il est possible d'étudier compositionnellement la présence ou non de τ-cycles dans le système complet, si l'on se place dans un cadre plus précis que celui de la décomposition, par exemple, celui des *systèmes à composants synchronisés*, présentés dans le chapitre suivant (voir précisément la section 5.2.3).

De manière plus générale, étudier l'absence de τ-cycles n'est pas toujours nécessaire. Souvent, le raffinement affaibli suffit puisqu'il garantit la présence d'une relation de τ-simulation préservant toutes les propriétés LTL de sûreté.

> **Exemple.** Dans la section 4.2.3, nous avions indiqué ne pas pouvoir étudier le raffinement du bras mobile robotisé $Robot_R$ en utilisant la décomposition proposée : $Robot_R = Robot_R^1 \cup Robot_R^2 \cup Robot_R^3$. Le théorème 4.3 permet cette étude. Nous pouvons établir successivement la relation de raffinement affaiblie ρ_w pour chacun des sous-systèmes :
>
> 1. $\mathcal{T}(Robot_R^1) \sqsubseteq_{\rho_w}^{D_1} Robot_A^1$ et $D_1 = \{q_e\}$,
> 2. $\mathcal{T}(Robot_R^2) \sqsubseteq_{\rho_w}^{D_2} Robot_A^2$ et $D_2 = \varnothing$, et
> 3. $\mathcal{T}(Robot_R^3) \sqsubseteq_{\rho_w}^{D_3} Robot_A^3$ et $D_3 = \varnothing$.
>
> L'ensemble D_1 n'est pas vide mais on peut facilement montrer que $D_1 \triangle D_2 \triangle D_3 = \varnothing$ et nous pouvons donc conclure que $Robot_R = Robot_R^1 \cup Robot_R^2 \cup Robot_R^3$ est bien un raffinement correct de $Robot_A = Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$.

4.3 Décomposition et vérification de propriétés

La décomposition d'un système en plusieurs sous-systèmes permet de simplifier la vérification de certaines propriétés du système complet. Au lieu de vérifier une propriété sur le système complet, nous la vérifierons sur les sous-systèmes.

De plus, puisque nous avons montré dans la section précédente que la décomposition et le raffinement étaient des démarches conciliables, nous bénéficions de toutes les propriétés de la relation de raffinement η et, particulièrement, de la préservation pour le système complet raffiné des propriétés vérifiées par décomposition au niveau abstrait (voir section 3.4.1). Dans certains cas (lorsque les schémas coïncideront), nous pourrons appliquer la démarche de reformulation pour ré-exprimer les propriétés abstraites au niveau du système raffiné (voir section 3.4.2).

4.3.1 Décomposition et propriétés d'invariance

La décomposition d'un système en plusieurs sous-systèmes permet de vérifier compositionnellement des propriétés d'invariance. En effet, un invariant qui est vérifié par chacun des

sous-systèmes sera forcément vérifié par le système complet formé par l'union de ces sous-systèmes. De même, si un invariant n'est pas vérifié par l'un des sous-systèmes, il ne sera pas non plus vérifié par le système complet.

Proposition 4.1 (Décomposition et Invariant)
Soient $SA^1 \cup SA^2$ un ST2E. Soit $\Box sp_A$ une propriété LTL telle que $sp_A \in SP_{V_A}$. Alors

$$\frac{SA^1 \models \Box sp_A, \quad SA^2 \models \Box sp_A,}{SA^1 \cup SA^2 \models \Box sp_A}$$

PREUVE. (par contradiction) Supposons que $SA^1 \cup SA^2 \not\models \Box sp_A$. Il existe donc un état $q \in Q_A$ tel que $q \not\models sp_A$. Par la définition 4.2, on a $q \in Q_A^1 \cup Q_A^2$. Donc, $SA^1 \not\models \Box sp_A$ si $q \in Q_A^1$ et $SA^2 \not\models \Box sp_A$ si $q \in Q_A^2$, ce qui contredit les hypothèses.

□

Par contraposition, si une propriété d'invariance n'est pas vérifiée par l'un des sous-systèmes, alors elle ne sera pas non plus vérifiée par le système complet.

Décomposition et raffinement sont des démarches conciliables : la préservation des propriétés LTL peut donc s'appliquer au cas particulier des propriétés d'invariance vérifiées par décomposition.

Proposition 4.2 (Décomposition et préservation des invariants)
Soient $SA^1 \cup SA^2$ et $SR^1 \cup SR^2$ deux ST2E. Soit $\Box sp_A$ une propriété LTL telle que $sp_A \in SP_{V_A}$.

$$\textit{Si} \begin{cases} \mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2) \sqsubseteq_\eta SA^1 \cup SA^2 \\ SA^1 \cup SA^2 \models \Box sp_A \end{cases} \quad \textit{alors } SR^1 \cup SR^2 \models_{pre} \Box sp_A$$

PREUVE. Conséquence directe du théorème 3.2 et de la proposition 4.1.

□

Exemple. Considérons la propriété d'invariance (3.2) énoncée section 3.1.2 :

$$\Box((el = mid) \Rightarrow (ar = back \wedge cl = close))$$

Nous pouvons vérifier que cette propriété est bien satisfaite par les trois sous-systèmes $Robot_A^1$, $Robot_A^2$ et $Robot_A^3$ donnés figure 4.2. Comme conséquence de la proposition 4.1, on a que $Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$ satisfait l'invariant (3.2). Comme $Robot_R^1 \cup Robot_R^2 \cup Robot_R^3 \sqsubseteq_\eta Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$, la proposition 4.2 nous donne que $Robot_R^1 \cup Robot_R^2 \cup Robot_R^3$ satisfait par préservation la propriété (3.2).

Les propositions 3.1 et 4.2 permettent de combiner, pour des propriétés d'invariance, les démarches de reformulation et de vérification par décomposition. Soient $\Box sp_A$ et $\Box sp_R$

deux propriétés LTL telles que $sp_A \in SP_{V_A}$ et $sp_R \in SP_{V_R}$.

$$\left.\begin{array}{l} \mathcal{T}_{SA^1}(SR^1) \cup \mathcal{T}_{SA^2}(SR^2) \sqsubseteq_\eta SA^1 \cup SA^2 \\ SA^1 \cup SA^2 \models \Box sp_A \\ sp_A \wedge gp \Rightarrow sp_R \end{array}\right\} \Rightarrow\ SR^1 \cup SR^2 \models \Box sp_R$$

Les propositions précédentes à propos de la préservation et de la reformulation de propriétés d'invariance nécessitent que le raffinement entre le système complet raffiné $SR^1 \cup SR^2$ et le système abstrait $SA^1 \cup SA^2$ soit vérifié, c.à.d. que le raffinement affaibli soit vérifié entre SR^1 et SA^1 et entre SR^2 et SA^2. Lors de la vérification du raffinement affaibli pour chacun des sous-systèmes, on explore, les uns après les autres, tous les états des sous-systèmes SA^1 et SA^2 : on peut facilement vérifier qu'un invariant sp_A est satisfait par chacun de ces états.

4.3.2 Décomposition et invariants dynamiques

Dans cette section, nous montrons que certaines propriétés, appelées *invariants dynamiques* par extension de la terminologie B, peuvent être vérifiées compositionnellement. Un invariant dynamique parle de satisfaction dans un état donné, mais aussi dans les états suivants l'état courant. Une telle propriété s'exprime en LTL par une formule de la forme $\Box(sp \Rightarrow \bigcirc sp')$, avec sp et sp' deux formules d'états.

Nous noterons $Dynamics(S, sp, sp')$, un prédicat exprimant que quelque soit l'état q du système S vérifiant la proposition d'état sp, il existe une transition partant de q et, pour toutes les transitions partant de q et conduisant à un état q', alors q' vérifie sp'.

$$Dynamics(S, sp, sp') \stackrel{\text{def}}{=} \forall q.\ (q \in Q \wedge q \models sp \Rightarrow q \rightarrow \wedge \forall q'.\ (q \rightarrow q' \Rightarrow q' \models sp')$$

Le prédicat $Dynamics(S, sp, sp')$ exprime la même propriété que la formule LTL $\Box(sp \Rightarrow \bigcirc sp')$ sans avoir besoin de considérer la totalité d'une exécution, mais uniquement un état et ses successeurs. On pourra remplacer $Dynamics(S, sp, sp')$ par $\Box(sp \Rightarrow \bigcirc sp')$ dans le cas des systèmes complets.

Proposition 4.3 (Invariant dynamique et décomposition)
Soit $SA = SA^1 \cup SA^2$ un ST2E. Soient $sp \in SP_V$ et $sp' \in SP_V$ deux formules d'états.

$$\textit{Si} \left\{\begin{array}{l} Dynamics(SA^1, sp, sp') \\ Dynamics(SA^2, sp, sp') \end{array}\right. \quad \textit{alors } Dynamics(SA^1 \cup SA^2, sp, sp')$$

PREUVE. Considérons $\neg Dynamics(SA^1 \cup SA^2, sp, sp') = \exists q.\ (q \in Q^1 \cup Q^2 \wedge q \models sp \wedge (q \nrightarrow \vee \exists q'.(q \rightarrow q' \wedge q' \not\models sp')))$
Si $q \in Q^1$, $q \models sp$ et $q \nrightarrow$ ou $q' \not\models sp'$, alors on a une contradiction avec $Dynamics(SA^1, sp, sp')$.
Si $q \in Q^2$, $q \models sp$ et $q \nrightarrow$ ou $q' \not\models sp'$, alors on a une contradiction avec $Dynamics(SA^2, sp, sp')$.
□

Exemple. Considérons le système $Robot_A$ qui se décompose en trois sous-systèmes $Robot_A^1$, $Robot_A^2$ et $Robot_A^3$ (voir figure 4.2). La propriété (3.4) exprimée par la formule LTL

$$\Box((el = down \wedge ar = front \wedge cl = open) \Rightarrow \bigcirc(ar = back))$$

peut être vérifiée par décomposition. On montre que
– $Dynamics(Robot_A^1, ' el = down \wedge ar = front \wedge cl = open', ' ar = back') = true$
– $Dynamics(Robot_A^2, ' el = down \wedge ar = front \wedge cl = open', ' ar = back') = true$
– $Dynamics(Robot_A^3, ' el = down \wedge ar = front \wedge cl = open', ' ar = back' = true$
et on en déduit que la propriété est aussi satisfaite par $Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$ grâce à la proposition 4.3.

4.3.3 Décomposition et propriétés LTL

Dans les deux sections précédentes, nous avons vu qu'il était possible de vérifier, de manière compositionnelle en utilisant les sous-systèmes, certaines propriétés LTL : les invariants $\Box sp$ et les invariants dynamiques $\Box(sp \Rightarrow \bigcirc sp')$. Mais, puisque la décomposition en plusieurs sous-systèmes décompose l'espace de transitions du système complet, il est possible que l'on perde certains comportements (ou partie de comportements) dans les sous-systèmes. Il semble dans ce cas assez délicat de s'assurer si une propriété LTL est satisfaite par les sous-systèmes et d'en déduire sa satisfaction par le système complet.

Nous pouvons par contre montrer que nous avons la propriété inverse, pour les propriétés de sûreté en tout cas. Si $\phi \in LTL_{saf}$ n'est pas satisfaite par l'un des sous-systèmes au moins, alors elle ne sera pas satisfaite par le système complet.

Considérons un ST2E $SA^1 \cup SA^2$. Il est assez facile de montrer que $SA^1 \cup SA^2$ simule SA^1 (au sens de $SA^1 \preceq_\gamma SA^1 \cup SA^2$). La relation de collage entre SA^1 et $SA^1 \cup SA^2$ sera exprimée comme l'égalité des états.

Lemme 4.3 (**$SA^1 \cup SA^2$ simule SA^1**)
Soit $SA^1 \cup SA^2$ un ST2E. Alors

$$SA^1 \preceq_\gamma SA^1 \cup SA^2$$

PREUVE. La preuve est immédiate puisque quelque soit la transition de SA^1 considérée, alors c'est aussi une transition de $SA^1 \cup SA^2$ en vertu de la définition 4.1 d'un sous-système.

□

Exemple. Considérons toujours l'exemple du bras mobile robotisé. Le système complet $Robot_A$ est présenté dans les sections 1.1.4 et 2.2.4. La figure 4.2, section 4.1, illustre une décomposition de $Robot_A$ telle que $Robot_A = Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$. On peut par exemple montrer que $Robot_A$ simule $Robot_A^3$: voir figure 4.10.

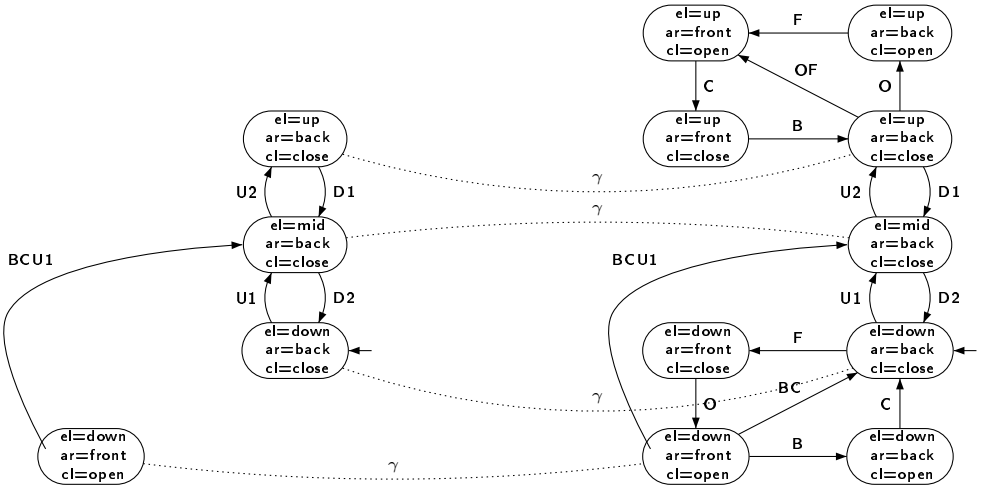


FIG. 4.10: $Robot_A^3$ simulé par $Robot_A$

Si $SA^1 \cup SA^2$ simule SA^1, alors on bénéficie de la préservation des propriétés de sûreté et on peut en déduire que pour toute propriété $\phi \in LTL_{saf}$ qui ne sera pas satisfaite par SA^1 ou par SA^2, alors elle ne sera pas non plus satisfaite par $SA^1 \cup SA^2$.

Proposition 4.4 (Préservation de la non-satisfaction par décomposition)
Soit $SA^1 \cup SA^2$ un ST2E. Soit $\phi \in LTL_{saf}$.

$$\textit{Si } SA^1 \not\models \phi \textit{ alors } SA^1 \cup SA^2 \not\models \phi$$

PREUVE. Grâce au lemme 4.3, on a $SA^1 \preceq_\gamma SA^1 \cup SA^2$. Considérons $\phi \in LTL_{saf}$ telle que $SA^1 \cup SA^2 \models \phi$. Alors on a $SA^1 \models \phi$, c.à.d. $SA^1 \cup SA^2 \models \phi \Rightarrow SA^1 \models \phi$.
Prenons la contraposée : $\neg(SA^1 \models \phi) \Rightarrow \neg(SA^1 \cup SA^2 \models \phi)$, c.à.d. $SA^1 \not\models \phi \Rightarrow SA^1 \cup SA^2 \not\models \phi$
□

D'un point de vue pratique, il est plus facile de vérifier qu'une propriété n'est pas satisfaite sur un des sous-systèmes, plutôt que sur le système pris dans sa totalité.

> **Exemple.** Considérons une propriété qui indique que si le dispositif de transport est en position basse alors il passera en position haute, plus tard :
>
> $$\Box((el = down) \Rightarrow \Diamond(el = up)) \tag{4.1}$$
>
> On peut montrer que la propriété (4.1) n'est pas satisfaite par le sous-système $Robot_A^3$. Par conséquent, elle n'est donc pas non plus satisfaite par le système complet $Robot_A^1 \cup Robot_A^2 \cup Robot_A^3$.

4.3.4 Décomposition et analyse d'atteignabilité

D'une manière générale, l'analyse d'atteignabilité consiste à étudier si une situation peut être atteinte, c.a.d. si un état (ou un ensemble d'états) peu(ven)t être atteint(s) via la

relation de transitions du système considéré. Nous noterons $reachable(q_c, S)$ un prédicat indiquant que l'état cible q_c est atteignable dans S (voir section 1.1.1).

Dans certains cas, la décomposition du système en plusieurs sous-systèmes pourra fournir une méthode efficace d'analyse d'atteignabilité. Considérons un système $SA = SA^1 \cup SA^2 \cup \cdots \cup SA^n$ qui se décompose. L'ensemble des états et la relation de transition d'un sous-système SA^i sont moins importants que ceux du système complet (voir définition 4.1) et si un état est atteignable dans l'un des sous-systèmes, il le sera *a fortiori* dans le système complet. Nous pouvons réduire de cette manière le nombre d'états (et de chemins) du système à explorer, et, par conséquent, nous repoussons d'autant le problème d'explosion combinatoire.

Proposition 4.5 (Atteignabilité compositionnelle)
Soit $SA = SA^1 \cup SA^2 \cup \cdots \cup SA^n$ un ST2E. Soit q_c un état.

$$\textit{S'il existe } i \textit{ tel que } 1 \leq i \leq n \textit{ et } reachable(q_c, SA^i) \textit{ alors } reachable(q_c, SA)$$

PREUVE. Evidente grâce à la définition 4.1.

□

En pratique, l'atteignabilité d'un état cible q_c nécessitera de passer par plusieurs sous-systèmes. Plaçons nous dans le cas d'une analyse d'atteignabilité par chaînage arrière, c.à.d. qu'il s'agit depuis l'état cible de réussir à atteindre un des états initiaux du système, en "remontant" les transitions. Dans ce cas, nous pourrons combiner plusieurs analyses d'atteignabilité successives de la manière suivante, en affinant la recherche.

1. Pendant l'exploration du graphe d'atteignabilité du premier sous-système, un ensemble d'états cibles "potentiels" Q_P est construit à partir des états de blocage rencontrés.
2. Si nous n'arrivons pas à atteindre un état initial lors de cette exploration, alors nous choisissons un autre sous-système S^i, et – souvent – un nouvel état cible q_i parmi Q_P. Nous pouvons faire ce choix car q_c est déjà connu comme état atteignable depuis ce nouvel état cible.
3. Nous recommençons l'analyse d'atteignabilité avec le nouveau sous-système et le nouvel état cible.

Nous donnons l'algorithme 4.1 pour illustrer de manière plus formelle cette analyse d'atteignabilité compositionnelle.

Algorithme 4.1 (Analyse d'atteignabilité compositionnelle)
Input
$AS_1^n = \{S^1, \ldots, S^n\}$, *(* Ensemble des sous-systèmes *)*
$Q_0 = Q_0^1 \cup \cdots \cup Q_0^n$, *(* Espace d'états initial *)*
$q_c \in Q^1 \cup \cdots \cup Q^n$ *(* Etat cible à atteindre *)*
Result
reach : boolean *(* vrai si q_c est atteint *)*

Variables
$q_i \in Q^1 \cup \cdots \cup Q^n$, (* *Etat courant* *)
$S^i \in AS_1^n$, (* *Sous-système courant* *)
$Q_P \subseteq Q^1 \cup \cdots \cup Q^n$, (* *Ensemble d'états à choisir* *)
$Q_D \subseteq Q^1 \cup \cdots \cup Q^n$, (* *Ensemble d'états bloquants* *)
$Q_{suc} \subseteq Q^1 \cup \cdots \cup Q^n$, (* *Ensemble des sucesseurs* *)
$Q_{pre} \subseteq Q^1 \cup \cdots \cup Q^n$, (* *Ensemble des prédecesseurs* *)
$tested \subseteq (Q^1 \cup \cdots \cup Q^n) \times \{S^1, \ldots, S^n\}$ (* *Combinaisons (état, sous-système) déja testées* *)
end, possible, deadlock : boolean
Begin
$Q_P := \{q_c\}$ (* *Initialisation* *)
$tested := \{\}$
$end :=$ false
$reach :=$ false
while (*end* = false) ***do***
Choice(Q_P,AS_1^n, *tested, possible*, q_i, S^i) (* *Choix d'une paire* (q_i, S^i) *)
if (*possible* = false) ***do***
$end :=$ true (* *Plus de choix possible* *)
else
$tested := tested \cup \{(q_i,S^i)\}$ (* *nouvelle paire testée* *)
$deadlock :=$ false (* *Ré-initialisation* *)
$Q_{suc} := \{q_i\}$
$Q_{pre} := \{\}$
while (*deadlock* = false) ***do***
Predecessors(Q_{suc},S^i,Q_{pre},Q_D) (* *Calcul des prédécesseurs* *)
$Q_P := Q_P \cup Q_D$ (* *Nouveaux états possibles* *)
if ($Q_{pre} = \{\}$) ***do*** (* *Aucun prédécesseur* *)
$deadlock :=$ true
else
if ($Q_0 \cap Q_{pre}$!= {}) ***do*** (* *Etat cible atteint* *)
$deadlock :=$ true
$end :=$ true
$reach :=$ true
else
$Q_{suc} := Q_{pre}$ (* *On continue* *)
endif
endif
endwhile
endif
endwhile
End

L'algorithme 4.1 donné précédemment utilise deux procédures :
- *Predecessors(***input** Q_{suc},S^i, **result** Q_{pre},Q_D*)* calcule l'ensemble Q_{pre} des prédécesseurs de Q_{suc} dans S^i. De plus, les états de Q_{suc} sans prédécesseurs sont placés dans Q_D.
- *Choice(***input** Q_P,AS_1^n, *tested*,**result** *possible*, q_i, S^i*)* choisit une paire (q_i, S^i) non encore testée pour continuer l'analyse d'atteignabilité. Si l'on ne peut plus choisir de paires (q_i, S^i), *possible* = false et l'analyse s'arrête.

Pour être efficace, le choix du nouveau sous-système et du nouvel état cible doit être guidé par des heuristiques. L'analyse de dépendances fortes [LNAH+01] entre composants pourrait être utilisée dans ce but (voir aussi [CGK97, CK99]).

Nous rappelons aussi que la LTL ne permet pas d'exprimer l'atteignabilité. Dans la mesure où la LTL est quantifiée implicitement sur toutes les exécutions, elle ne peut exprimer l'atteignabilité que négativement : q_c n'est pas atteignable s'exprime alors $\Box\neg q_c$. Préservation et analyse d'atteignabilité compositionnelle seront conciliables uniquement si l'analyse d'atteignabilité montre que l'état cible q_c n'est pas atteignable.

4.4 Conclusion

Dans ce chapitre, nous avons présenté une méthode générale pour la vérification par décomposition du raffinement d'un système et de ses propriétés.

Nous avons défini la notion de *décomposition* d'un ST2E en plusieurs sous-systèmes comme une "union" très générale de plusieurs sous-systèmes permettant l'étude séparée de chacun des sous-systèmes. L'opérateur de décomposition est un opérateur générique qui autorise des décompositions plus fines, comme par exemple les systèmes de transitions séparables [BQ96]. Nous verrons dans le chapitre 5 une manière systématique pour obtenir une décomposition d'un produit synchronisé à partir de ses composants.

Nous avons ensuite étudié la compositionnalité de la relation de raffinement η. Des exemples mettent en évidence que l'étude du raffinement des sous-systèmes ne permet que dans de rares cas, de conclure à propos du raffinement du système complet :
- La décomposition fait apparaître de "faux" nouveaux états de blocage dans les sous-systèmes raffinés qui font échouer la vérification du raffinement des sous-systèmes.
- La décomposition en sous-systèmes peut masquer dans les sous-systèmes l'entrelacement de nouvelles transitions dans le système complet, empêchant alors de conclure de manière systématique à propos du raffinement du système complet.

Nous définissons alors une relation de raffinement *affaiblie* pour prendre en compte le problème des nouveaux blocages. Une procédure de réduction nous permet de vérifier s'il s'agit ou non de "faux" nouveaux blocages introduits par la décomposition. Nous montrons qu'il suffit d'étudier une relation de raffinement *affaiblie* entre chacun des sous-systèmes issus de la décomposition pour pouvoir conclure à propos du raffinement affaibli du système

complet.

Néanmoins, calculer $D_1 \triangle D_2$ est coûteux puisqu'il nécessite de connaître simultanément les deux ensembles de nouveaux blocages D_1 et D_2 issus des raffinements affaiblis des sous-systèmes. Notons aussi que l'étude compositionnelle de la non-τ-divergence n'est pas abordée dans ce chapitre. Néanmoins, étudier l'absence de τ-cycles n'est pas forcément nécessaire. Souvent, le raffinement affaibli suffit : il garantit une relation de τ-simulation préservant toutes les propriétés LTL de sûreté.

Décomposition et raffinement sont rendus compatibles, et, puisque l'opérateur de décomposition présenté dans ce chapitre est un opérateur générique, le théorème 4.2 à propos du raffinement par décomposition définit un cadre très général pour vérifier de manière compositionnelle la correction du raffinement. Nous étudierons dans le chapitre 5 une application du théorème 4.2 aux systèmes à composants synchronisés. Notons que dans le cadre des systèmes à composants synchronisés, la présence ou non de τ-cycles dans le système complet pourra être vérifiée compositionnellement, directement sur les composants du système.

La décomposition d'un système en plusieurs sous-systèmes peut simplifier la vérification de certaines propriétés du système complet. Au lieu de vérifier une propriété sur le système complet, nous la vérifierons sur les sous-systèmes :

- Une propriété d'invariance qui est vérifiée par chacun des sous-systèmes sera forcément vérifiée par le système complet formé par l'union de ces sous-systèmes.
- Une propriété que nous appelons *invariant dynamique*, notée $Dynamics(S, sp, sp')$ et qui correspond en LTL à une formule de la forme $\Box(sp \Rightarrow \bigcirc sp')$ peut aussi être vérifiée sur les sous-systèmes. Sa satisfaction sur chaque sous-système assure sa satisfaction sur le système complet.
- D'une manière plus générale, si l'on montre la non-satisfaction, pour au moins un des sous-systèmes, d'une propriété LTL ϕ exprimant une propriété de sûreté, alors elle ne sera pas satisfaite par le système complet.
- Nous faisons aussi le constat suivant : si un état est atteignable dans l'un des sous-systèmes, il le sera (*a fortiori* dans le système complet. Cela nous permet de donner un algorithme d'analyse d'atteignabilité compositionnelle utilisant les sous-systèmes.

Puisque le raffinement et la décomposition sont compatibles, nous bénéficions toujours de la préservation pour le système complet raffiné des propriétés vérifiées au niveau abstrait par les sous-systèmes.

Chapitre 5

Raffinement des systèmes à composants synchronisés

Ce chapitre est consacré à l'étude compositionnelle du raffinement des systèmes à composants. Nous rappelons que malgré certaines restrictions le raffinement par décomposition permet d'étudier compositionnellement le raffinement d'un système complet en étudiant simplement le raffinement *affaibli* des sous-systèmes issus de la décomposition du système complet. Néanmoins, plusieurs limitations apparaissent :

- nous n'indiquons pas comment obtenir la décomposition des systèmes abstrait et raffiné,
- il est nécessaire de connaître tous les sous-systèmes simultanément pour *réduire* les ensembles de "faux" blocages,
- le problème de la non τ-divergence n'a pas été abordé.

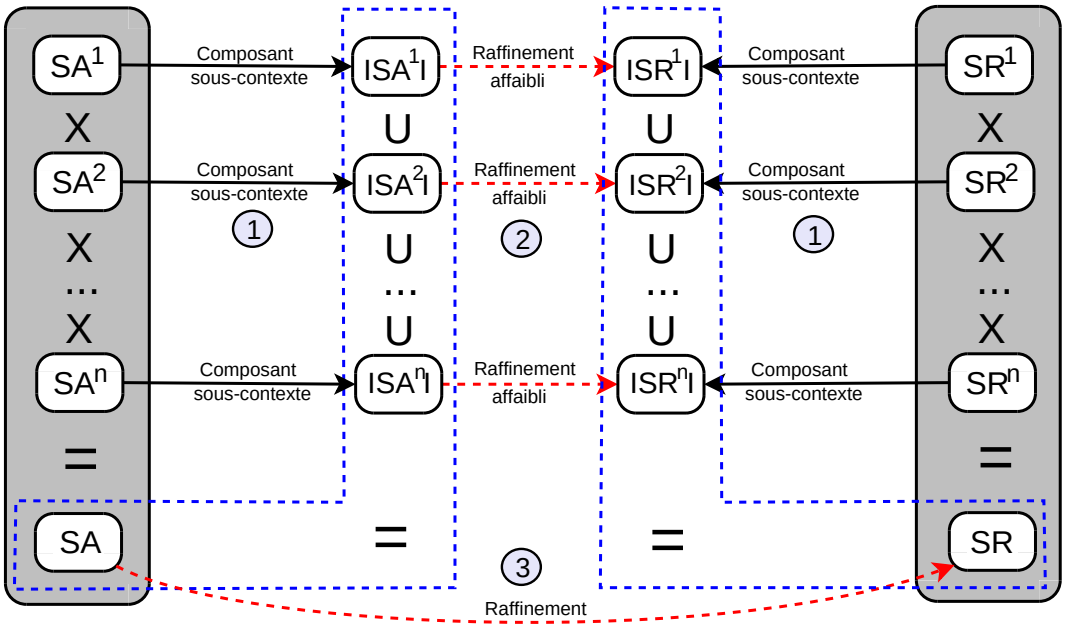


FIG. 5.1: Raffinement compositionnel d'un système à 3 composants

Dans ce chapitre, nous allons nous placer dans le cadre des *systèmes à composants synchronisés* (voir figure 5.1). Ce cadre, plus précis, nous permettra de repousser les limitations du raffinement par décomposition, rappelées ci-dessus.

Nous définissons dans la section 5.1 la notion de *système à composants* synchronisés, comme une extension du produit synchronisé classique d'Arnold et Nivat : nous donnons les composants S^1, S^2, ..., S^n du système sous la forme de ST2Es indépendants les uns des autres et nous décrivons les interactions entre les composants par un ensemble de synchronisations *contraintes* Syn^c. Nous établissons ensuite un lien avec l'opérateur de décomposition du chapitre précédent. Ce lien repose avant tout sur la notion de composant *sous-contexte* : un composant sous-contexte $[S^i]$ est un composant qui tient compte de son environnement, i.e. des autres composants S^1, S^2, ..., S^n et de l'ensemble de synchronisations Syn^c. Nous montrons que les composants sous-contexte forment une *décomposition* du système à composants, obtenue automatiquement, sans avoir besoin de considérer le système à composants synchronisés dans son ensemble, puis de le décomposer.

Dans la section 5.2, nous montrons que l'étude compositionnelle du raffinement d'un système à composants synchronisés se ramène à étudier un raffinement par décomposition, entre autres grâce aux composants sous-contexte. Nous réutilisons en les adaptant, les résultats du chapitre précédent à propos du raffinement affaibli des sous-systèmes. Nous montrons aussi que les limitations du raffinement par décomposition n'ont plus lieu d'être dans le cadre des systèmes à composants : la non τ-divergence est assurée de manière compositionnelle et la réduction des ensembles de "faux" blocages se ramène à étudier uniquement l'ensemble raffiné de synchronisations.

Finalement, nous étudions, dans la section 5.3, la vérification compositionnelle de certaines propriétés LTL, dans le cadre des systèmes à composants synchronisés. Des propriétés locales à un composant – invariant local, et plus généralement, toutes les propriétés LTL exprimant la sûreté – peuvent être vérifiées uniquement sur le composant, puis automatiquement préservées pour le système à composants. Puisque, grâce aux composants sous-contexte, nous obtenons de manière automatique une décomposition du système à composants, nous bénéficions des résultats à propos de la vérification par décomposition de certaines propriétés globales : invariant, invariant dynamique, (in)atteignabilité entre autre. Nous bénéficions toujours de la préservation des propriétés LTL par le raffinement. Il semble alors intéressant d'étudier quand le raffinement compositionnel d'un système à composants et la vérification compositionnelle de certaines propriétés du système peuvent être combinés.

5.1 Systèmes à composants synchronisés

5.1.1 Produit synchronisé contraint

La notion de *produit synchronisé* défini par Arnold et Nivat – présenté dans la section 1.2.2 – ne permet pas d'exprimer tous les comportements que l'on peut envisager.

- Grâce à l'ensemble de synchronisations, il est possible de préciser si une transition e_1 d'un composant S^1 a lieu simultanément (ou pas) à une transition d'étiquette e_2 d'un autre composant S^2.
- En revanche, on ne peut pas indiquer que, dans un cas précis, la transition d'étiquette e_1 a lieu simultanément à e_2, et que, dans un autre cas, elle a lieu indépendamment.

Nous identifierons les différents cas possibles en fonction des variables des composants en utilisant des formules d'états : nous parlerons alors de conditions d'activation. Nous étendons donc la définition 1.9 en définissant un ensemble de synchronisations contraintes Syn^c. Cet ensemble contient les tuples d'étiquettes des transitions que l'on souhaite synchroniser avec des conditions contraignant l'activation de ces transitions.

Définition 5.1 (Ensemble de synchronisations contraintes)
Soient $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ *et* $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ *deux ST2E. Un ensemble de synchronisations contraintes* Syn^c *est un sous-ensemble de* $\{(e_1, e_2)$ **when** $sp \mid (e_1, e_2) \in E^1 \otimes E^2 \wedge sp \in SP_{V_1 \cup V_2}\}$.

Remarque. Nous rappelons que $E^1 \otimes E^2 \stackrel{\text{def}}{=} ((E^1 \cup \{-\}) \times (E^2 \cup \{-\})) \setminus \{(-,-)\}$, comme précisé dans la section 1.2.

Nous pouvons étendre la définition 1.10 afin de définir le *produit synchronisé contraint* de deux composants S^1 et S^2 sous un ensemble de synchronisations contraintes Syn^c.

Définition 5.2 (Produit synchronisé contraint)
Soient $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ et $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ deux ST2E. Soit Syn^c un ensemble de synchronisations contraintes. Le produit synchronisé contraint de S^1 et S^2, noté $S^1 \times_{Syn^c} S^2$ est défini par le six-uplet (Q, Q_0, E, T, V, l) :
- $Q \subseteq Q^1 \times Q^2$,
- $Q_0 \subseteq Q_0^1 \times Q_0^2$,
- $E = \{(e_1, e_2) \mid (e_1, e_2)$ **when** $sp \in Syn^c\}$,
- $T \subseteq Q \times E \times Q$ *est définie ainsi :*
  - $(q_1, q_2) \xrightarrow{(e_1,-)} (q_1', q_2) \in T$ *si* $q_1 \xrightarrow{e_1} q_1' \in T^1$, $(e_1, -)$ **when** $sp \in Syn^c$ *et* $(q_1, q_2) \models sp$
  - $(q_1, q_2) \xrightarrow{(-,e_2)} (q_1, q_2') \in T$ *si* $q_2 \xrightarrow{e_2} q_2' \in T^2$, $(-, e_2)$ **when** $sp \in Syn^c$ *et* $(q_1, q_2) \models sp$
  - $(q_1, q_2) \xrightarrow{(e_1,e_2)} (q_1', q_2') \in T$ *si* $q_1 \xrightarrow{e_1} q_1' \in T^1$, $q_2 \xrightarrow{e_2} q_2' \in T^2$, (e_1, e_2) **when** $sp \in Syn^c$, *et* $(q_1, q_2) \models sp$
- $V = V^1 \cup V^2$,
- $l((q_1, q_2)) = l^1(q_1) \cup l^2(q_2)$.

Il ne suffit plus que la transition synchronisée figure dans l'ensemble des transitions autorisées pour qu'elle apparaisse dans le produit, il faut maintenant qu'elle soit "activable" depuis l'état source de la transition, c.à.d. que la condition d'activation soit satisfaite par l'état source.

La définition du produit synchronisé permet de définir une notion connexe, celle de *système à composants.*

Définition 5.3 (Système à composants (synchronisés))
Soient $S^1 = (Q^1, Q_0^1, E^1, T^1, V^1, l^1)$ et $S^2 = (Q^2, Q_0^2, E^2, T^2, V^2, l^2)$ deux ST2E. Soit Syn^c un ensemble de synchronisations contraintes. Un système à composants – dit synchronisés – S est un tuple de composants muni d'un ensemble de synchronisations contraintes que l'on notera $S = (S^1, S^2, Syn^c)$. La sémantique de S est la même que celle du produit synchronisé contraint $S^1 \times_{Syn^c} S^2$.

Le produit synchronisé contraint donné par la définition 5.2 généralise le produit synchronisé classique d'Arnold et Nivat puisqu'il permet d'exprimer – au moins – autant de comportements que le produit synchronisé classique : considérons un ensemble de synchronisations contraintes dans lequel toutes les conditions d'activations sont égales à *true*. Dans ce cas, le produit synchronisé contraint utilisant cet ensemble est identique au produit synchronisé classique. La figure 5.2 montre une hiérarchie montrant l'expressivité des différents produits définis dans la section 1.2.2. Le produit synchronisé contraint est le plus expressif de tous : on peut aussi bien décrire des produits synchrones et asynchrones ou

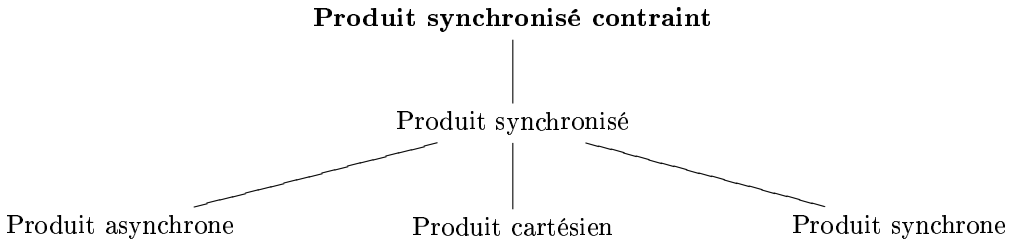


FIG. 5.2: Hiérarchie des différents produits de ST2E

même le produit cartésien grâce au produit synchronisé contraint.

Remarque. La définition 5.2 se généralise de façon évidente au produit synchronisé contraint de n composants. Nous noterons $\Pi^{i=1...n}_{Syn^c}\ S^i$, le produit synchronisé contraint de n composants et $(S^1, S^2, \ldots, S^n, Syn^c)$ le système à composants correspondant. Dans la suite de ce chapitre, nous ne considérons, pour établir définitions et théorèmes, que des systèmes à deux composants synchronisés.

Nous définissons maintenant un opérateur de projection qui, à partir d'un chemin quelconque d'un système à composants, se ramène à un chemin dans l'un des composants.

Définition 5.4 (Projection d'un chemin d'un système à composants)
Soit $\sigma_S = (q_1^0, q_2^0) \xrightarrow{(e_1^1, e_2^1)} (q_1^1, q_2^1) \xrightarrow{(e_1^2, e_2^2)} \ldots \xrightarrow{(e_1^i, e_2^i)} (q_1^i, q_2^i) \xrightarrow{(e_1^{i+1}, e_2^{i+1})} \ldots$ *un chemin de* S *avec* $\forall i \geq 0, \forall j \in \{1,2\}.\ (e_j^i \in E^j \cup \{-\})$. *La projection de* σ_S *sur* S^1, *notée* $proj(\sigma_S, S^1)$ *est définie ainsi :*

$$proj(\sigma_S, S^1) = q_1^0 \xrightarrow{e_1^1} q_1^1 \xrightarrow{e_1^2} \ldots \xrightarrow{e_1^i} q_1^i \xrightarrow{e_1^{i+1}} \ldots \quad \text{où } \forall i \geq 0.\ (e_1^i \in E^1 \cup \{-\})$$

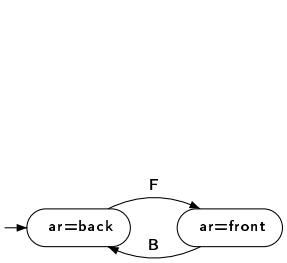
Remarque. Nous pouvons supprimer de $proj(\sigma_S, S^1)$ les transitions de la forme $q_1^i \xrightarrow{'-'} q_1^{i+1}$ sans perdre d'information[1], puisque $q_1^i = q_1^{i+1}$ dans ce cas.

5.1.2 Exemple d'un système à composants : le bras mobile robotisé

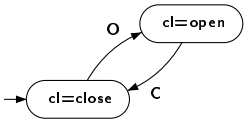
Nous nous intéressons toujours à modéliser une partie d'un dispositif industriel, et plus précisément, la partie concernant un bras mobile robotisé, chargé de déplacer des pièces. Les contraintes de spécification sont les mêmes que celles indiquées dans la section 1.1.4.

Nous voyons maintenant que le bras robotisé est construit à partir de trois composants simples et nous le modélisons par un système à composants. Les composants simples à partir desquels est élaboré le bras robotisé sont les suivants :

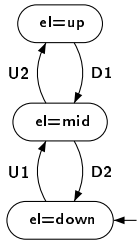
[1] Les transitions étiquetées $'-'$ jouent ici le même rôle – d'un point de vue observationnel – que les ϵ-transitions utilisées par Milner [Mil89] pour caractériser l'action vide.



(a) Arm_A



(b) $Clip_A$



(c) $Elev_A$

$(-, O, -)$	**when**	$(ar = back \wedge el = up) \vee (ar = front \wedge el = down)$,
$(-, C, -)$	**when**	$(ar = front \wedge el = up) \vee (ar = back \wedge el = down)$,
$(F, -, -)$	**when**	$(cl = open \wedge el = up) \vee (cl = close \wedge el = down)$,
$(B, -, -)$	**when**	$(cl = close \wedge el = up) \vee (cl = open \wedge el = down)$,
$(-, -, U1)$	**when**	$cl = close \wedge ar = back$,
$(-, -, U2)$	**when**	$cl = close \wedge ar = back$,
$(-, -, D1)$	**when**	$cl = close \wedge ar = back$,
$(-, -, D2)$	**when**	$cl = close \wedge ar = back$,
$(F, O, -)$	**when**	$el = up$,
$(B, C, -)$	**when**	$el = down$,
$(B, C, U1)$	**when**	$true$

(d) Syn^c_A

FIG. 5.3: $(Arm_A, Clip_A, Elev_A, Syn^c_A)$: composants et ensemble de synchronisations

1. Le composant Arm_A, illustré par la figure 5.3(a) spécifie un dispositif de déplacement horizontal qui peut prendre deux positions : avant ($ar = front$) et arrière ($ar = back$).
2. Le composant $Clip$ illustré par la figure 5.3(b) modélise le comportement d'une pince. La pince peut être fermée ($cl = close$) ou bien ouverte ($cl = open$).
3. Le composant $Elev_A$, illustré par la figure 5.3(c) décrit un dispositif de déplacement vertical à trois positions : une position basse ($el = down$), une position médiane ($el = mid$) et une position haute ($el = up$).

L'ensemble de synchronisations Syn^c_A, donné figure 5.3(d), permet d'exprimer comment les trois composants Arm_A, $Clip_A$ et $Elev_A$ interagissent les uns par rapport aux autres, tout en respectant les contraintes de fonctionnement du dispositif (voir section 1.1.4). Considérons par exemple la contrainte suivante : "quand le dispositif est en position haute, le bras peut s'avancer uniquement si la pince est déjà ouverte, ou alors s'avancer simultanément à l'ouverture de la pince". Cela se traduit par deux synchronisations contraintes :
- $(F, -, -)$ **when** $cl = open \wedge el = up$, qui indique que l'avancée a lieu quand on est en position haute, uniquement si la pince est ouverte, et
- $(F, O, -)$ **when** $el = up$, qui indique, quant-à-elle, que l'avancée et l'ouverture de la pince peuvent avoir lieu simultanément quand le dispositif est en position haute.

Nous pouvons bien entendu calculer le produit synchronisé contraint des trois composants Arm_A, $Clip_A$ et $Elev_A$ sous l'ensemble de synchronisations Syn^c_A (bien que nous ne souhaitions pas avoir besoin de le calculer pour en vérifier le raffinement) :

$$(Arm_A, Clip_A, Elev_A, Syn^c_A) = Clip_A \times_{Syn^c_A} Arm_A \times_{Syn^c_A} Elev_A$$

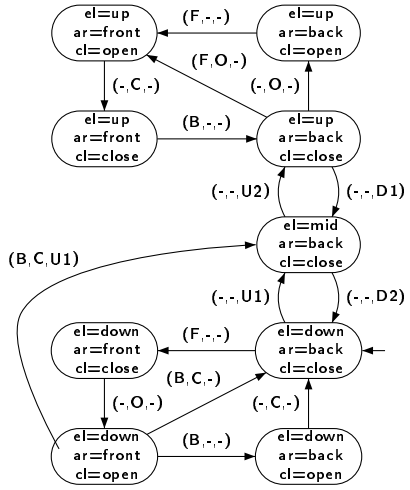


FIG. 5.4: $(Arm_A, Clip_A, Elev_A, Syn^c_A)$: produit synchronisé contraint

Le résultat est illustré par la figure 5.4. Remarquons que nous obtenons le même ST2E que dans les sections 1.1.4 et 2.2.4, modulo un renommage des transitions synchronisées.

5.1.3 Composants sous-contexte

Nous avons indiqué dans la définition 5.3 que la sémantique d'un système à composants synchronisés (S^1, S^2, Syn^c) était obtenue en calculant le produit synchronisé contraint $S^1 \times_{Syn^c} S^2$. Néanmoins, nous souhaitons pouvoir étudier le système à composants sans calculer ce produit, mais en focalisant notre analyse sur les composants du système pris les uns après les autres.

Considérons chaque composant du système. Il s'agit d'un composant "hors contexte", c.à.d. qu'il ne partage aucune variable avec les autres composants et son comportement propre est complètement défini indépendamment des autres composants ou de l'ensemble de synchronisations. Néanmoins, nous avons besoin, pour étudier de manière compositionnelle le raffinement d'un système à composants, de tenir compte de l'environnement d'un composant, c.à.d. des autres composants et de l'ensemble de synchronisations contraintes.

Nous restreignons, pour un composant donné, l'ensemble de synchronisations contraintes Syn^c défini 5.1 afin d'obtenir un nouvel ensemble de synchronisations $[Syn^c]_{S^1}$ plus "petit" : $[Syn^c]_{S^1} \subseteq Syn^c$. Cet ensemble de synchronisations est défini de telle manière qu'il ne contienne que les synchronisations concernant le composant considéré.

Définition 5.5 (Ensemble de synchronisations contraintes restreint)
Soit (S^1, S^2, Syn^c) un système à composants. L'ensemble de synchronisations restreint au composant S^1, noté $[Syn^c]_{S^1}$, est défini comme suit :

$$[Syn^c]_{S^1} \stackrel{def}{=} \{(e_1, e_2) \textbf{ when } sp \mid (e_1, e_2) \textbf{ when } sp \in Syn^c \wedge e_1 \in E^1 \wedge e_2 \in E^2 \cup \{-\}\}$$

Nous pouvons maintenant définir un *composant sous-contexte*, i.e., un composant dans le contexte des autres composants sous un ensemble de synchronisations. Pour cela, nous réutilisons la définition 5.2 du produit synchronisé contraint appliquée avec l'ensemble de synchronisations contraintes restreint au composant considéré.

Définition 5.6 (Composant sous-contexte)
Soit (S^1, S^2, Syn^c) *un système à composants. Le composant sous-contexte* $[S^1]^{S^2}_{Syn^c}$ *correspondant à* S^1 *dans le contexte de* S^2 *et de* Syn^c *est défini comme suit :*

$$[S^1]^{S^2}_{Syn^c} \stackrel{def}{=} S^1 \times_{[Syn^c]_{S^1}} S^2$$

En fait, dans le pire des cas, un composant sous-contexte aura la même taille que le système à composants complet, mais, en pratique, on obtient bien une réduction du nombre d'états et de transitions (voir aussi section 6.3).

La notion de composant sous-contexte présentée ici est proche de celle *d'extension d'un composant* proposée dans [AS02] pour la vérification compositionnelle de certaines propriétés CTL (voir section 3.3).

Remarque. On notera simplement le composant sous-contexte $[S^1]^{S^2}_{Syn^c}$, $[S^1]$ de manière abrégée, quand le contexte le permettra, et $([Q^1], [Q^1_O], [E^1], [T^1], [V^1], [l^1])$, sous la forme d'un six-uplet, quand cela sera nécessaire. Le composant sous-contexte $[S^2]^{S^1}_{Syn^c}$ peut être défini de manière similaire en calculant $S^1 \times_{[Syn^c]_{S^2}} S^2$. On généralisera la notion de composant sous-contexte à un composant pris dans le contexte de n composants, noté $[S^1]^{S^2,\ldots,S^n}_{Syn^c}$, de manière évidente.

Un composant sous-contexte peut aussi être vu comme le résultat d'une opération de masquage, dans le produit synchronisé complet, d'une partie de son comportement :

$$[S^1]^{S^2}_{Syn^c} = \mathcal{H}(S^1 \times_{Syn^c} S^2, \overline{[Syn^c]_{S^1}})$$

où $\mathcal{H}()$ est l'opération de masquage définie définition 1.6 et $\overline{[Syn^c]_{S^1}}$ le complémentaire de $[Syn^c]_{S^1}$.

> **Exemple.** La figure 5.5 donne $[Syn^c_A]_{Arm_A}$, l'ensemble de synchronisations Syn^c_A restreint au composant Arm_A. Nous pouvons donc calculer le composant sous-contexte correspondant à Arm_A dans le contexte des autres composants en calculant un produit synchronisé réduit, dont le résultat est illustré par la figure 5.6(a) :
>
> $$[Arm_A]^{Clip_A, Elev_A}_{Syn^c_A} = \Pi_{[Syn^c_A]_{Arm_A}}(Arm_A, Clip_A, Elev_A)$$
>
> Nous pouvons aussi calculer de manière successive les ensembles de synchronisations restreints, puis les composants sous-contexte correspondants aux composants $Clip_A$ et $Elev_A$: voir figures 5.6(b) et 5.6(c).

$(F,-,-)$ **when** $(cl = open \wedge el = up) \vee (cl = close \wedge el = down)$,
$(B,-,-)$ **when** $(cl = close \wedge el = up) \vee (cl = open \wedge el = down)$,
$(F,O,-)$ **when** $el = up$,
$(B,C,-)$ **when** $el = down$,
$(B,C,U1)$ **when** $true$

Fig. 5.5: Restriction de Syn^c_A pour le composant Arm_A

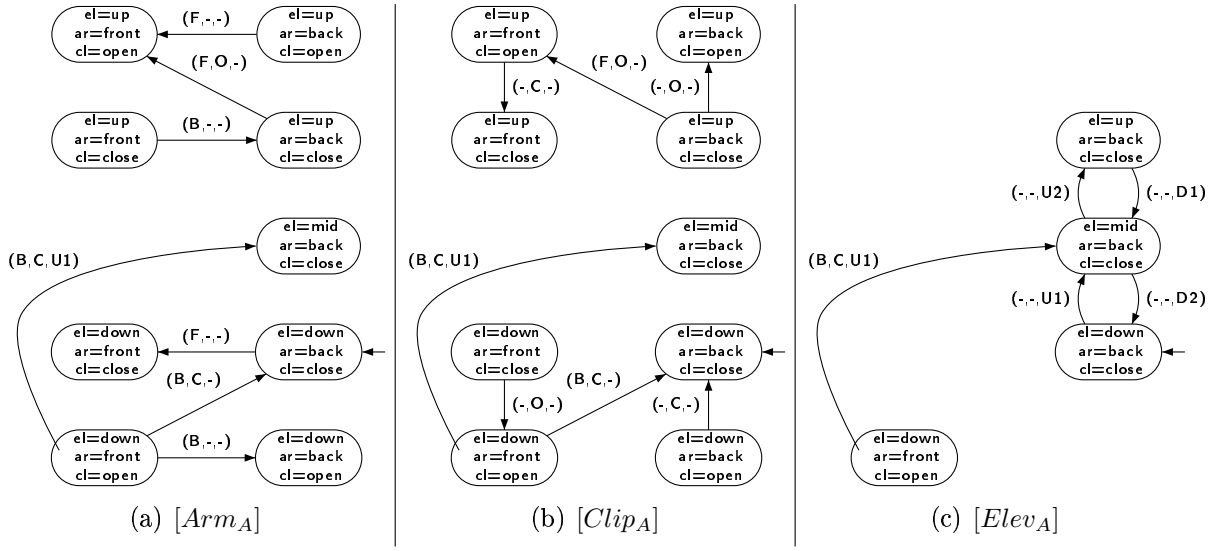


Fig. 5.6: Composants sous-contexte issus de $(Arm_A, Clip_A, Elev_A, Syn^c_A)$

5.1.4 Décomposition d'un système à composants

Remarquons que les composants sous-contexte forment des sous-systèmes du système à composants correspondant. Ils sont définis sur un même ensemble de variables $V^1 \cup V^2$. Cet ensemble de variables est aussi celui du système à composants synchronisés complet. Au lieu de calculer le produit synchronisé complet pour analyser (S^1, S^2, Syn^c), nous proposons de calculer "morceaux par morceaux" plusieurs produits synchronisés plus "petits" afin d'obtenir chacun des composants sous-contexte, c.à.d. des sous-systèmes du système à composants, et cela, sans avoir besoin de considérer le système dans son ensemble.

$$S^1 \times_{Syn^c} S^2 = (S^1 \times_{[Syn^c]_{S^1}} S^2) \cup (S^1 \times_{[Syn^c]_{S^2}} S^2) \tag{5.1}$$

Nous montrons maintenant que les composants sous-contexte forment une décomposition du système à composants synchronisés, au sens de la définition 4.2 donnée section 4.1.

Théorème 5.1 (Système à composants vs. composants sous contexte)
Soit (S^1, S^2, Syn^c) un système à composants. On a l'égalité suivante :

$$(S^1, S^2, Syn^c) = [S^1]^{S^2}_{Syn^c} \cup [S^2]^{S^1}_{Syn^c}$$

Preuve. Montrons que toutes les transitions $(q_1, q_2) \xrightarrow{(e_1,e_2)} (q'_1, q'_2)$ de (S^1, S^2, Syn^c) sont aussi, soit des transitions de $[S^1]$, soit des transitions de $[S^2]$, soit des transitions des deux composants

sous-contexte. Enumérons les différentes transitions de (S^1, S^2, Syn^c) à partir des définitions 5.2 et 5.1 :

- $(q_1, q_2) \xrightarrow{(e_1,-)} (q'_1, q_2)$ et $(e_1, -)$ **when** $sp \in Syn^c$. Grâce à la définition 5.5, on a $(e_1, -)$ **when** $sp \in [Syn^c]_{S^1}$ et, grâce à la définition 5.6, $(q_1, q_2) \xrightarrow{(e_1,-)} (q'_1, q_2) \in [T^1]$ est bien dans $[S^1]$.
- $(q_1, q_2) \xrightarrow{(-,e_2)} (q_1, q'_2)$ et $(-, e_2)$ **when** $sp \in Syn^c$. Grâce à la définition 5.5, on a $(-, e_2)$ **when** $sp \in [Syn^c]_{S^2}$ et, grâce à la définition 5.6, $(q_1, q_2) \xrightarrow{(-,e_2)} (q_1, q'_2) \in [T^2]$ est bien dans $[S^2]$.
- $(q_1, q_2) \xrightarrow{(e_1,e_2)} (q'_1, q'_2)$ et (e_1, e_2) **when** $sp \in Syn^c$. Grâce à la définition 5.5, on a
  - (e_1, e_2) **when** $sp \in [Syn^c]_{S^1}$ et, grâce à la définition 5.6, $(q_1, q_2) \xrightarrow{(e_1,e_2)} (q'_1, q'_2) \in [T^1]$ est bien dans $[S^1]$, et
  - (e_1, e_2) **when** $sp \in [Syn^c]_{S^2}$ et, grâce à la définition 5.6, $(q_1, q_2) \xrightarrow{(e_1,e_2)} (q'_1, q'_2) \in [T^2]$ est bien dans $[S^2]$.

La réciproque se montre de manière similaire.

□

Exemple. On va illustrer le théorème précédent sur l'exemple du bras mobile robotisé. Les composants sous-contexte $[Arm_A]$, $[Clip_A]$ et $[Elev_A]$ forment bien des sous-systèmes de $(Arm_A, Clip_A, Elev_A, Syn^c_A)$, au sens de la définition 4.1.

$$(Arm_A, Clip_A, Elev_A, Syn^c_A) = [Arm_A] \cup [Clip_A] \cup [Elev_A]$$

Le théorème 5.1 donne une manière automatique pour obtenir une décomposition d'un système à composants, sans avoir besoin de calculer le produit synchronisé complet, mais en calculant uniquement plusieurs produits plus "petits" en utilisant la définition 5.6. Puisque nous obtenons une décomposition, nous pouvons bénéficier des résultats présentés dans le chapitre 4 à propos de la vérification de la correction du raffinement et de la vérification de certaines propriétés.

5.2 Raffinement des systèmes à composants

Dans la section précédente nous avons vu qu'un système à composants synchronisés pouvait systématiquement se décomposer grâce à la notion de composants sous-contexte. Considérons un système à composants abstrait $SA = (SA^1, SA^2, Syn^c_A)$ ainsi qu'un système à composants plus concret $SR = (SR^1, SR^2, Syn^c_R)$. Puisque les systèmes à composants se décomposent, nous pouvons envisager d'appliquer le théorème 4.2 pour étudier si SR est bien un raffinement de SA.

Nous verrons dans cette section que nous allons réutiliser le théorème 4.2 et vérifier les raffinements affaiblis $\mathcal{T}([SR^1]) \sqsubseteq^{D_1}_{\rho_w} [SA^1]$ et $\mathcal{T}([SR^2]) \sqsubseteq^{D_2}_{\rho_w} [SA^2]$; cela d'une manière adaptée aux systèmes à composants, étant données les spécificités des systèmes à composants.

5.2.1 Exemple de raffinement d'un système à composants : le bras mobile robotisé

Nous donnons maintenant $(Arm_R, Clip_R, Elev_R, Syn^c_R)$, un système à composants correspondant à un raffinement du système à composants $(Arm_A, Clip_A, Elev_A, Syn^c_A)$ introduit dans la section 5.1.1. Nous avons montré dans la section précédente que le système à composants $(Arm_A, Clip_A, Elev_A, Syn^c_A)$ se décomposait en trois sous-systèmes $[Arm_A]$, $[Clip_A]$ et $[Elev_A]$ grâce à la notion de composants sous-contexte et à la restriction de l'ensemble de synchronisations.

Le système à composants $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ est élaboré à partir de trois composants simples Arm_R, $Clip_R$ et $Elev_R$ et d'un ensemble de synchronisations contraintes Syn^c_R.

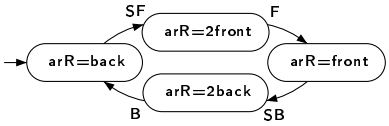
1. Le composant Arm_R, illustré par la figure 5.7(a) décrit un raffinement de Arm_A. On observe maintenant le point de départ des déplacements horizontaux, grâce aux valeurs que peut prendre la variable arR : $arR = back$ pour indiquer que le dispositif est en position arrière, $arR = 2front$, pour indiquer que le dispositif commence à avancer, $arR = front$, pour indiquer que le dispositif est en position avancée et $arR = 2back$, pour indiquer que le dispositif commence à reculer.
2. Le composant $Clip_R$, illustré par la figure 5.7(b) modélise le comportement d'une pince avec plus de détails que $Clip_A$. La pince peut être fermée ($clR = close$), ou en train de s'ouvrir ($clR = 2open$), ou bien ouverte ($clR = open$) ou encore en train de se fermer ($clR = 2close$).
3. Le composant $Elev_R$, illustré par la figure 5.7(c) décrit un dispositif de déplacement vertical à trois positions dans lequel, on observe le point de départ des différents déplacements. La variable elR peut prendre sept valeurs, pour traduire les différents états du dispositif : $elR \in \{down, do2mid, mid, 2up, up, up2mid, 2down\}$.
4. L'ensemble de synchronisations contraintes Syn^c_R, donné figure 5.7(d) exprime comment les trois composants Arm_R, $Clip_R$ et $Elev_R$ interagissent les uns par rapport aux autres, tout en respectant les contraintes de fonctionnement du dispositif, pour décrire le comportement global de $(Arm_R, Clip_R, Elev_R, Syn^c_R)$.

Nous pouvons obtenir une représentation du système à composants synchronisés $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ en calculant le produit synchronisé contraint des trois composants Arm_R, $Clip_R$ et $Elev_R$ sous l'ensemble de synchronisations Syn^c_R grâce à la définition 5.2.

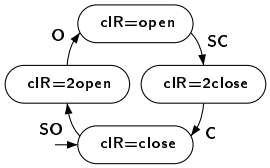
$$(Arm_R, Clip_R, Elev_R, Syn^c_R) = Clip_R \times_{Syn^c_R} Arm_R \times_{Syn^c_R} Elev_R$$

La figure 5.8 illustre le système à composants synchronisés $(Arm_R, Clip_R, Elev_R, Syn^c_R)$. Notons que le ST2E obtenu est le même que celui présenté section 2.2.4, modulo un renommage des transitions synchronisées. Nous sommes donc assurés que

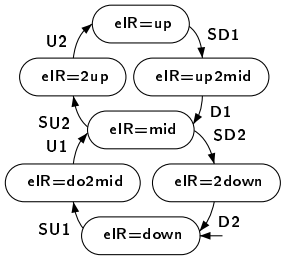
$$\mathcal{T}(Arm_R, Clip_R, Elev_R, Syn^c_R) \sqsubseteq_\eta (Arm_A, Clip_A, Elev_A, Syn^c_A)$$



(a) Arm_R



(b) $Clip_R$



(c) $Elev_R$

$(-, SO, -)$	**when**	$(arR = back \wedge elR = up) \vee (arR = front \wedge elR = down)$,
$(-, O, -)$	**when**	$(arR \in \{back, 2front\} \wedge elR = up) \vee (arR = front \wedge elR = down)$,
$(-, SC, -)$	**when**	$(arR = front \wedge elR = up) \vee (arR \in \{back, 2back\} \wedge elR = down)$,
$(-, C, -)$	**when**	$(arR = front \wedge elR = up) \vee (arR = back \wedge elR = down)$,
$(SF, -, -)$	**when**	$(clR \in \{open, 2open\} \wedge elR = up) \vee (clR = close \wedge elR = down)$,
$(F, -, -)$	**when**	$(clR = open \wedge elR = up) \vee (clR = close \wedge elR = down)$,
$(SB, -, -)$	**when**	$(clR = close \wedge elR = up) \vee (clR = open \wedge elR = down)$,
$(B, -, -)$	**when**	$(clR = close \wedge elR = up) \vee (clR \in \{open, 2close\} \wedge elR = down)$,
$(-, -, SU1)$	**when**	$clR \in \{close, 2close\} \wedge arR = back$,
$(-, -, U1)$	**when**	$clR = close \wedge arR = back$,
$(-, -, SU2)$	**when**	$clR = close \wedge arR = back$,
$(-, -, U2)$	**when**	$clR = close \wedge arR = back$,
$(-, -, SD1)$	**when**	$clR = close \wedge arR = back$,
$(-, -, D1)$	**when**	$clR = close \wedge arR = back$,
$(-, -, SD2)$	**when**	$clR = close \wedge arR = back$,
$(-, -, D2)$	**when**	$clR = close \wedge arR = back$,
$(SF, SO, -)$	**when**	$elR = up$,
$(SF, O, -)$	**when**	$elR = up$,
$(F, O, -)$	**when**	$elR = up$,
$(SB, SC, -)$	**when**	$elR = down$,
$(B, SC, -)$	**when**	$elR = down$,
$(B, C, -)$	**when**	$elR \in \{down, do2mid\}$,
$(-, SC, SU1)$	**when**	$arR = 2back$,
$(SB, SC, SU1)$	**when**	$true$,
$(B, C, SU1)$	**when**	$true$,
$(B, C, U1)$	**when**	$true$

(d) Syn^c_R

FIG. 5.7: $(Arm_R, Clip_R, Elev_R, Syn^c_R)$: composants et ensemble de synchronisations

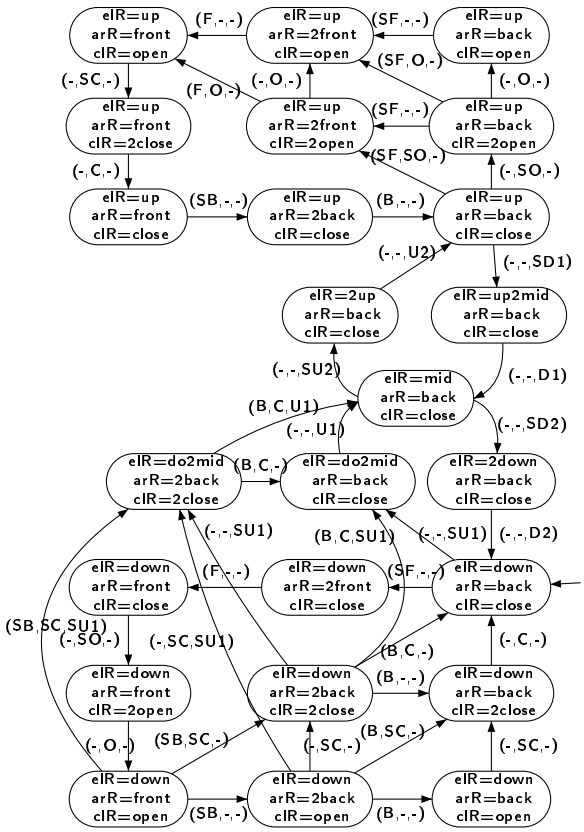
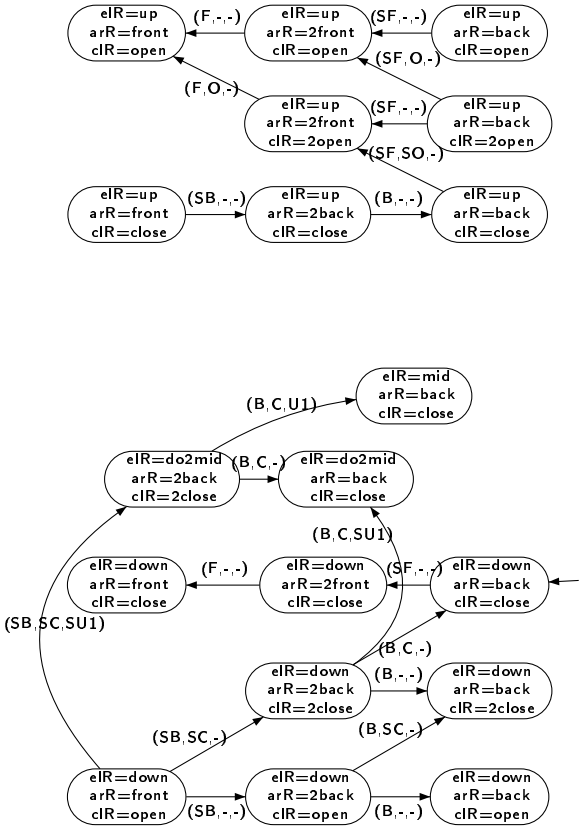
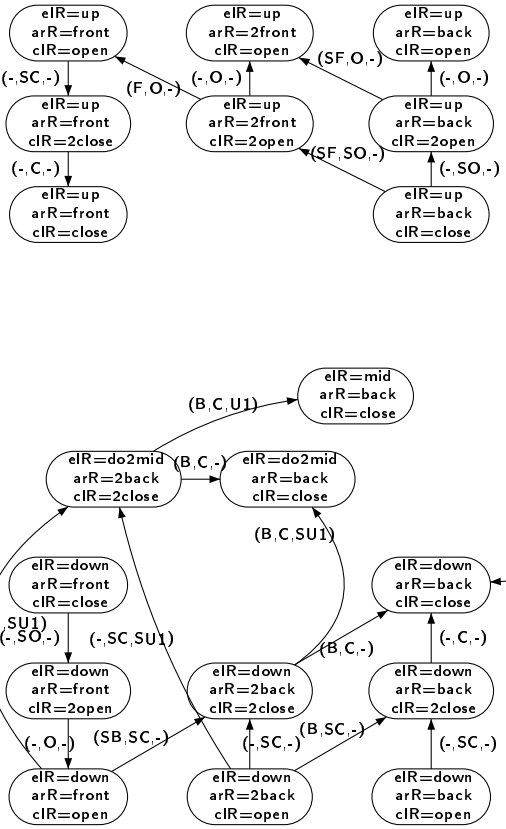


FIG. 5.8: $(Arm_R, Clip_R, Elev_R, Syn^c_R)$: produit synchronisé contraint



(a) $[Arm_R]$



(b) $[Clip_R]$

(c) $[Elev_R]$

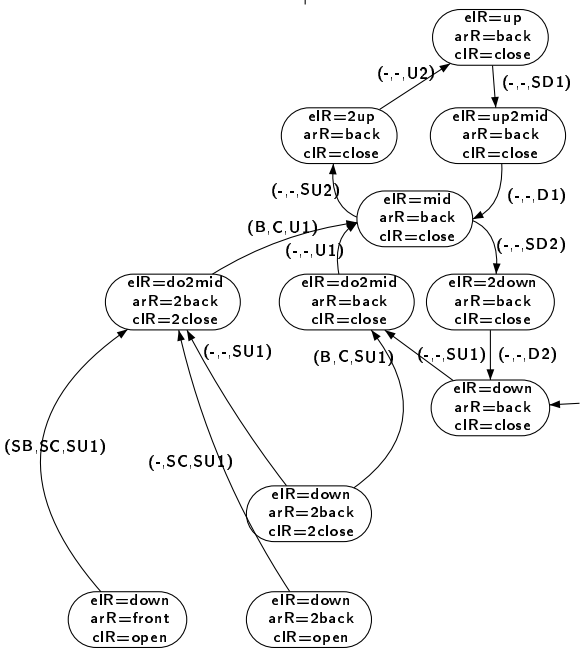


FIG. 5.9: Composants sous-contexte issus de $(Arm_R, Clip_R, Elev_R, Syn^c_R)$

Remarque. Il faut noter que nous n'avons calculé qu'à titre d'illustration le produit synchronisé complet. Rappelons que l'idée principale dans notre démarche est de ne calculer ni le produit synchronisé abstrait complet, ni le produit synchronisé raffiné complet pour en étudier le raffinement, mais de calculer, de manière successive, des produits plus "petits" correspondant aux composants sous-contexte.

Les définitions 5.5 et 5.6 permettent de calculer, successivement et pour chacun des composants raffinés, l'ensemble de synchronisations restreint au composant considéré, puis le composant sous-contexte correspondant. Nous obtenons ainsi automatiquement les composants sous-contexte $[Arm_R]$, $[Clip_R]$ et $[Elev_R]$, illustrés par la figure 5.9. Nous pouvons montrer que $[Arm_R]$, $[Clip_R]$ et $[Elev_R]$ sont bien des sous-systèmes de $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ et qu'ils en forment bien une décomposition :

$$(Arm_R, Clip_R, Elev_R, Syn^c_R) = [Arm_R] \cup [Clip_R] \cup [Elev_R]$$

Nous donnons figure 5.10 un prédicat de collage gp afin d'établir un lien entre les variables des composants abstraits Arm_A, $Clip_A$ et $Elev_A$ et celles des composants raffinés Arm_R, $Clip_R$ et $Elev_R$. Ce prédicat de collage nous permettra d'étudier le raffinement affaibli des composants sous contexte $[Arm_R]$, $[Clip_R]$ et $[Elev_R]$, mais aussi le raffinement strict du système à composants synchronisés $(Arm_R, Clip_R, Elev_R, Syn^c_R)$. Grâce à gp nous pouvons montrer que les composants simples raffinés sont bien des raffinements des composants simples abstraits. Nous avons $\mathcal{T}(Arm_R) \sqsubseteq_\eta Arm_A$, $\mathcal{T}(Clip_R) \sqsubseteq_\eta Clip_A$ et également $\mathcal{T}(Elev_R) \sqsubseteq_\eta Elev_A$.

$$\begin{array}{lll} & el = up & \Leftrightarrow \quad elR \in \{up, up2mid\} \\ \wedge & el = mid & \Leftrightarrow \quad elR \in \{2up, mid, 2down\} \\ \wedge & el = down & \Leftrightarrow \quad elR \in \{down, do2mid\} \\ \wedge & ar = front & \Leftrightarrow \quad arR \in \{front, 2back\} \\ \wedge & ar = back & \Leftrightarrow \quad arR \in \{back, 2front\} \\ \wedge & cl = open & \Leftrightarrow \quad clR \in \{open, 2close\} \\ \wedge & cl = close & \Leftrightarrow \quad clR \in \{close, 2open\} \end{array}$$

Fig. 5.10: Prédicat de collage gp

5.2.2 Renommage par τ dans un système à composants

La première étape pour la vérification du raffinement consiste à renommer par τ les transitions étiquetées par de nouvelles étiquettes de transitions. Ce renommage par τ n'est pas aussi simple à réaliser dans le cas des systèmes à composants synchronisés que dans le cas général : puisque nous ne faisons aucune restriction sur l'ensemble de synchronisations raffiné Syn^c_R, cet ensemble peut contenir des transitions étiquetées de différentes manières, soit par des couples d'anciennes étiquettes, soit par des couples de nouvelles étiquettes, soit par des couples formés d'une ancienne et d'une nouvelle étiquette. Dans ce dernier cas, est-ce qu'il s'agit d'une ancienne ou d'une nouvelle transition ?

Nous proposons l'approche suivante pour exprimer le renommage par τ dans le cas d'un système à composants raffinés (SR^1, SR^2, Syn^c_R). Considérons une transition synchronisée étiquetée (e_1, e_2) :

1. Si (e_1, e_2) est formée uniquement d'étiquettes anciennes, alors la transition synchronisée est conservée à l'identique.
2. Si (e_1, e_2) est formée uniquement d'étiquettes nouvelles, alors la transition synchronisée est renommée par τ, c.à.d. $(e_1, e_2)\backslash\tau$.

$$\mathcal{R}((SR^1, SR^2, Syn_R^c), ((E_R^1 \setminus E_A^1) \cup \{-\}) \times ((E_R^2 \setminus E_A^2) \cup \{-\}), \tau)$$

3. Si (e_1, e_2) est formée d'une ancienne étiquette et d'une nouvelle étiquette alors uniquement la partie ancienne e_1 (resp. e_2) de l'étiquette de la transition synchronisée doit être conservée. La partie nouvelle e_2 (resp. e_1) est remplacée par '$-$' uniquement afin de conserver un typage cohérent pour les étiquettes des transitions, c.à.d. $(e_1, e_2)\backslash(e_1, -)$ (resp. $(e_1, e_2)\backslash(-, e_2)$).

$$\forall e_1 \in E_A^1.\ \mathcal{R}((SR^1, SR^2, Syn_R^c), \{e_1\} \times (E_R^2 \setminus E_A^2), (e_1, -))$$

$$\forall e_2 \in E_A^2.\ \mathcal{R}((SR^1, SR^2, Syn_R^c), (E_R^1 \setminus E_A^1) \times \{e_2\}, (-, e_2))$$

Nous étendons aux systèmes à composants synchronisés, la définition 2.3.

Définition 5.7 (Renommage par τ vs. un système à composants)
Soient (SA^1, SA^2, Syn_A^c) et (SR^1, SR^2, Syn_R^c) deux systèmes à composants, avec $(SR^1, SR^2, Syn_R^c) = (Q_R, Q_0, E_R, T_R, V_R, l_R)$. Le système $\mathcal{T}_{SA^1,SA^2}(SR^1, SR^2, Syn_R^c)$ est donné par le six-uplet $(Q_R, Q_0, E'_R, T'_R, V_R, l_R)$ suivant :

- *E'_R est définie ainsi :*
  - *$(e_1, -) \in E'_R$ si $(e_1, -) \in E_R$ et $e_1 \in E_A^1$,*
  - *$(-, e_2) \in E'_R$ si $(-, e_2) \in E_R$ et $e_2 \in E_A^2$,*
  - *$(e_1, e_2) \in E'_R$ si $(e_1, e_2) \in E_R$, $e_1 \in E_A^1$ et $e_2 \in E_A^2$,*
  - *$(e_1, -)\backslash\tau \in E'_R$ si $(e_1, -) \in E_R$ et $e_1 \in E_R^1 \setminus E_A^1$,*
  - *$(-, e_2)\backslash\tau \in E'_R$ si $(-, e_2) \in E_R$ et $e_2 \in E_R^2 \setminus E_A^2$,*
  - *$(e_1, e_2)\backslash\tau \in E'_R$ si $(e_1, e_2) \in E_R$, $e_1 \in E_R^1 \setminus E_A^1$ et $e_2 \in E_R^2 \setminus E_A^2$,*
  - *$(e_1, e_2)\backslash(e_1, -) \in E'_R$ si $(e_1, e_2) \in E_R$, $e_1 \in E_A^1$ et $e_2 \in E_R^2 \setminus E_A^2$,*
  - *$(e_1, e_2)\backslash(-, e_2) \in E'_R$ si $(e_1, e_2) \in E_R$, $e_1 \in E_R^1 \setminus E_A^1$ et $e_2 \in E_A^2$*
- *$T'_R = \{q \xrightarrow{(e_1,e_2)\backslash(e'_1,e'_2)} q' \mid (e_1, e_2)\backslash(e'_1, e'_2) \in E'_R \wedge q \xrightarrow{(e_1,e_2)} q' \in T_R\}$*

Exemple. Nous pouvons appliquer la définition 5.7 sur le système à composants raffiné $(Arm_R, Clip_R, Elev_R, Syn_R^c)$ afin d'obtenir un nouveau système à composants $\mathcal{T}(Arm_R, Clip_R, Elev_R, Syn_R^c)$ dans lequel les transitions synchronisées étiquetées par de nouvelles étiquettes de transitions ont été renommées par τ (voir figure 5.11).

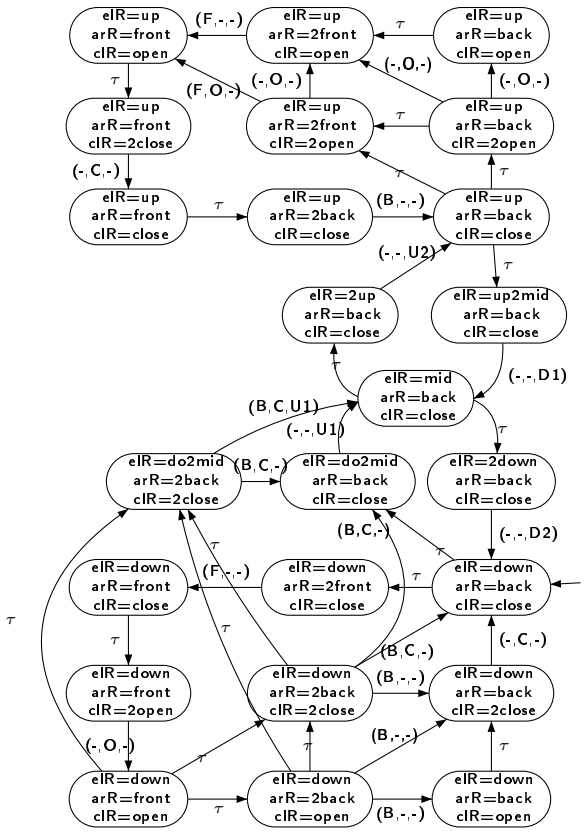


Fig. 5.11: Renommage par τ dans $(Arm_R, Clip_R, Elev_R, Syn^c_R)$

5.2.3 Absence de τ-cycles dans un système à composants

L'une des conditions du théorème 4.2 concerne l'absence de τ-cycles dans le système complet, c.à.d. $\neg\ div^\tau(\mathcal{T}(SR))$. Dans le chapitre précédent, nous n'avions pas abordé ce problème. Dans cette section, nous allons montrer que si $SR = (SR^1, SR^2, Syn_R^c)$ est un système à composants, nous pouvons maintenant traiter le problème de l'absence de τ-cycles, étant donné la nature compositionnelle des systèmes à composants : vérifier l'absence de τ-cycles pour chacun des composants simples SR^1 et SR^2 permet automatiquement de vérifier l'absence de τ-cycles pour le système complet SR.

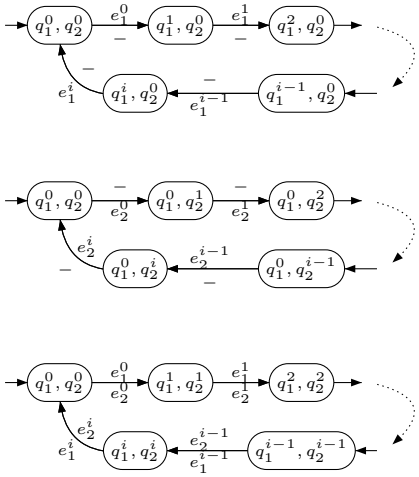
Lemme 5.1 (Absence de τ-cycles)
Soit $\mathcal{T}_{SA^1,SA^2}(SR^1, SR^2, Syn_R^c)$ un système à composants. On a

$$\neg\ div^\tau(\mathcal{T}_{SA^1}(SR^1)) \wedge \neg\ div^\tau(\mathcal{T}_{SA^2}(SR^2)) \Rightarrow \neg\ div^\tau(\mathcal{T}_{SA^1,SA^2}(SR^1, SR^2, Syn_R^c))$$

PREUVE. (par contradiction) Supposons qu'il existe un τ-cycle dans $\mathcal{T}_{SA^1,SA^2}(SR^1, SR^2, Syn_R^c)$.

Soit $\sigma = (q_1^0, q_2^0) \xrightarrow{(e_1^0,e_2^0)\backslash\tau} (q_1^1, q_2^1) \rightarrow \ldots \rightarrow (q_1^i, q_2^i) \xrightarrow{(e_1^i,e_2^i)\backslash\tau} (q_1^0, q_2^0) \ldots$ un tel τ-cycle avec $\forall j \in \{1,2\}, \forall k \geq 0\ .\ e_j^k \in E_R^j \cup \{-\}$ et $e_1^0 \in E_R^1\ \vee\ e_2^0 \in E_R^2$.

1. Si $\forall k \geq 0\ .\ e_2^k =' -'$,
   $proj(\sigma, \mathcal{T}_{SA^1}(SR^1)) = q_1^0 \xrightarrow{e_1^0\backslash\tau} q_1^1 \rightarrow \ldots \rightarrow q_1^i \xrightarrow{e_1^i\backslash\tau} q_1^0 \ldots$ et
   $proj(\sigma, \mathcal{T}_{SA^2}(SR^2)) = q_2^0$.
   On a donc $div^\tau(\mathcal{T}_{SA^1}(SR^1))$.
2. Si $\forall k \geq 0\ .\ e_1^k =' -'$,
   $proj(\sigma, \mathcal{T}_{SA^1}(SR^1)) = q_1^0$ et
   $proj(\sigma, \mathcal{T}_{SA^2}(SR^2)) = q_2^0 \xrightarrow{e_2^0\backslash\tau} q_2^1 \rightarrow \ldots \rightarrow q_2^i \xrightarrow{e_2^i\backslash\tau} q_2^0 \ldots$
   On a donc $div^\tau(\mathcal{T}_{SA^2}(SR^2))$.
3. Sinon,
   $proj(\sigma, \mathcal{T}_{SA^1}(SR^1)) = q_1^0 \xrightarrow{e_1^0\backslash\tau} q_1^1 \rightarrow \ldots \rightarrow q_1^i \xrightarrow{e_1^i\backslash\tau} q_1^0 \ldots$ et
   $proj(\sigma, \mathcal{T}_{SA^2}(SR^2)) = q_2^0 \xrightarrow{e_2^0\backslash\tau} q_2^1 \rightarrow \ldots \rightarrow q_2^i \xrightarrow{e_2^i\backslash\tau} q_2^0 \ldots$
   On a donc $div^\tau(\mathcal{T}_{SA^1}(SR^1))$ et $div^\tau(\mathcal{T}_{SA^2}(SR^2))$.



□

Si l'on suppose que chaque composant simple qui compose le système complet est construit par raffinement, c.à.d. que $\mathcal{T}_{SA^1}(SR^1) \sqsubseteq_\eta SA^1$ et $\mathcal{T}_{SA^2}(SR^2) \sqsubseteq_\eta SA^2$ alors la vérification séparée de l'absence de τ-cycles dans $\mathcal{T}_{SA^1}(SR^1)$ et dans $\mathcal{T}_{SA^2}(SR^2)$ n'est pas nécessaire à cause de la condition de correction 4 de la définition 2.7 de la relation de raffinement η qui assure cette absence de τ-cycles.

> **Exemple.** On a précisé dans la section 5.2.1 que $\mathcal{T}(Arm_R) \sqsubseteq_\eta Arm_A$, $\mathcal{T}(Clip_R) \sqsubseteq_\eta Clip_A$ et que $\mathcal{T}(Elev_R) \sqsubseteq_\eta Elev_A$. On est donc assuré de l'absence de τ-cycles dans chacun des composants simples. Le lemme 5.1 permet d'affirmer que le système à composants synchronisés $\mathcal{T}(Arm_R, Clip_R, Elev_R, Syn_R^c)$ ne contient pas non plus de τ-cycles.

5.2.4 Renommage par τ dans un composant sous-contexte

Dans la section 5.2.2, nous avons indiqué comment réaliser le renommage par τ dans un système à composants synchronisé complet. Un problème similaire se pose lorsqu'il s'agit d'identifier dans un composant sous-contexte quelles transitions synchronisées doivent être renommées par τ. Le cas des transitions synchronisées formées d'une étiquette ancienne et d'une étiquette nouvelle est là aussi problématique. Considérons le composant sous-contexte $[SR^1]$ et une transition synchronisée étiquetée (e_1, e_2) :

1. Les cas où (e_1, e_2) est formée uniquement d'étiquettes anciennes ou uniquement d'étiquettes nouvelles sont traitées comme dans la section 5.2.2.
2. Si (e_1, e_2) est formée d'une partie ancienne correspondant au composant $[SR^1]$ et d'une partie nouvelle correspondant à l'autre composant alors uniquement la partie ancienne de l'étiquette de la transition synchronisée doit être conservée.
$$\forall e_1 \in E_A^1.\ \mathcal{R}([SR^1], \{e_1\} \times (E_R^2 \setminus E_A^2), (e_1, -))$$
3. Si (e_1, e_2) est formée d'une partie nouvelle appartenant au composant $[SR^1]$ et d'une partie ancienne appartenant à l'autre composant ($e_1 \in E_R^1 \setminus E_A^1$ et $e_2 \in E_A^2$) alors la transition synchronisée doit être supprimée.
$$\mathcal{H}([SR^1], (E_R^1 \setminus E_A^1) \times E_A^2)$$
Notons que nous pouvons supprimer cette transition puisqu'elle est aussi une transition synchronisée dans l'autre composant sous-contexte (c.f. définition 5.5) et qu'elle sera traitée comme en 2. pour cet autre composant sous-contexte.

Définition 5.8 (Renommage par τ dans un composant sous-contexte)
Soient (SA^1, SA^2, Syn_A^c) et (SR^1, SR^2, Syn_R^c) deux systèmes à composants. Considérons le composant sous-contexte $[SR^1] = ([Q_R], [Q_0], [E_R], [T_R], [V_R], [l_R])$. Le composant sous-contexte $\mathcal{T}_{SA_1,SA_2}([SR^1])$ est donné par le six-uplet $([Q'_R], [Q'_0], [E'_R], [T'_R], [V_R], [l'_R])$ suivant :
- $[Q'_R] \subseteq [Q_R]$,
- $[Q'_0] \subseteq [Q_0]$,
- $[E'_R]$ *est définie ainsi :*
  - $(e_1, -) \in [E'_R]$ *si* $(e_1, -) \in [E_R]$ *et* $e_1 \in E_A^1$
  - $(e_1, e_2) \in [E'_R]$ *si* $(e_1, e_2) \in [E_R]$, $e_1 \in E_A^1$ *et* $e_2 \in E_A^2$
  - $(e_1, -)\backslash\tau \in [E'_R]$ *si* $(e_1, -) \in [E_R]$ *et* $e_1 \in E_R^1 \setminus E_A^1$
  - $(e_1, e_2)\backslash\tau \in [E'_R]$ *si* $(e_1, e_2) \in [E_R]$, $e_1 \in E_R^1 \setminus E_A^1$ *et* $e_2 \in E_R^2 \setminus E_A^2$
  - $(e_1, e_2)\backslash(e_1, -) \in [E'_R]$ *si* $(e_1, e_2) \in [E_R]$, $e_1 \in E_A^1$ *et* $e_2 \in E_R^2 \setminus E_A^2$,
- $[T'_R] = \{q \xrightarrow{e_R \backslash e'_R} q' \mid\ e_R \backslash e'_R \in [E'_R] \wedge q \xrightarrow{e_R} q' \in [T_R]\}$,
- $\forall q_R \in [Q'_R],\ [l'_R](q_R) = [l_R](q_R)$

Le composant sous-contexte $\mathcal{T}_{SA_1,SA_2}([SR^2])$ peut être défini de manière similaire.

Nous allons maintenant montrer que les renommages par τ dans un système à composants synchronisés (définition 5.7) et dans les composants sous-contexte (définition 5.8) respectent la décomposition d'un système en plusieurs sous-systèmes (définition 5.1).

Lemme 5.2 (Décomposition vs. renommage par τ)
Soient (SA^1, SA^2, Syn_A^c) et (SR^1, SR^2, Syn_R^c) deux systèmes à composants. Nous avons l'égalité suivante :

$$\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c) = \mathcal{T}_{SA_1,SA_2}([SR^1]) \cup \mathcal{T}_{SA_1,SA_2}([SR^2])$$

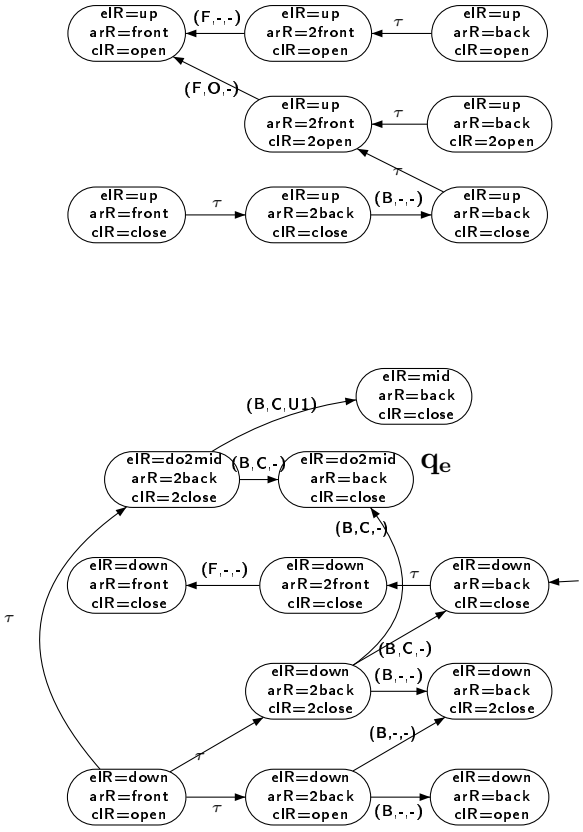
PREUVE.

1. Montrons que, quelque soit la transition de (SR^1, SR^2, Syn_R^c), elle est aussi présente dans un des composants sous-contexte au moins, et qu'elle est renommée de la même manière dans le système à composants et dans les composants sous-contexte dans laquelle elle est présente. Nous noterons e_1 (resp. e_2) une ancienne étiquette de E_A^1 (resp. E_A^2) et e_1' (resp. e_2') une nouvelle étiquette de $E_R^1 \setminus E_A^1$ (resp. $E_R^2 \setminus E_A^2$).
   - $q_R \xrightarrow{(e_1,-)} q_R'$. Cette transition est présente uniquement dans $[SR^1]$. Elle n'est pas renommée dans $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$, ni dans $\mathcal{T}_{SA_1,SA_2}([SR^1])$.
   - $q_R \xrightarrow{(-,e_2)} q_R'$. Cette transition est présente uniquement dans $[SR^2]$. Elle n'est pas renommée dans $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$, ni dans $\mathcal{T}_{SA_1,SA_2}([SR^2])$.
   - $q_R \xrightarrow{(e_1,e_2)} q_R'$. Cette transition est présente dans $[SR^1]$ et dans $[SR^2]$. Elle n'est pas renommée dans $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$, ni dans $\mathcal{T}_{SA_1,SA_2}([SR^1])$, ni dans $\mathcal{T}_{SA_1,SA_2}([SR^2])$.
   - $q_R \xrightarrow{(e_1',-)} q_R'$. Cette transition est présente uniquement dans $[SR^1]$. Elle est renommée par τ dans $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$ et dans $\mathcal{T}_{SA_1,SA_2}([SR^1])$.
   - $q_R \xrightarrow{(-,e_2')} q_R'$. Cette transition est présente uniquement dans $[SR^2]$. Elle est renommée par τ dans $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$ et dans $\mathcal{T}_{SA_1,SA_2}([SR^2])$.
   - $q_R \xrightarrow{(e_1',e_2')} q_R'$. Cette transition est présente dans $[SR^1]$ et dans $[SR^2]$. Elle est renommée par τ dans $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$, dans $\mathcal{T}_{SA_1,SA_2}([SR^1])$ et dans $\mathcal{T}_{SA_1,SA_2}([SR^2])$.
   - $q_R \xrightarrow{(e_1,e_2')} q_R'$. Cette transition est présente dans $[SR^1]$ et dans $[SR^2]$. Elle est renommée par $(e_1, -)$ dans $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$ et dans $\mathcal{T}_{SA_1,SA_2}([SR^1])$. Par contre elle n'est plus présente dans $\mathcal{T}_{SA_1,SA_2}([SR^2])$.
   - $q_R \xrightarrow{(e_1',e_2)} q_R'$. Cette transition est présente dans $[SR^1]$ et dans $[SR^2]$. Elle est renommée par $(-, e_2)$ dans $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$ et dans $\mathcal{T}_{SA_1,SA_2}([SR^2])$. Par contre elle n'est plus présente dans $\mathcal{T}_{SA_1,SA_2}([SR^1])$.
2. La réciproque est aussi immédiate.

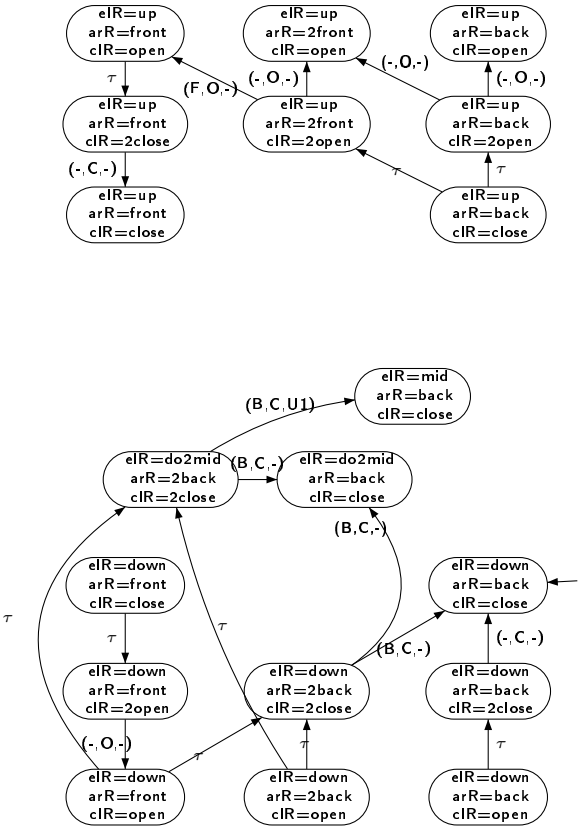
□

Exemple. Nous pouvons illustrer le résultat précédent avec l'exemple du système à composants $(Arm_R, Clip_R, Elev_R, Syn_R^c)$. Commençons par appliquer la définition 5.8 sur les composants sous-contexte $[Arm_R]$, $[Clip_R]$ et $[Elev_R]$ (voir figure 5.12). Nous pouvons ensuite obtenir l'égalité suivante :

$$\mathcal{T}(Arm_R, Clip_R, Elev_R, Syn_R^c) = \mathcal{T}([Arm_R]) \cup \mathcal{T}([Clip_R]) \cup \mathcal{T}([Elev_R])$$



(a) $\mathcal{T}([Arm_R])$



(b) $\mathcal{T}([Clip_R])$

(c) $\mathcal{T}([Elev_R])$

Fig. 5.12: Renommage par τ dans les composants sous-contexte

5.2.5 Ensemble des blocages réductibles

L'une des hypothèses du théorème 4.2 concerne la réduction des ensembles de blocages issus du raffinement affaibli des sous-systèmes. Le calcul de $D_1 \triangle D_2$ est coûteux puisqu'il nécessite de connaître simultanément les deux ensembles de nouveaux blocages D_1 et D_2 issus des raffinements affaiblis des sous-systèmes. On ne peut donc pas analyser le premier sous-système indépendamment du second.

Dans le cas des systèmes à composants, nous allons voir qu'il est possible de calculer un ensemble de blocages réductibles pour un des composants sous-contexte, uniquement à partir des composants simples et de l'ensemble de synchronisations. On pourra donc analyser successivement le raffinement de chacun des composants sous-contexte, sans avoir besoin d'étudier simultanément tous les ensembles de blocages.

L'ensemble RD_1 des blocages réductibles est calculé de la manière suivante : considérons un état q appartenant à D_1. Cet état q appartient à RD_1 s'il existe dans Syn_R^c une transition synchronisée concernant l'autre composant et activable depuis q. Si c'est la cas, cela signifie que q est aussi un état dans l'autre composant sous-contexte et qu'il est non-bloquant.

Définition 5.9 (Ensemble des blocages réductibles)
Soient (SA^1, SA^2, Syn_A^c) et (SR^1, SR^2, Syn_R^c) deux systèmes à composants tels que l'on ait $\mathcal{T}_{SA_1,SA_2}([SR^1]) \sqsubseteq_{\rho_w}^{D_1} [SA^1]$. L'ensemble RD_1 des blocages réductibles de D_1 est défini comme suit.

$$\begin{aligned} RD_1 \stackrel{def}{=} \{(q_1,q_2) \mid (q_1,q_2) \in D_1 \ \wedge\ \exists e_1, e_2.(e_1 \in E_R^1 \cup \{-\} \ \wedge\ e_2 \in E_R^2 \ \wedge \\ q_2 \stackrel{e_2}{\rightarrow} q_2' \in T_R^2 \ \wedge\ (e_1,e_2) \textbf{ when } sp \in Syn_R^c \ \wedge\ (q_1,q_2) \models sp)\} \end{aligned}$$

Montrons maintenant que le calcul de l'ensemble des blocages réductibles est suffisant pour s'assurer qu'un état de blocage est un vrai nouvel état de blocage introduit dans le produit synchronisé par le raffinement ou un faux nouvel état de blocage introduit par la décomposition du système en plusieurs composants sous-contexte.

Considérons le complémentaire la définition 4.5 : un état $q_R \in D_1$ est un "faux" nouvel état de blocage pour le raffinement du système complet si q_R est un état de $[SR^2]$ ($q_R \in [Q_R^2]$) et qu'il n'est pas état de blocage ($q_R \notin D_2$).

Afin d'étudier séparément les ensembles de blocages D_1 et D_2, nous allons montrer pour cela que si D_1 (resp. D_2) est complètement contenu par $[Q_R^2] \setminus D_2$ (resp. $[Q_R^1] \setminus D_1$), cela signifie que D_1 (resp. D_2) ne contenait que de "faux" nouveaux états de blocage pour le produit synchronisé. La réduction des blocages pour le système complet $D_1 \triangle D_2$ sera égale à l'ensemble vide et la clause 3 de la définition 2.7 du raffinement strict sera maintenue pour le système complet.

Lemme 5.3 (Ensemble de blocages réductibles et réduction des blocages)
Soient (SA^1, SA^2, Syn_A^c) et $\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c)$ deux systèmes à composants tels que l'on ait $\mathcal{T}_{SA_1,SA_2}([SR^1]) \sqsubseteq_{\rho_w}^{D_1} [SA^1]$ et $\mathcal{T}_{SA_1,SA_2}([SR^2]) \sqsubseteq_{\rho_w}^{D_2} [SA^2]$. Soit RD_1 et RD_2 les

ensembles des blocages réductibles de D_1 et D_2. On a alors

$$\left.\begin{array}{l} D_1 = RD_1 \\ D_2 = RD_2 \end{array}\right\} \Rightarrow D_1 \triangle D_2 = \varnothing$$

PREUVE.

1. Si on a $D_1 = RD_1$ alors on a $D_1 \subseteq RD_1$. Montrons maintenant que $D_1 \subseteq RD_1$ implique $D_1 \subseteq ([Q_R^2] \setminus D_2)$, c.à.d. que $RD_1 \subseteq ([Q_R^2] \setminus D_2)$.
   Considérons un état $(q_1, q_2) \in RD_1$. La définition 5.9 donne qu'il existe (e_1, e_2) **when** $sp \in Syn_R^c$ tel que $e_1 \in E_R^1 \cup \{-\}$ et $e_2 \in E_R^2$. La définition 5.5 donne que (e_1, e_2) **when** $sp \in [Syn_R^c]_{SR^2}$. Puisque $q_2 \xrightarrow{e_2} \in T_R^2$ et $(q_1, q_2) \models sp$, la définition 5.6 donne $(q_1, q_2) \xrightarrow{(e_1,e_2)} \in [T_R^2]$, c.à.d. $(q_1, q_2) \in [Q_R^2]$ et $(q_1, q_2) \notin D_2$.
2. La démonstration de $D_2 = RD_2 \Rightarrow\ D_2 \subseteq ([Q_R^1] \setminus D_1)$ est identique.
3. Montrons maintenant que $D_1 \subseteq ([Q_R^2] \setminus D_2)\ \wedge\ D_2 \subseteq ([Q_R^1] \setminus D_1)\ \Rightarrow\ D_1 \triangle D_2 = \varnothing$.
   Par définition, $D_1 \triangle D_2 = (D_1 \cap D_2) \cup (D_1 \setminus [Q_R^2]) \cup (D_2 \setminus [Q_R^1])$. Pour que $D_1 \triangle D_2$ soit vide, il faut que $D_1 \cap D_2$ soit vide, que $D_1 \setminus [Q_R^2]$ soit vide et que $D_2 \setminus [Q_R^1]$ soit vide.
   - Comme $D_1 \subseteq ([Q_R^2] \setminus D_2)$, on a $D_1 \cap D_2 = \varnothing$.
   - Comme $D_1 \subseteq ([Q_R^2] \setminus D_2)$, on a $D_1 \setminus [Q_R^2] = \varnothing$.
   - Comme $D_2 \subseteq ([Q_R^1] \setminus D_1)$, on a $D_2 \setminus [Q_R^1] = \varnothing$.

□

5.2.6 Raffinement d'un système à composants

Nous avons étudié jusqu'ici comment obtenir automatiquement une décomposition à partir d'un système à composants synchronisés – abstraits ou raffinés – et comment réaliser le renommage par τ pour le cas particulier des composants sous-contexte. Nous avons aussi proposé une solution compositionnelle pour vérifier l'absence de τ-cycles dans le système complet raffiné et une manière efficace de s'assurer de l'absence de nouveaux blocages pour le raffinement du système à composants complet. Nous exprimons maintenant un théorème à propos de la vérification compositionnelle du raffinement d'un système à composants synchronisés. Ce théorème est une adaptation au cas particulier des systèmes à composants du théorème 4.2 à propos du raffinement par décomposition donné chapitre 4.

Théorème 5.2 (Raffinement d'un système à composants)
Soient (SA^1, SA^2, Syn_A^c) et (SR^1, SR^2, Syn_R^c) deux systèmes à composants.

$$\frac{\begin{array}{ll} 1. & \neg\, div^\tau(\mathcal{T}_{SA_1}(SR^1)),\ \neg\, div^\tau(\mathcal{T}_{SA_2}(SR^2)), \\ 2. & \mathcal{T}_{SA_1,SA_2}([SR^1]) \sqsubseteq_{\rho_w}^{D_1} [SA^1],\ D_1 = RD_1, \\ 3. & \mathcal{T}_{SA_1,SA_2}([SR^2]) \sqsubseteq_{\rho_w}^{D_2} [SA^2],\ D_2 = RD_2 \end{array}}{\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c) \sqsubseteq_\eta (SA^1, SA^2, Syn_A^c)}$$

PREUVE. Les hypothèses $\neg\, div^\tau(\mathcal{T}_{SA_1}(SR^1))$ et $\neg\, div^\tau(\mathcal{T}_{SA_2}(SR^2))$ donnent, grâce au lemme 5.1, que $\neg\, div^\tau(\mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn_R^c))$.

D'après le lemme 5.3, nous avons $D_1 \triangle D_2 = \varnothing$.
Puisque $\mathcal{T}_{SA_1,SA_2}([SR^1]) \cup \mathcal{T}_{SA_1,SA_2}([SR^2]) = \mathcal{T}_{SA_1,SA_2}(SR^1, SR^2, Syn^c_R)$ (Lemme 5.2), nous pouvons réutiliser le théorème 4.2 pour conclure.

□

Exemple. Nous utilisons le théorème 5.2 pour vérifier de manière compositionnelle que le système à composants raffinés $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ est bien un raffinement correct du système à composants abstraits $(Arm_A, Clip_A, Elev_A, Syn^c_A)$.

1. Comme $\mathcal{T}(Arm_R) \sqsubseteq_\eta Arm_A$, $\mathcal{T}(Clip_R) \sqsubseteq_\eta Clip_A$ et que $\mathcal{T}(Elev_R) \sqsubseteq_\eta Elev_A$, nous avons $\neg\, div^\tau(\mathcal{T}(Arm_R))$, $\neg\, div^\tau(\mathcal{T}(Clip_R))$ et $\neg\, div^\tau(\mathcal{T}(Elev_R))$.
2. Nous pouvons construire successivement les composants sous-contexte raffinés et abstraits correspondant à chacun des composants simples, et en étudier le raffinement affaibli :
   (a) Nous construisons les composants sous-contexte $\mathcal{T}([Arm_R])$ et $[Arm_A]$ (voir figures 5.12(a) et 5.6(a)) et nous en étudions le raffinement affaibli. Nous obtenons $\mathcal{T}([Arm_R]) \sqsubseteq^{D_A}_{\rho_w} [Arm_A]$ avec $D_A = \{q_e\}$. Nous calculons l'ensemble des états de blocages réductibles $RD_A = \{q_e\}$.
   (b) Nous construisons les composants sous-contexte $\mathcal{T}([Clip_R])$ et $[Clip_A]$ (voir figures 5.12(b) et 5.6(b)) et nous en étudions le raffinement affaibli. Nous obtenons $\mathcal{T}([Clip_R]) \sqsubseteq^{D_C}_{\rho_w} [Clip_A]$ avec $D_C = \varnothing$.
   (c) Nous construisons les composants sous-contexte $\mathcal{T}([Elev_R])$ et $[Elev_A]$ (voir figures 5.12(c) et 5.6(c)) et nous en étudions le raffinement affaibli. Nous obtenons $\mathcal{T}([Elev_R]) \sqsubseteq^{D_E}_{\rho_w} [Elev_A]$ avec $D_E = \varnothing$.
3. Nous pouvons conclure grâce au théorème 5.2 que

$$\mathcal{T}(Arm_R, Clip_R, Elev_R, Syn^c_R) \sqsubseteq_\eta (Arm_A, Clip_A, Elev_A, Syn^c_A)$$

   sans avoir eu besoin de construire ni $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ ni $(Arm_A, Clip_A, Elev_A, Syn^c_A)$ pour réaliser cette vérification.

5.3 Systèmes à composants et vérification de propriétés

Un système à composants est le résultat d'un produit synchronisé particulier de plusieurs systèmes de transitions (les composants) sous un ensemble de synchronisations contraintes. Dans cette section, nous allons montrer que certaines propriétés peuvent se vérifier de manière compositionnelle : quand on se place dans le cadre des systèmes à composants synchronisés, la vérification de ces propriétés pourra être effectuée simultanément à la vérification compositionnelle du raffinement (voir section 5.2).

– Certaines propriétés vérifiées *localement* pour un composant donné, seront préservées au niveau du système à composants.

– D'autres propriétés – *globales* – du système à composants complet pourront être vérifiées en réutilisant la décomposition issue des composants sous-contexte et les résultats de la section 4.3 à propos de la vérification de propriétés dans le cadre de la décomposition d'un système.

Remarque. Soit (S^1, S^2, Syn^c) un système à composants. On parlera de *propriété locale* pour désigner une propriété (LTL ou non) exprimée uniquement sur les variables d'un des composants – c.à.d. sur V^1 ou bien sur V^2 – alors qu'une *propriété globale* désignera une propriété faisant intervenir des variables des deux composants – c.a.d. des variables de $V^1 \cup V^2$.

Comme nous nous plaçons toujours dans le cadre d'un développement par raffinement, nous bénéficions de la préservation (voir section 3.4.1) et de la reformulation (voir section 3.4.2) des propriétés vérifiées au niveau le plus abstrait.

Dans la section suivante, nous nous intéressons particulièrement aux propriétés d'invariance locales, puis nous généralisons nos résultats à toutes les propriétés LTL de sûreté. Finalement, nous montrons que les résultats de la section 4.3 peuvent être réutilisés dans le cadre des systèmes à composants.

5.3.1 Systèmes à composants et propriétés d'invariance locales

Les résultats de cette section sont simples et peu surprenants. Une propriété d'invariance satisfaite par l'un des composants est aussi satisfaite par le système à composants synchronisés complet. L'idée est la suivante : un invariant local ne concerne que les variables d'un des composants. La seule manière pour modifier les valeurs de ces variables – dans le composant ou dans le système à composants – c'est d'effectuer une transition du composant. Ce qui nous donne le lemme suivant à propos de la préservation d'une propriété d'invariance locale sur le système à composants synchronisés.

Lemme 5.4 (Invariant local)
Soient (SA^1, SA^2, Syn^c_A) *un système à composants et* $\Box sp_{A1}$ *une propriété LTL telle que* $sp_{A1} \in SP_{V^1_A}$.

$$\frac{SA^1 \models \Box sp_{A1}}{(SA^1, SA^2, Syn^c_A) \models \Box sp_{A1}}$$

PREUVE. (par contradiction) Supposons que le lemme précédent soit faux. Il existe donc un état (q_1, q_2) du système à composants (SA^1, SA^2, Syn^c_A) tel que $(q_1, q_2) \not\models sp_{A1}$. Comme $sp_{A1} \in SP_{V^1_A}$ concerne uniquement les variables de SA^1, la définition 3.2 nous donne que $q_1 \not\models sp_{A1}$, ce qui contredit l'hypothèse $SA^1 \models \Box sp_{A1}$.

□

Dans le cas des systèmes à composants synchronisés, S^1 (resp. S^2) ne peut modifier que les variables de V^1_A (resp. V^2_A). De plus, un invariant local ne concerne que ces variables. Pour ces raisons, nous pouvons donner le lemme suivant.

Lemme 5.5 (Composition des invariants locaux)
Soient (SA^1, SA^2, Syn_A^c) *un système à composants et* $\Box sp_{A1}$ *et* $\Box sp_{A2}$ *deux propriétés LTL telles que* $sp_{A1} \in SP_{V_A^1}$ *et* $sp_{A2} \in SP_{V_A^2}$.

$$\frac{SA^1 \models \Box sp_{A1},\ SA^2 \models \Box sp_{A2}}{(SA^1, SA^2, Syn_A^c) \models \Box(sp_{A1} \wedge sp_{A2})}$$

PREUVE. Evidente grâce au lemme 5.4 et à la règle de $\wedge$-introduction du calcul des séquents. □

Dans les sections 3.4.1 et 3.4.2, nous avons indiqué que la relation de raffinement utilisée préservait les propriétés vérifiées par le système abstrait au niveau du système raffiné et simplifiait la vérification de certaines propriétés du système raffiné, grâce à la reformulation. La proposition 5.1 indique comment combiner vérification compositionnelle du raffinement d'un système à composants avec la vérification locale de propriétés d'invariance.

Proposition 5.1 (Préservation des invariants locaux)
Soient (SA^1, SA^2, Syn_A^c) *et* (SR^1, SR^2, Syn_R^c) *deux systèmes à composants. Soient* $\Box sp_{A1}$ *et* $\Box sp_{A2}$ *deux propriétés LTL telles que* $sp_{A1} \in SP_{V_A^1}$ *et* $sp_{A2} \in SP_{V_A^2}$.

$$\textit{Si} \begin{cases} \mathcal{T}_{SA^1,SA^2}(SR^1, SR^2, Syn_R^c) \sqsubseteq_\eta (SA^1, SA^2, Syn_A^c) \\ SA^1 \models \Box sp_{A1} \\ SA^2 \models \Box sp_{A2} \end{cases}$$

$$\textit{alors } (SR^1, SR^2, Syn_R^c) \models_{pre} \Box(sp_{A1} \wedge sp_{A2})$$

PREUVE. Conséquence directe du théorème 3.2 et du lemme 5.5. □

La vérification compositionnelle du raffinement et la vérification de propriétés d'invariance abstraites locales pourraient simplifier la vérification d'une propriété d'invariance raffinée pour le système à composants raffinés, grâce à la reformulation de propriétés d'invariance (cf. proposition 3.1).

$$\left.\begin{array}{l} SA^1 \models \Box sp_{A1} \\ SA^2 \models \Box sp_{A2} \\ sp_{A1} \wedge gp \Rightarrow sp_{R1} \\ sp_{A2} \wedge gp \Rightarrow sp_{R2} \\ (SR^1, SR^2, Syn_R^c) \sqsubseteq_\eta (SA^1, SA^2, Syn_A^c) \end{array}\right\} \Rightarrow (SR^1, SR^2, Syn_R^c) \models \Box(sp_{R1} \wedge sp_{R2})$$

5.3.2 Systèmes à composants et propriétés LTL

Dans la section précédente, nous avons indiqué qu'une propriété d'invariance $\Box sp_1$ locale satisfaite par un composant SA^1 était préservée par le système à composants (SA^1, SA^2, Syn_A^c).

Avons-nous le même résultat lorsqu'il s'agit d'une propriété LTL quelconque ? D'une manière générale, nous ne pouvons pas affirmer qu'une propriété LTL locale à un composant sera toujours satisfaite par le système à composants synchronisés.
- L'ensemble de synchronisations contraintes Syn_A^c peut, dans certains cas, supprimer certains comportements des composants (par le biais des conditions de synchronisations **when**) : la ou les transitions qui garantissaient la satisfaction de la propriété par le composant ne se retrouve plus dans le système complet.
- Puisque le produit synchronisé combine les comportements de plusieurs composants, il est fortement possible qu'il vienne intercaler une ou plusieurs transitions de SA^2 entre deux transitions dont la succession directe garantissait la satisfaction de la propriété par SA^1.

Etudier si une propriété LTL satisfaite par SA^1 est aussi satisfaite par (SA^1, SA^2, Syn_A^c) se ramène à étudier si SA^1 "simule" (SA^1, SA^2, Syn_A^c). En fait, nous allons montrer que SA^1 τ-simule (SA^1, SA^2, Syn_A^c) au sens de la définition 2.5 donnée section 2.2. Comme nous l'avons déjà précisé dans la section 3.4.1, si $(SA^1, SA^2, Syn_A^c) \preceq_\varrho SA^1$ et si SA^1 satisfait une propriété de sûreté ϕ alors ϕ sera préservée pour (SA^1, SA^2, Syn_A^c). Pour une formule LTL quelconque, la partie sûreté de ϕ sera également préservée.

La relation de collage est alors donnée par l'égalité des variables de V_A^1 dans SA^1 et dans (SA^1, SA^2, Syn_A^c), c.à.d. que quelque soit l'état (q_1, q_2) de (SA^1, SA^2, Syn_A^c), il sera collé à q_1 :

$$\forall (q_1, q_2).\ (l^1(q_1) \subseteq l((q_1, q_2)) \Rightarrow (q_1, q_2)\ \mu\ q_1)$$

Il est aussi nécessaire de préciser comment renommer les transitions de (SA^1, SA^2, Syn_A^c) pour se ramener uniquement à des transitions qui concernent SA_1 (ou bien à τ) :
- Les étiquettes de transitions de la forme $(e_1, -)$ ou (e_1, e_2) seront renommées par e_1.
- Les étiquettes de transitions de la forme $(-, e_2)$ seront renommées par τ.

Lemme 5.6 (Système à composants simulé par un composant)
Soit (SA^1, SA^2, Syn_A^c) un système à composants. Alors

$$(SA^1, SA^2, Syn_A^c) \preceq_\varrho SA^1$$

PREUVE. Montrons que, quelque soit la transition de (SA^1, SA^2, Syn_A^c), soit elle est simulée dans SA^1, soit elle est τ-simulée dans SA^1. Cela dépend de l'étiquette de la transition.
- $(q_1, q_2) \overset{(e_1,-)\backslash e_1}{\longrightarrow} (q_1', q_2)$. Par la définition 5.2 du produit synchronisé contraint, il existe $q_1 \overset{e_1}{\rightarrow} q_1' \in T_A^1$, $(q_1, q_2)\ \mu\ q_1$ et $(q_1', q_2)\ \mu\ q_1'$.
- $(q_1, q_2) \overset{(e_1,e_2)\backslash e_1}{\longrightarrow} (q_1', q_2)$. Identique.
- $(q_1, q_2) \overset{(-,e_2)\backslash \tau}{\longrightarrow} (q_1, q_2')$. $(q_1, q_2)\ \mu\ q_1$ et $(q_1, q_2')\ \mu\ q_1$.

□

Le résultat précédent est un résultat classique adapté à notre définition particulière de la composition. Clarke, par exemple, énonce un résultat similaire dans [CGP00] à propos d'un produit synchrone.

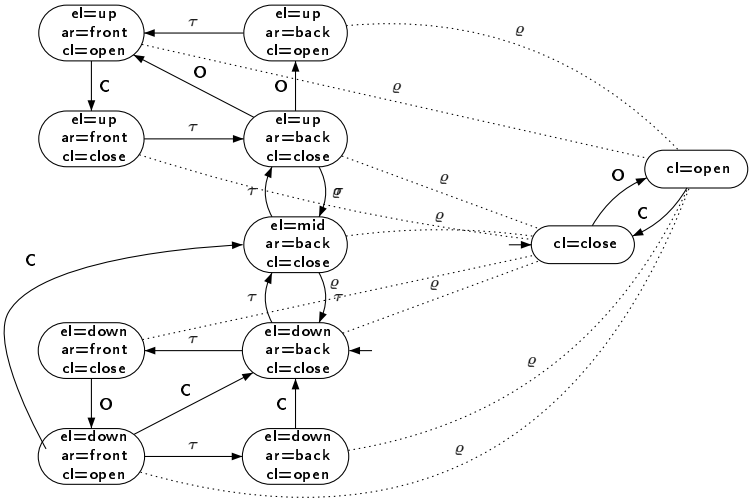


FIG. 5.13: $(Arm_A, Clip_A, Elev_A, Syn^c_A)$ τ-simulé par $Clip_A$

Exemple. Considérons le système à composants $(Arm_A, Clip_A, Elev_A, Syn^c_A)$ donné section 5.1.2. On peut facilement montrer que le composant $Clip_A$ τ-simule $(Arm_A, Clip_A, Elev_A, Syn^c_A)$. Voir figure 5.13.

Le résultat précédent donne immédiatement un autre résultat à propos de la préservation pour un système à composants synchronisés des propriétés LTL satisfaites localement.

Lemme 5.7 (Préservation des propriétés LTL d'un composant)
Soit (SA^1, SA^2, Syn^c_A) *un système à composants.*
- *Si* $\phi \in LTL_{saf}$ *et si* $SA^1 \models \phi$ *alors* $(SA^1, SA^2, Syn^c_A) \models_{pre} \phi$
- *Si* $\phi \in LTL$ *et si* $SA^1 \models \phi$ *alors il existe* $\phi_s \in LTL_{saf}$ *et* $\phi_l \in LTL_{liv}$ *telles que* $\phi \equiv \phi_s \wedge \phi_l$ *et* $(SA^1, SA^2, Syn^c_A) \models_{pre} \phi_s$

PREUVE. Immédiate grâce au lemme 5.6.

□

La table 5.14 présente quelques schémas de propriétés LTL pour lesquelles on a le résultat de préservation précédent. Par exemple pour la propriété $sp_1 \mathcal{U} sp'_1 \equiv sp_1 \mathcal{W} sp'_1 \wedge \Diamond sp'_1$ (schéma 5.14.3), la partie sûreté uniquement est préservée pour le système à composants synchronisés.

Bien entendu, le théorème de vérification compositionnelle du raffinement et le lemme 5.7 peuvent être composés. Une propriété LTL satisfaite localement par un composant, sera préservée par la τ-simulation – ou tout du moins la partie sûreté de la propriété – pour le système à composants abstraits, puis préservée par la relation de raffinement pour le système à composants raffinés.

Proposition 5.2 (Préservation des propriétés LTL locales par le raffinement)
Soient (SA^1, SA^2, Syn^c_A) *et* (SR^1, SR^2, Syn^c_R) *deux systèmes à composants. Soit* $\phi \in LTL$

	$SA^1 \models$	$(SA^1, SA^2, Syn_A^c) \models_{pre}$
Schéma 5.14.1	$\Box sp_1$	$\Box sp_1$
Schéma 5.14.2	$sp_1 \mathcal{W} sp'_1$	$sp_1 \mathcal{W} sp'_1$
Schéma 5.14.3	$sp_1 \mathcal{U} sp'_1$	$sp_1 \mathcal{W} sp'_1$
Schéma 5.14.4	$sp_1 \Rightarrow \bigcirc sp'_1$	$sp_1 \Rightarrow sp_1 \mathcal{W} sp'_1$
Schéma 5.14.5	$sp_1 \Rightarrow \Diamond sp'_1$	$sp_1 \Rightarrow sp_1 \mathcal{W} sp'_1$

FIG. 5.14: Schémas de préservation de certaines propriétés LTL locales

une formule LTL quelconque.

$$Si \begin{cases} \mathcal{T}_{SA^1,SA^2}(SR^1, SR^2, Syn_R^c) \sqsubseteq_\eta (SA^1, SA^2, Syn_A^c) \\ SA^1 \models \phi \\ \phi \equiv \phi_s \wedge \phi_l \text{ avec } \phi_s \in LTL_{saf} \text{ et } \phi_l \in LTL_{liv} \end{cases} \quad alors\ (SR^1, SR^2, Syn_R^c) \models_{pre} \phi_s$$

PREUVE. Evidente grâce au lemme 5.7.

□

Exemple. Considérons la propriété (3.3) indiquant que la pince ne reste pas indéfiniment ouverte :

$$\Box((cl = open) \Rightarrow \Diamond(cl = close))$$

Cette propriété est satisfaite par le composant $Clip_A$. Comme le système à composants $(Arm_A, Clip_A, Elev_A, Syn_A^c)$ est τ-simulé par le composant $Clip_A$, on a directement que $(Arm_A, Clip_A, Elev_A, Syn_A^c)$ satisfait aussi la propriété (3.3) (lemme 5.7).
Bien entendu, la propriété (3.3) sera préservée par le raffinement : le système à composants raffinés $(Arm_R, Clip_R, Elev_R, Syn_R^c)$ satisfait par préservation la propriété (3.3) (Proposition 5.2).

5.3.3 Systèmes à composants et propriétés globales

Comme nous l'avons indiqué dans la section 5.1.1, le théorème 5.1 nous permet d'obtenir une décomposition d'un système à composants afin de vérifier de manière compositionnelle le raffinement du système à composants.

$$(SA^1, SA^2, Syn_A^c) = [SA^1]_{Syn_A^c}^{SA^2} \cup [SA^2]_{Syn_A^c}^{SA^1}$$

Dans le chapitre 4, et plus particulièrement la section 4.3, nous avons donné certains résultats quant à la vérification compositionnelle de certaines propriétés dans le cas de sous-systèmes issus d'une décomposition. Les résultats à propos de la vérification compositionnelle d'invariants, d'invariants dynamiques, de la non-satisfaction des propriétés de sûreté ou de l'atteignabilité d'un état donné sont applicables directement sur les composants sous-contexte.

– Invariant global et composants sous-contexte (voir proposition 4.1, section 4.3) :

$$\left.\begin{array}{l} [SA^1]^{SA^2}_{Syn^c_A} \models \Box sp_A \\ [SA^2]^{SA^1}_{Syn^c_A} \models \Box sp_A \end{array}\right\} \Rightarrow (SA^1, SA^2, Syn^c_A) \models \Box sp_A$$

– Invariant dynamique et composants sous-contexte (voir proposition 4.3, section 4.3) :

$$\left.\begin{array}{l} Dynamics([SA^1]^{SA^2}_{Syn^c_A}, sp, sp') \\ Dynamics([SA^2]^{SA^1}_{Syn^c_A}, sp, sp') \end{array}\right\} \Rightarrow Dynamics((SA^1, SA^2, Syn^c_A), sp, sp')$$

– Non-satisfaction et composant sous-contexte (voir proposition 4.4, section 4.3) :

$$[SA^1]^{SA^2}_{Syn^c_A} \not\models \phi \Rightarrow (SA^1, SA^2, Syn^c_A) \not\models \phi$$

– Atteignabilité et composant sous-contexte (voir proposition 4.5, section 4.3) :

$$reachable(q_c, [SA^1]^{SA^2}_{Syn^c_A}) \Rightarrow reachable(q_c, (SA^1, SA^2, Syn^c_A))$$

Bien entendu, ces vérifications sont compatibles avec la vérification compositionnelle du raffinement. Dans tous les cas, il est nécessaire de construire les composants sous-contexte pour pouvoir conclure. La vérification de certaines propriétés globales peut donc être réalisée conjointement à celle du raffinement. Et, comme on aura le raffinement, les propriétés vérifiées seront automatiquement préservées sur le système à composants raffinés.

5.4 Conclusion

Dans ce chapitre, nous avons étudié le raffinement et la vérification de propriétés dans le cadre plus précis des *systèmes à composants synchronisés*.

Nous avons défini une extension du produit synchronisé d'Arnold et Nivat : les *systèmes à composants synchronisés*. Comme dans le cas du produit synchronisé classique, les interactions entre les différents composants sont exprimées par un ensemble de synchronisations, mais dans notre cas, nous contraignons l'activation des transitions synchronisées par des conditions d'activations.

Néanmoins, nous souhaitons pouvoir étudier le système à composants sans calculer ce produit, mais en focalisant notre analyse sur les composants du système pris les uns après les autres : si l'on considère chaque composant l'un après l'autre, il s'agit d'un composant "hors contexte", qui ne partage aucune variable avec les autres composants et dont le comportement est complètement défini indépendamment des autres composants ou de l'ensemble de synchronisations. Afin d'étudier composant par composant le système complet, nous établissons pour cela un lien avec l'opérateur de décomposition du chapitre précédent : nous définissons alors les composants *sous-contexte* en restreignant l'ensemble de synchronisations aux comportements du composant considéré et nous montrons ensuite que les

composants sous-contexte forment une *décomposition* du système à composants, obtenue automatiquement, sans avoir besoin de considérer le système dans son ensemble.

Comme, grâce aux composants sous-contexte, nous obtenons une décomposition du système à composants, nous pouvons bénéficier des résultats du chapitre 4 à propos de la vérification par décomposition du raffinement et de certaines propriétés.

Nous montrons ensuite que l'étude compositionnelle du raffinement d'un système à composants synchronisés se ramène, grâce aux composants sous-contexte, à étudier un raffinement par décomposition. Les limitations du raffinement par décomposition n'ont plus lieu d'être dans le cadre des systèmes à composants.

- La non τ-divergence dans le système à composants synchronisés raffinés est vérifiée de manière compositionnelle : vérifier l'absence de τ-cycles pour chacun des composants simples permet automatiquement de vérifier l'absence de τ-cycles pour le système à composants.
- La réduction des ensembles de "faux" blocages se ramène à étudier uniquement l'ensemble raffiné de synchronisations. Pour chacun des composants sous-contexte, nous calculons un ensemble de blocages réductibles, uniquement à partir des composants simples et de l'ensemble de synchronisations.

En adaptant le raffinement par décomposition, le raffinement du système à composants synchronisés complet est alors assuré en étudiant successivement le raffinement affaibli de chacun des composants sous-contexte, puis en calculant l'ensemble des blocages réductibles pour chaque composant sous-contexte.

Nous étudions pour finir la vérification compositionnelle de certaines propriétés LTL, dans le cadre des systèmes à composants synchronisés. Nous avons montré que les propriétés LTL de sûreté locales à l'un des composants – en particulier les propriétés d'invariance – sont préservées automatiquement au niveau du système à composants complet. Nous indiquons aussi que, grâce aux composants sous-contexte, nous pouvons bénéficier des résultats du chapitre précédent à propos de la vérification par décomposition.

Chapitre 6

Une implantation : SynCo

Dans ce chapitre, nous allons présenter le *prototype* que nous avons implanté afin de valider la démarche de vérification compositionnelle du raffinement des systèmes à composants.

L'outil que nous avons développé s'appelle SynCo[1] et il a fait l'objet de plusieurs présentations [KL, Lan04b]. Il s'agit d'un *prototype* qui met en oeuvre la démarche présentée dans le chapitre précédent à propos du raffinement des systèmes à composants synchronisés, et, en particulier, le théorème 5.2. SynCo construit les composants sous-contexte associés à chacun des composants du système, puis vérifie leurs raffinements affaiblis et finalement en déduit la correction du raffinement du système à composants complet ou non.

Bien qu'il s'agisse d'un prototype, nous avons pu réaliser plusieurs études de cas que nous présentons dans la section 6.2 : l'une porte sur le module *essuyage avant* d'une automobile et une autre concerne le porte-monnaie électronique *CEPS*.

Remarque. Le développement de SynCo et les études de cas associées ont été réalisés en collaboration avec plusieurs étudiants de Master 2 Informatique lors de leurs projets de fin d'étude [CL03, HD04, LT04, Col04a].

6.1 Principe et fonctionnement de SynCo

SynCo est une application développée principalement en Java qui permet de spécifier des systèmes sous la forme de systèmes à composants synchronisés (voir définition 5.3), puis de vérifier leur raffinement. Elle met en oeuvre la démarche présentée dans le chapitre précédent, en appliquant le théorème 5.2 à propos de la vérification compositionnelle du raffinement des systèmes à composants.

6.1.1 Spécification d'un système à composants synchronisés

Afin de spécifier les composants, nous avons choisi d'adapter la syntaxe des *Fair Transitions Systems* (FTSs) à notre propre usage. Le langage FTS est l'un des langages d'entrée de STeP (The Stanford TEmporal Prover), un prouveur de théorèmes développé à l'université de Stanford [MBB+95, BBC+98]. Le langage FTS permet de décrire des ST2E. La figure 6.1 donne la grammaire FTS implantée dans SynCo.

Pour décrire un ST2E en FTS, on déclare d'abord certains types énumérés (**type**), puis les variables manipulées par le ST2E (**local**). On doit spécifier une condition initiale portant sur les variables du ST2E qui permettra d'établir le ou les états initiaux du système (**Initially**). On donne ensuite la liste des transitions[2] (**Transition**) du ST2E. Chacune des transitions est composée de deux clauses, l'une indiquant la condition d'activation de la transition (**enable**) et l'autre donnant une liste d'affectations représentant les modifications de variables effectuées lors du franchissement de la transition (**assign**).

[1] pour SYNchronized COmponent-based systems.

[2] Les transitions FTS "correspondent" à des événements en B. Elles donnent lieu à un ensemble de transitions dans le ST2E correspondant.

transition-system	::=	`Transition System` [*types*] *decls* [*invariant*] *initially transitions*
types	::=	*type* [*types*]
type	::=	`type` *id* `=` `{` *ids* `}`
decls	::=	*decl* [*decls*]
decl	::=	*mode ids* `:` *type*
mode	::=	`local` | `in` | `out`
type	::=	`bool` | `int` | *id*
invariant	::=	`Invariant` *expn*
initially	::=	`Initially` *expn*
transitions	::=	*transition* [*transitions*]
transition	::=	`Transition` *id* `:` *enable* `;` *assign*
enable	::=	`enable` *expn*
assign	::=	`assign` *assignments*
assignments	::=	*assignment* [`,` *assignments*]
assignment	::=	*id* `:=` *expn*
expn	::=	*id* | *int* | *bool* | `(` *expn* `)`
	|	*expn infix expn* | *prefix expn*
	|	`if` *expn* `then` *expn* `end` *expn*
infix	::=	/\ | \/ | `<->`
	|	`+` | `*` | `-` | `/` | `mod` | `div`
	|	`=` | `!=` | `<` | `>` | `>=` | `<=`
prefix	::=	`!` | `-`
bool	::=	`true` | `false`
int	::=	INTEGER
ids	::=	*id* [`,` *ids*]
id	::=	WORD

FIG. 6.1: Grammaire FTS implantée dans SynCo

synchronisations	::=	*synchronisation* [`,` *synchronisations*]
synchronisation	::=	*id* `when` *expn*
		| `(` *ids* `)` `when` *expn*

FIG. 6.2: Grammaire des ensembles de synchronisations dans SynCo

Transition System

type CLIP = {open, close}
type ARM = {front, back}
type ELEVATOR = {up, mid, down}

local cl : CLIP
local ar : ARM
local el : ELEVATOR

Initially (cl = close ∧ ar = back ∧ el = down)

Transition O :
enable (cl = close ∧ ar = back ∧ el = up)
∨ (cl = close ∧ ar = front ∧ el = down) ;
assign cl := open

Transition C :
enable (cl = open ∧ ar = back ∧ el = down)
∨ (cl = open ∧ ar = front ∧ el = up) ;
assign cl := close

Transition F :
enable (ar = back ∧ cl = close ∧ el = down)
∨ (ar = back ∧ cl = open ∧ el = up) ;
assign ar := front

Transition B :
enable (ar = front ∧ cl = close ∧ el = up)
∨ (ar = front ∧ cl = open ∧ el = down) ;
assign ar := back

Transition BC :
enable (ar = front ∧ cl = open ∧ el = down) ;
assign ar := back, cl := close

Transition OF :
enable (ar = back ∧ cl = close ∧ el = up) ;
assign ar := front, cl := open

Transition BCU1 :
enable (ar = front ∧ cl = open ∧ el = down) ;
assign ar := back, cl := close, el := mid

Transition U1 :
enable (el = down ∧ ar = back ∧ cl=close) ;
assign el := mid

Transition D2 :
enable (el = mid ∧ ar = back ∧ cl=close) ;
assign el := down

Transition U2 :
enable (el = mid ∧ ar = back ∧ cl=close) ;
assign el := up

Transition D1 :
enable (el = up ∧ ar = back ∧ cl=close) ;
assign el := mid

FIG. 6.3: Spécification FTS du ST2E *Robot*

Exemple. La figure 6.3 montre la spécification FTS du ST2E *Robot* présenté dans les sections 1.1.4 et 2.2.4. Trois types énumérés ARM= {front, back}, ELEVATOR = {up, mid, down} et CLIP = {open, close} sont déclarés ainsi que trois variables *el*, *ar* et *cl*, afin de spécifier respectivement les positions horizontale et verticale du bras et l'état de la pince. La condition d'initialisation caractérise l'état initial de *Robot* et les différentes transitions spécifiées modélisent les comportements souhaités du bras robotisé :

- O et C représentent respectivement l'ouverture et la fermeture de la pince,
- F et B représentent les mouvements horizontaux du bras robotisé,
- BC représente le recul du bras simultanément à la fermeture de la pince,
- OF représente l'avancée du bras simultanément à l'ouverture de la pince,
- $BCU1$ représente le recul du bras simultanément à la fermeture de la pince et à la montée, et
- $U1$, $D2$, $U2$ et $D1$ représentent les mouvements verticaux du bras.

Nous avons aussi étendu la grammaire des FTS afin de pouvoir spécifier, directement avec le composant, une propriété d'invariance locale au composant (**Invariant**).

Nous avons choisi de spécifier l'ensemble de synchronisations Syn^c décrivant les interactions entre les composants à l'aide d'une grammaire illustrée par la figure 6.2 qui correspond à la définition 5.1. Pour vérifier le raffinement, il faut aussi spécifier, pour chaque composant, un *prédicat de collage*, c.à.d. un prédicat qui traduit la relation qu'il y a entre les variables définies dans le composant abstrait et celles définies dans le composant raffiné.

Un système à composants synchronisés est un n-uplet $(S^1, S^2, \ldots, S^n, Syn^c)$ formé d'une liste de composants $S^1, S^2, \ldots, S^n$ et d'un ensemble de synchronisations contraintes Syn^c (voir définition 5.3).

L'interface graphique de SynCo permet de spécifier un système à composants synchronisés, au niveau abstrait et au niveau raffiné : le système à composants est vu comme un nouveau projet créé sous SynCo dont les composants sont représentés de manière arborescente par des fichiers ".fts" correspondant à leurs spécifications FTS abstraites et raffinées. De manière similaire, les ensembles de synchronisations apparaissent sous la forme de fichiers ".syn" et les prédicats de collage, sous la forme de fichiers ".inv".

Actuellement, SynCo ne permet pas de spécifier plusieurs niveaux de raffinement : pour cela, il est nécessaire de créé sous SynCo, autant de projets que l'on a de niveaux à notre raffinement.

Nous pouvons modifier chacun des composants, des ensembles de synchronisations ou des prédicats de collage depuis la fenêtre centrale d'édition de SynCo (en bénéficiant alors de la coloration syntaxique et d'une aide à l'édition). On peut aussi visualiser très facilement le graphe correspondant au ST2E considéré, cela, si la taille du système le permet. Cette fonctionnalité utilise le package *Grappa* [ATTb] qui permet d'interfacer *Graphviz*, un outil de création, de placement et de visualisation de graphes [Gra, ATTa]. Il est à noter que les états apparaissent automatiquement numérotés de la façon suivante `<s0>`, `<s1>`, ...(voir figure 6.5(b)).

Transition System
type ARM = {front, back}
local ar : ARM
Initially ar = back
Transition F :
  enable ar = back ;
  assign ar := front
Transition B :
  enable ar = front ;
  assign ar := back

(a) Arm_A

Transition System
type CLIP = {open, close}
local cl : CLIP
Initially cl = close
Transition O :
  enable cl = close ;
  assign cl := open
Transition C :
  enable cl = open ;
  assign cl := close

(b) $Clip_A$

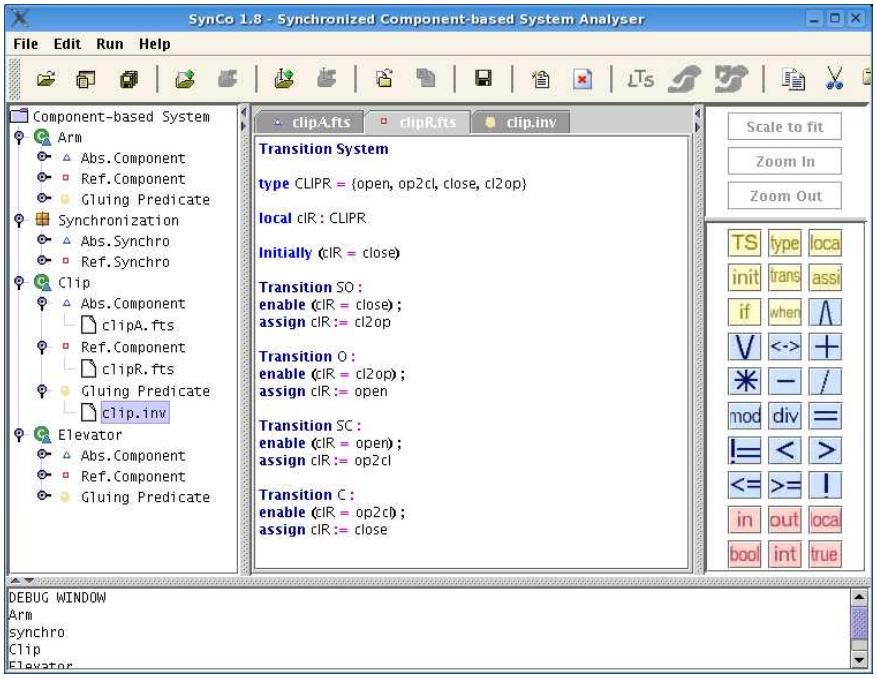
Transition System
type ELEVATOR = {up, mid, down}
local el : ELEVATOR
Initially el = down
Transition U1 :
  enable el = down ;
  assign el := mid
Transition D2 :
  enable el = mid ;
  assign el := down
Transition U2 :
  enable el = mid ;
  assign el := up
Transition D1 :
  enable el = up ;
  assign el := mid

(c) $Elev_A$

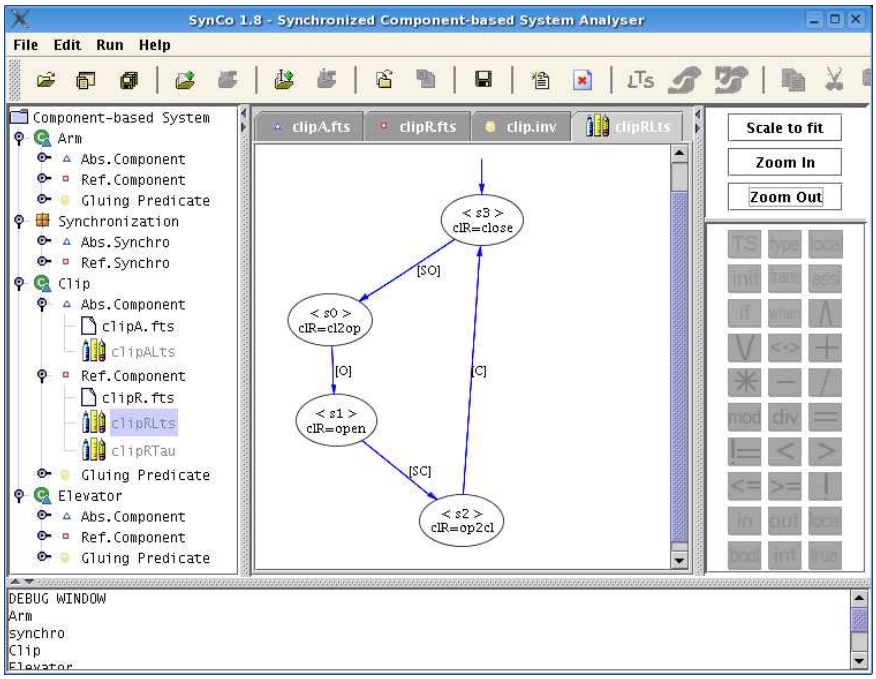
O **when** (ar=back $\wedge$ el=up) $\vee$ (ar=front $\wedge$ el=down),
C **when** (ar=front $\wedge$ el=up) $\vee$ (ar=back $\wedge$ el=down),
F **when** (cl=open $\wedge$ el=up) $\vee$ (cl=close $\wedge$ el=down),
B **when** (cl=close $\wedge$ el=up) $\vee$ (cl=open $\wedge$ el=down),
U1 **when** cl=close $\wedge$ ar=back,
U2 **when** cl=close $\wedge$ ar=back,
D1 **when** cl=close $\wedge$ ar=back,
D2 **when** cl=close $\wedge$ ar=back,
(F,O) **when** el=up,
(B,C) **when** el=down,
(B,C,U1) **when** true

(d) Syn^c_A

Fig. 6.4: Spécifications FTS du système à composants $(Arm_A, Clip_A, Elev_A, Syn^c_A)$



(a) Edition de la spécification FTS de $Clip_A$ sous SynCo



(b) Représentation graphique de $Clip_A$ sous SynCo

FIG. 6.5: Interface graphique de SynCo

Transition System
type ARMR = {front, 2back, back, 2front}
local arR : ARMR
Initially arR = back
Transition SF :
  enable arR = back ;
  assign arR := 2front
Transition F :
  enable arR = 2front ;
  assign arR := front
Transition SB :
  enable arR = front ;
  assign arR := 2back
Transition B :
  enable arR = 2back ;
  assign arR := back

(a) Arm_R

Transition System
type CLIPR = {open, 2close, close, 2open}
local clR : CLIPR
Initially clR = close
Transition SO :
  enable clR = close ;
  assign clR := 2open
Transition O :
  enable clR = 2open ;
  assign clR := open
Transition SC :
  enable clR = open ;
  assign clR := 2close
Transition C :
  enable clR = 2close ;
  assign clR := close

(b) $Clip_R$

(ar = front ⇔
  (arR = front ∨ arR =2back))
∧ (ar = back ⇔
  (arR = back ∨ arR = 2front))

(c) gp_{arm}

(el = up ⇔
  (elR = up ∨ elR =up2mid))
∧ (el = mid
  ⇔ (elR = mid ∨ elR = 2up ∨ elR = 2down))
∧ (el = down
  ⇔ (elR = down ∨ elR = do2mid))

(e) gp_{elev}

(cl = open ⇔
  (clR = open ∨ clR =2close))
∧ (cl = close ⇔
  (clR = close ∨ clR = 2open))

(d) gp_{clip}

Fig. 6.6: Spécifications FTS du système à composants $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ (1)

Transition System
type ELEVATORR =
{up, up2mid, mid, 2down, down, do2mid, 2up}
local elR : ELEVATORR
Initially elR = down
Transition SU1 :
enable elR = down ;
assign elR := do2mid
Transition U1 :
enable elR = do2mid ;
assign elR := mid
Transition SU2 :
enable elR = mid ;
assign elR := 2up
Transition U2 :
enable elR = 2up ;
assign elR := up
Transition SD1 :
enable elR = up ;
assign elR := up2mid
Transition D1 :
enable elR = up2mid ;
assign elR := mid
Transition SD2 :
enable elR = mid ;
assign elR := 2down
Transition D2 :
enable elR = 2down ;
assign elR := down

(a) $Elev_R$

SO **when** (arR=back ∧ elR=up)
∨ (arR=front ∧ elR=down),
O **when** (arR∈{back,2front} ∧ elR=up)
∨ (arR=front ∧ elR=down),
SC **when** (arR=front ∧ elR=up)
∨ (arR∈{back,2back} ∧ elR=down),
C **when** (arR=front ∧ elR=up)
∨ (arR=back ∧ elR=down),
SF **when** (clR∈{open,2open} ∧ elR=up)
∨ (clR=close ∧ elR=down),
F **when** (clR=open ∧ elR=up)
∨ (clR=close ∧ elR=down),
SB **when** (clR=close ∧ elR=up)
∨ (clR=open ∧ elR=down),
B **when** (clR=close ∧ elR=up)
∨ (clR∈{open,2close} ∧ elR=down),
SU1 **when** clR∈{close,2close} ∧ arR=back,
U1 **when** clR=close ∧ arR=back,
SU2 **when** clR=close ∧ arR=back,
U2 **when** clR=close ∧ arR=back,
SD1 **when** clR=close ∧ arR=back,
D1 **when** clR=close ∧ arR=back,
SD2 **when** clR=close ∧ arR=back,
D2 **when** clR=close ∧ arR=back,
(SF,SO) **when** elR=up,
(SF,O) **when** elR=up,
(F,O) **when** elR=up,
(SB,SC) **when** elR=down,
(B,SC) **when** elR=down,
(B,C) **when** elR∈{down,do2mid},
(SC,SU1) **when** arR=2back,
(SB,SC,SU1) **when** true,
(B,C,SU1) **when** true,
(B,C,U1) **when** true

(b) Syn^c_R

Fig. 6.7: Spécifications du système à composants $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ (2)

Exemple. Les spécifications FTS du système à composants $(Arm_A, Clip_A, Elev_A, Syn^c_A)$, correspondant au bras mobile robotisé présenté section 5.1.2, sont données figure 6.4. La figure 6.5(a) illustre l'édition du composant $Clip_A$ sous SynCo alors que la figure 6.5(b) montre une représentation graphique de $Clip_A$.
De manière similaire, les figures 6.6 et 6.7 donnent les spécifications FTS du système à composants raffinés $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ ainsi que les prédicats de collage gp_{arm}, gp_{clip} et gp_{elev}.

6.1.2 Vérification du raffinement d'un système simple

SynCo permet dans un premier temps de vérifier le raffinement entre deux ST2E simples. La relation de raffinement η est calculée par analyse locale : elle est synthétisée par énumération conjointe des ST2E SR et SA en s'assurant à chaque étape que toutes les conditions du raffinement sont vérifiées (voir section 2.2.3). Grâce à SynCo, il est possible de visualiser cette relation quand les systèmes étudiés ont une taille raisonnable, c.à.d. un nombre d'états et de transitions représentables graphiquement : dans ce cas, les deux graphes du système raffiné et du système abstrait sont dessinés côte-à-côte La relation de raffinement qui lie chaque état du système raffiné à un état du système abstrait est, elle aussi, dessinée (voir figure 6.8).

Exemple. La figure 6.8 montre – entre autre – la relation de raffinement η établie entre le composant raffiné $Elev_R$ et le composant abstrait $Elev_A$ grâce à SynCo. On est alors assuré que $Elev_R$ est bien un raffinement de $Elev_A$.

6.1.3 Vérification du raffinement d'un système à composants

En ce qui concerne la vérification du raffinement d'un système à composants synchronisés, le principe de fonctionnement de SynCo est le suivant (voir aussi figure 6.9) :

1. A partir des spécifications FTS des composants abstraits SA^1, SA^2, ..., SA^n et raffinés SR^1, SR^2, ..., SR^n et des ensembles de synchronisations Syn^c_A et Syn^c_R, SynCo construit successivement pour chaque couple de composants SR^i et SA^i, les composants sous-contexte $[SR^i]$ et $[SA^i]$ correspondants (définition 5.6, section 5.1.3).
2. Pour chaque composant sous-contexte raffiné $[SR^i]$ et abstrait $[SA^i]$, la relation de raffinement affaiblie ρ_w (définition 4.3, section 4.2.5) est synthétisée par analyse locale, de manière similaire au calcul de η. L'ensemble de blocages potentiels D_i est aussi calculé et l'on s'assure ainsi que $[SR^i] \sqsubseteq^{D_i}_{\rho_w} [SA^i]$.
3. Pour chaque D_i non vide, on calcule son ensemble de blocages réductibles RD_i (définition 5.9, section 5.2.5) et on s'assure que $D_i = RD_i$.
4. On répète les étapes 1., 2. et 3. pour chacun des composants SR^1, SR^2, ..., SR^n. On applique ensuite le théorème 5.2 et on en déduit si $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ est un raffinement correct de $(Arm_A, Clip_A, Elev_A, Syn^c_A)$.

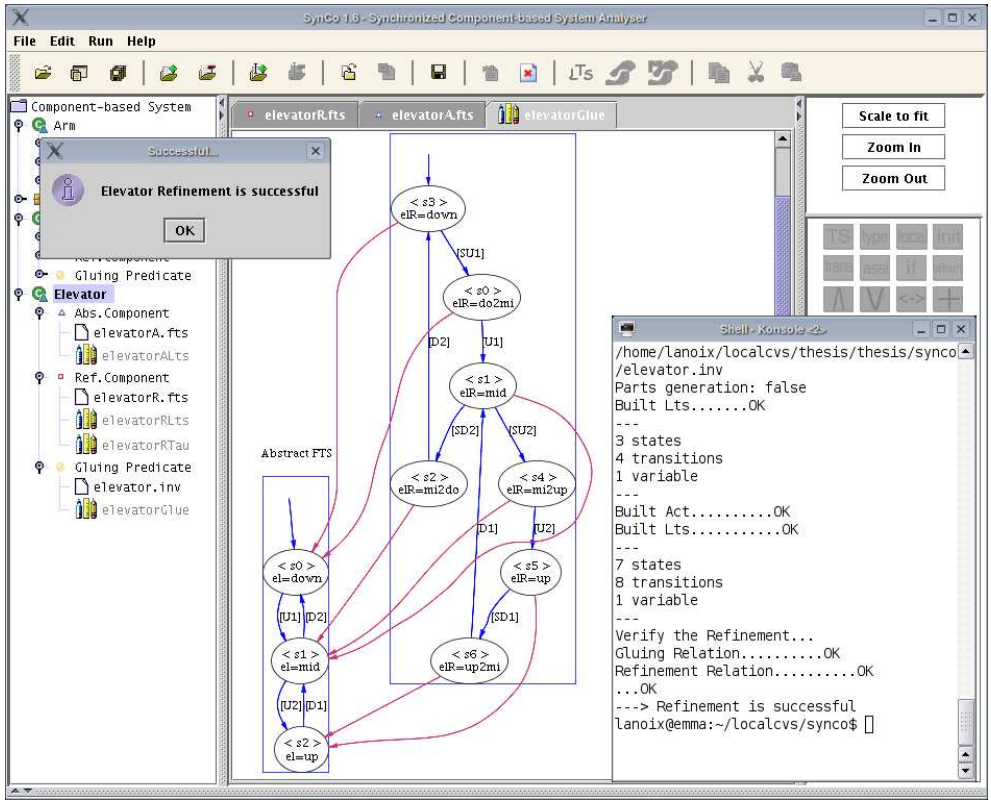


FIG. 6.8: Relation de raffinement η entre $Elev_R$ et $Elev_A$

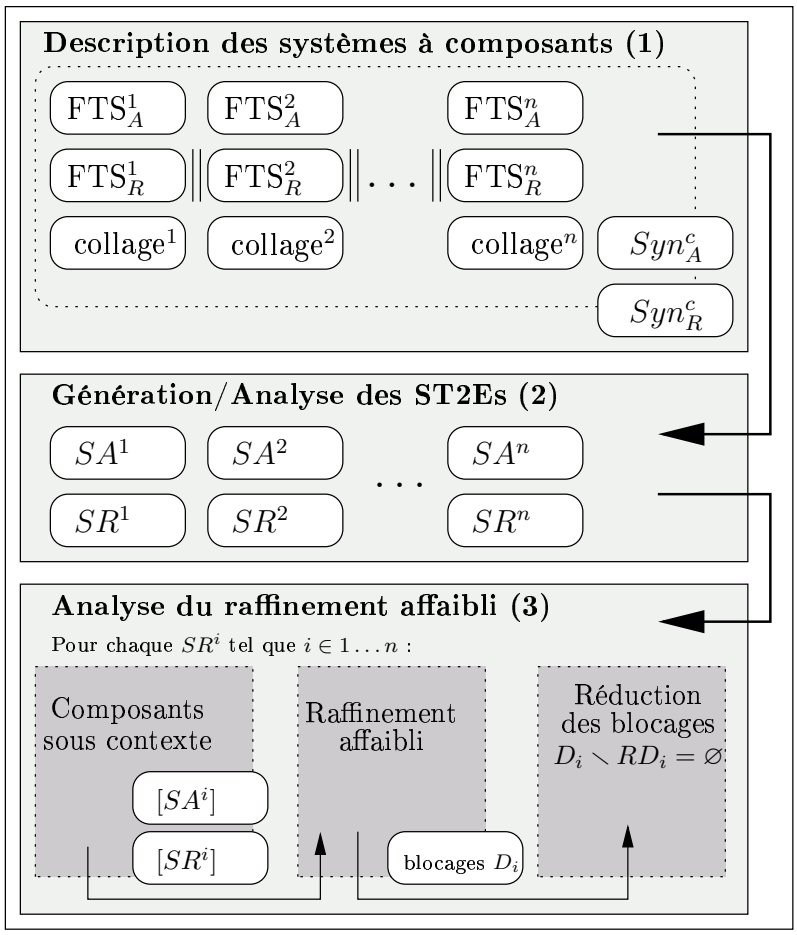


Fig. 6.9: Architecture de SynCo

Exemple. Avec SynCo, on peut vérifier que $(Arm_R, Clip_R, Elev_R, Syn^c_R)$ raffine $(Arm_A, Clip_A, Elev_A, Syn^c_A)$. Les composants sous contexte sont générés les uns après les autres et leurs raffinements affaiblis sont étudiés. Finalement, on conclut que $(Arm_R, Clip_R, Elev_R, Syn^c_R) \sqsubseteq_\eta (Arm_A, Clip_A, Elev_A, Syn^c_A)$ (voir figure 6.10).

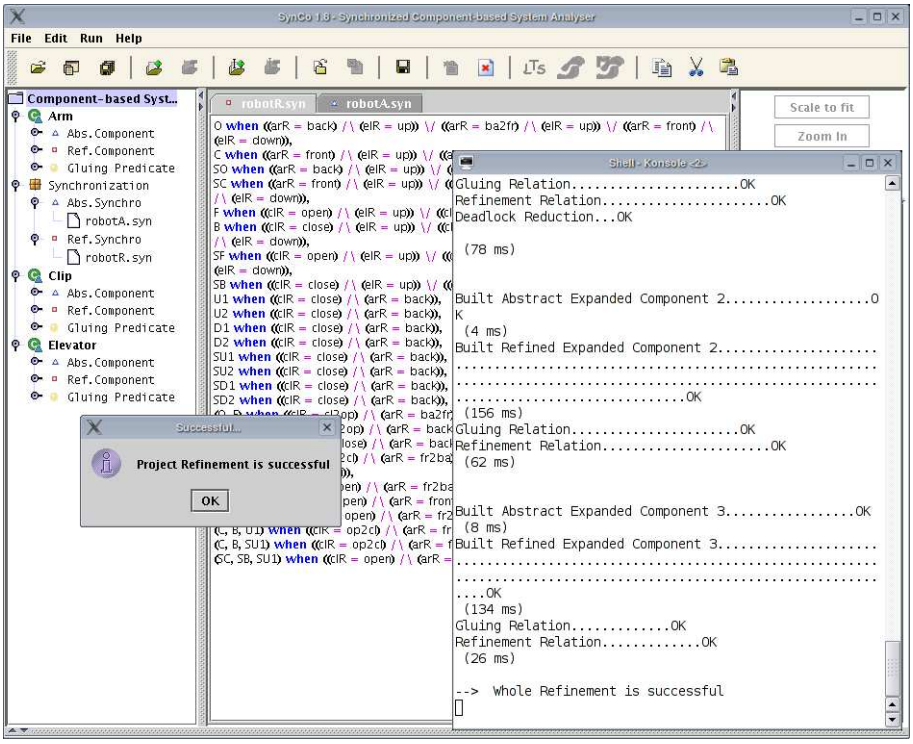


Fig. 6.10: Vérification sous SynCo du raffinement de $(Arm_A, Clip_A, Elev_A, Syn^c_A)$

6.1.4 Collaborations entre SynCo et d'autres outils

Nous avons envisagé d'intégrer SynCo à plusieurs autres outils de vérification, afin d'améliorer les possibilités de SynCo quant à la vérification du raffinement et des propriétés des systèmes à composants.

- Puisque SynCo utilise l'un des langages d'entrée de STeP [MBB+95, BBC+98], il serait possible – bien que nous n'ayons pas explorer cette possibilité – de travailler conjointement avec SynCo et avec STeP, par exemple pour combiner preuve et vérification algorithmique du raffinement.
- Nous nous sommes inspirés des travaux de [BSZ00] pour établir un lien avec le model-checker SPIN [Hol97, Hol03, SPI], et écrire un traducteur automatique nous permettant de générer à partir des spécifications FTS des composants, les spécifications Promela correspondantes (voir section 3.2 à propos de SPIN). Cela permet de

1. Vérifier par model-checking des propriétés LTL sur les composants abstraits (voir section 5.3),
2. Vérifier ensuite le raffinement du système à composants grâce à SynCo et
3. Conclure automatiquement à propos de la préservation pour le système à composants raffinés des propriétés vérifiées sur les composants abstraits.

```
/* Autogenerate PROMELA from SynCo */

/* TYPE DEFINITION */
#define up 0
#define do2mi 1
#define mid 2
#define up2mi 3
#define ELEVATORR 4
#define mi2up 5
#define mi2do 6
#define down 7

/* Initialisation */
int elR = down;

/* Transitions */

proctype trans0() {
   do
    :: atomic{ (elR == do2mi) -> elR = mid }
   od
}

proctype trans1() {
   do
    :: atomic{ (elR == mi2do) -> elR = down }
   od
}

proctype trans2() {
   do
    :: atomic{ (elR == mi2up) -> elR = up }
   od
}

proctype trans3() {
   do
    :: atomic{ (elR == down) -> elR = do2mi }
   od
}

proctype trans4() {
   do
    :: atomic{ (elR == up) -> elR = up2mi }
   od
}

proctype trans5() {
   do
    :: atomic{ (elR == mid) -> elR = mi2up }
    :: atomic{ (elR == mid) -> elR = mi2do }
   od
}

proctype trans6() {
   do
    :: atomic{ (elR == up2mi) -> elR = mid }
   od
}

/* MAIN FUNCTION */
init {
   atomic {
      run trans0();
      run trans1();
      run trans2();
      run trans3();
      run trans4();
      run trans5();
      run trans6();
   }
}
```

FIG. 6.11: Spécification Promela correspondant à $Elev_R$

Exemple. La figure 6.11 illustre la spécification Promela générée automatiquement grâce à SynCo à partir de la spécification FTS du composant $Elev_A$ donnée figure 6.7(a).

- Nous avons aussi lié SynCo avec ALDEBARAN et BCG, deux des outils qui font partie du toolkit CADP développé par l'équipe VASY de l'INRIA Rhônes-Alpes [FM91b, GJM+97, GLM02, VAS] :

- BCG (pour Binary Coded Graphs) est avant tout un format de représentation efficace pour des STEs de grande taille. Certaines fonctionnalités sont particulièrement intéressantes. BCG_LABELS permet de masquer et/ou de renommer certaines étiquettes de transitions d'un STE. BCG_MIN minimise un STE suivant la bisimulation forte ou la bisimulation de branchement.
- ALDEBARAN est un outil permettant principalement de réduire ou de comparer des STEs par rapport à différentes relations d'équivalence ou de préordre classiques : bisimulation forte [Par81], $\tau^*.a$-bisimulation [Fer90], équivalence observationnelle [Mil71, Mil89], bisimulation de branchement [vG90], etc. De plus, ALDEBARAN utilise différentes méthodes pour la réduction du STE : Algorithme de Paige et Tarjan [PT87], vérification à la volée [FM90, FM91a], utilisation de BDD.
- BISIMULATOR [Mat03, BDJM05] propose aussi de calculer plusieurs relations d'équivalences ou de préordre : la bisimulation forte, l'équivalence observationnelle, la τ^*, a bisimulation, l'équivalence de sûreté, l'équivalence de traces, la bisimulation de branchement, etc.

Nous avons implanté dans SynCo un traducteur vers ALDEBARAN. A partir d'un fichier ".fts" décrivant un ST2E dans le langage des FTS, nous générons automatiquement deux fichiers : le fichier ".aut" contient la description de l'ensemble des transitions du STEs (fichier d'entrée d'ALDEBARAN) et, le fichier ".states", l'interprétation des états.

```
des (0, 8, 7)
(1, [U1], 2)
(6, [D2], 0)
(2, [SU2], 3)
(3, [U2], 4)
(5, [D1], 2)
(4, [SD1], 5)
(0, [SU1], 1)
(2, [SD2], 6)
```

(a) Fichier ".aut"

```
1 : elR = do2mi
6 : elR = mi2do
3 : elR = mi2up
0 : elR = down
4 : elR = up
2 : elR = mid
5 : elR = up2mi
```

(b) Fichier ".states"

FIG. 6.12: Fichiers ALDEBARAN correspondant à $Elev_R$

Exemple. La figure 6.12 donne les fichiers ".aut" et ".states" générés automatiquement par SynCo à partir de la spécification FTS du composant $Elev_R$ donnée figure 6.7(a).

Le fichier ".aut" commence par indiquer le numéro de l'état initial, puis le nombre de transitions et le nombre d'états (ici, 'des (0, 8, 7)'). Ensuite chacune des transitions de $Elev_R$ est décrite : par exemple, '(1,[U1],2)' indique qu'il existe une transition étiquetée '[U1]' entre les états '1' et '2' du système de transitions considéré. Le fichier ".states" donne l'interprétation de chacun des états figurant dans le fichier ".aut" : '4 : elR=up' indique que, dans l'état '4', la variable 'elR' est égale à 'up'.

Comme l'implantation du calcul de la relation de raffinement η que nous avons réalisée dans SynCo est loin d'être aussi performante que les calculs de simulations ou de classes d'équivalence réalisés par ALDEBARAN, nous aurions souhaité pouvoir utiliser ALDEBARAN pour calculer tout ou partie de la relation de raffinement η. Malheureusement, même en combinant plusieurs calculs de simulation et en imaginant des traitements extérieurs, les relations de simulation calculées par ALDEBARAN ne correspondent pas à η. Les problèmes principaux concernent surtout l'établissement de la relation de collage, mais aussi la vérification d'absence de τ-divergence, la τ-simulation ϱ correspondant à la $\tau^*.a$-bisimulation [Fer90].

6.2 Etudes de cas réalisées grâce à SynCo

6.2.1 Module Essuyage Avant

Il s'agit d'une spécification réalisée à partir d'une étude de cas réalisée pour PSA par Séverine Colin lors de son stage de DEA [Col02, BCL02].

Le module *Essuyage Avant* fait partie d'un système embarqué plus important chargé de la gestion de toutes les fonctionnalités ayant trait à tout ce qui concerne la visibilité d'une automobile : lavage du pare-brise, de la lunette arrière et des projecteurs, l'essuyage du pare-brise et de la lunette arrière, la gestion du dégivrage de la lunette arrière et des projecteurs, etc.

Le module Essuyage Avant concerne plus particulièrement le contrôle de l'essuie-vitre avant. Ce module est contrôlé par un *comodo*, c.à.d. une manette de commande qui permet au conducteur d'activer les essuies-vitres avant, de se placer en mode automatique et de choisir la vitesse d'essuyage. En mode automatique, un *capteur de pluie* (*Rain Sensor*) détecte s'il pleut ou non, et la force de la pluie afin de positionner automatiquement la bonne vitesse d'essuyage. Le module Essuyage Avant contrôle aussi les moteurs des *deux essuies-vitres gauche et droit* (*Left Wiper* et *Right Wiper*), c.à.d. leurs états et leurs vitesses.

Nous avons identifié quatre composants qui interagissent les uns avec les autres : le comodo, le capteur de pluie et les moteurs des deux essuie-vitres gauche et droit. Le comodo et le capteur de pluie indiquent aux moteurs des essuie-vitres de monter et de descendre, et à quelle vitesse. Le comodo active le capteur de pluie (passage en mode automatique) et le désactive.

Nous allons spécifier le module Essuyage Avant sous la forme d'un système à composants synchronisés, avec un niveau de raffinement. Chacun des composants sera spécifié par deux ST2Es décrits en FTS, l'un abstrait, le second plus raffiné : $Comodo_A$ et $Comodo_R$ pour le comodo, $Sensor_A$ et $Sensor_R$ pour le capteur de pluie, $Left_A$ et $Left_R$ pour le moteur de l'essuie-vitre gauche, et, $Right_A$ et $Right_R$ pour le moteur de l'essuie-vitre droit. Nous spécifierons ensuite les interactions entre les composants par des ensembles de synchronisations contraintes csw^c_A et csw^c_R et nous étudierons si le système à composants raffinés $(Comodo_R, Sensor_R, Left_R, Right_R, csw^c_R)$ est bien un raffinement correct du système à

composants abstraits ($Comodo_A, Sensor_A, Left_A, Right_A, csw_R^A$).

Spécifications du composant "comodo"

La figure 6.13 donne les spécifications abstraites et raffinées du comodo.

Transition System
type COMODO = {stop, auto, manual}
local co : COMODO
Initially co = stop
Transition coStop :
  enable co != stop ;
  assign co := stop
Transition coAuto :
  enable co != auto ;
  assign co := auto
Transition coManual :
  enable co != manual ;
  assign co := manual

(a) $Comodo_A$

(co = manual ⇔ coR = manual)
∧ (co = stop ⇔ coR = stop)
∧ (co = auto ⇔ coR = auto)

(b) gp_{comodo}

Transition System
type COMODO = {stop, auto, manual}
type SPEED = {sp0, sp1, sp2}
local coR : COMODO
local cos : SPEED
Invariant
  (cos ∈ {sp1,sp2} ⇒ coR = manual) ∧
  (coR ∈ {stop, auto} ⇒ cos = sp1)
Initially coR = stop ∧ cos = sp0
Transition coStop :
  enable coR != stop ;
  assign coR := stop, cos := sp0
Transition coAuto :
  enable coR != auto ;
  assign coR := auto, cos := sp0
Transition coManual :
  enable coR != manual ;
  assign coR := manual, cos := sp1
Transition coSelect :
  enable coR = manual ∧ cos = sp1 ;
  assign cos := sp2

(c) $Comodo_R$

Fig. 6.13: Spécifications du composant comodo

Dans un premier temps, nous n'observons que l'état d'activation par le conducteur du système d'essuyage avant. L'ensemble $COMODO = \{stop, auto, manual\}$ de la spécification abstraite $Comodo_A$ indique si les essuies-vitres sont arrêtés ($stop$), en fonctionnement manuel ($manual$), ou en mode automatique ($auto$). Les trois transitions $coStop$, $coAuto$ et $coManual$ modélisent les actions du conducteur sur le comodo.

Dans la spécification raffinée $Comodo_R$, nous introduisons la vitesse d'essuyage choisie par le conducteur quand il se place en mode manuel. L'ensemble $SPEED = \{sp0, sp1, sp2\}$ indique deux niveaux de vitesses ($sp1$ et $sp2$), ainsi que la position arrêtée ($sp0$). Les transitions $coStop$ et $coAuto$ remettent le choix de vitesse à $sp0$, la transition $coManual$ place par défaut la vitesse à $sp1$. Une nouvelle transition, $coSelect$, permet de passer en vitesse $sp2$ uniquement quand on est en mode manuel.

Nous avons aussi spécifié deux propriétés invariantes dans $Comodo_R$:
- $cos \in \{sp1, sp2\} \Rightarrow coR = manual$ indique que si le choix de vitesse est $sp1$ ou $sp2$, alors on est forcément en mode manuel,

– $coR \in \{stop, auto\} \Rightarrow cos = sp0$ indique que si on est en mode automatique ou en mode arrêt, alors le choix de vitesse est forcément positionné à $sp0$.

Grâce à SynCo, nous pouvons vérifier de manière complètement automatique que $Comodo_R$ est bien un raffinement de $Comodo_A$.

Spécifications du composant “capteur de pluie"

La figure 6.14 donne les spécifications abstraites et raffinées du capteur de pluie.

Transition System
type RAIN = {off, not, tiny, strong}
local se : RAIN
Initially se = off
Transition seOff :
  enableenable se != off ;
  assign se := off
Transition seNot :
  enable se != not ;
  assign se := not
Transition seTiny :
  enable se != tiny ;
  assign se := tiny
Transition seStrong :
  enable se != strong ;
  assign se := strong

(a) $Sensor_A$

(se = off ⇔ seR = offR)
∧ (se = not ⇔ seR = notR)
∧ (se = strong ⇔ seR ∈{strongR, minStrongR})
∧ (se = tiny ⇔ seR ∈ {tinyR, maxTinyR})

(b) gp_{sensor}

Transition System
type RAINR = {offR, notR, tinyR,
  maxTinyR, strongR, minStrongR}
local seR : RAINR
Initially seR = offR
Transition seOff :
  enable seR != offR ;
  assign seR := offR
Transition seNot :
  enable seR != notR ;
  assign seR := notR
Transition seTiny :
  enable seR = notR ∨ seR = minStrongR ;
  assign seR := tinyR
Transition seStrong :
  enable seR = notR ∨ seR = maxTinyR ;
  assign seR := strongR
Transition seMaxTiny :
  enable seR = tinyR ;
  enable seR := maxTinyR
Transition seMinStrong :
  enable seR = strongR ;
  enable seR := minStrongR

(c) $Sensor_R$

Fig. 6.14: Spécifications du composant capteur de pluie

La spécification abstraite $Sensor_A$ du capteur de pluie donnée figure 6.14(a) décrit l'état du capteur de pluie. L'ensemble $RAIN = \{off, not, tiny, strong\}$ indique si le capteur de pluie est arrêté (off) ou s'il est en fonctionnement et si il ne détecte pas du tout de pluie (not), s'il détecte un peu de pluie ($tiny$) ou s'il détecte beaucoup de pluie ($strong$). Les transitions $seOff$, $seNot$, $seTiny$ et $seStrong$ modélisent ces changements d'état.

Dans la spécification raffinée $Sensor_R$ donnée figure 6.14(c), nous nous intéressons à spécifier de manière plus précise la limite entre la petite pluie et la grande pluie. Nous raf-

finons l'ensemble $RAIN$ par l'ensemble $RAINR$ en introduisant deux nouvelles valeurs $maxTinyR$, pour indiquer la limite maximum avant le changement vers grande pluie, et $minStrongR$, pour indiquer la limite minimum avant le changement vers petite pluie. Deux nouvelles transitions $seMaxTiny$ et $seMinStrong$ spécifient ces changements d'états.

Nous pouvons vérifier que $Sensor_A$ est bien raffiné par $Sensor_R$ à l'aide de SynCo.

Spécifications des composants "essuie-vitres"

La figure 6.15 donne les spécifications abstraites et raffinées de l'essuie-vitre gauche et la figure 6.16 celles de l'essuie-vitre droit.

Transition System
type POS = {up, down}
local lw : POS
Initially lw = down
Transition leftDown :
  enable lw = up ;
  assign lw := down
Transition leftUp :
  enable lw = down ;
  assign lw := up

(a) $Left_A$

(lw = down ⇔ lwR = down)
∧ (lw = up ⇔ (lwR = up ∧ lws != sp0)

(b) gp_{left}

Transition System
type POS = {up, down}
type SPEED = {sp0, sp1, sp2}
local lwR : POS
local lws : SPEED
Invariant
  (lws = sp0 ⇒ lwR = down) ∧
  (lwR = up ⇒ (lws = sp1 ∨ lws = sp2))
Initially lwR = down ∧ lws = sp0
Transition leftDown :
  enable lwR = up ;
  assign lwR := down, lws := sp0
Transition leftUp :
  enable lwR = down ∧ lws ∈ {sp1, sp2} ;
  assign lwR := up
Transition leftSpeed1 :
  enable lwR = down ∧ lws = sp0 ;
  enable lws := sp1
Transition leftSpeed2 :
  enable lwR = down ∧ lws = sp0 ;
  enable lws := sp2

(c) $Left_R$

FIG. 6.15: Spécifications du composant essuie-vitre gauche

Remarque. Les deux spécifications de l'essuie-vitre gauche et celles de l'essuie-vitre droit sont identiques modulo un renommage des variables et des transitions et nous n'expliquons que celles de l'essuie-vitre gauche.

L'idée de spécifier indépendamment chacun des essuie-vitres nous permettra d'affirmer que les essuie-vitres gauche et droit seront toujours synchronisés.

La figure 6.15(a) donne la spécification abstraite de $Left_A$. L'ensemble $POS = \{up, down\}$ nous permet de spécifier si l'essuie-vitre gauche doit être en position haute ou en position

Transition System
type POS = {up, down}
local rw : POS
Initially rw = down
Transition rightDown :
  enable rw = up ;
  assign rw := down
Transition rightUp :
  enable rw = down ;
  assign rw := up

(a) $Right_A$

(rw = down ⇔ rwR = down)
∧ (rw = up ⇔ (rwR = up ∧ rws != sp0)

(b) gp_{right}

Transition System
type POS = {up, down}
type SPEED = {sp0, sp1, sp2}
local rwR : POS
local rws : SPEED
Invariant
  (rws = sp0 ⇒ rwR = down) ∧
  (rwR = up ⇒ (rws = sp1 ∨ rws = sp2))
Initially rwR = down ∧ rws = sp0
Transition rightDown :
  enable rwR = up ;
  assign rwR := down, rws := sp0
Transition rightUp :
  enable rwR = down ∧ rws ∈ {sp1, sp2} ;
  assign rwR := up
Transition rightSpeed1 :
  enable rwR = down ∧ rws = sp0 ;
  enable rws := sp1
Transition rightSpeed2 :
  enable rwR = down ∧ rws = sp0 ;
  enable rws := sp2

(c) $Right_R$

FIG. 6.16: Spécifications du composant essuie-vitre droit

basse. Bien entendu, au départ l'essuie-vitre est en bas. Les transitions $leftDown$ et $leftUp$ spécifient les mouvements de l'essuie-vitre. Au niveau raffiné (figure 6.15(c)), la spécification $Left_R$ introduit un ensemble $SPEED = \{sp0, sp1, sp2\}$ pour indiquer la vitesse d'essuyage de l'essuie-vitre gauche. Deux nouvelles transitions $leftSpeed1$ et $leftSpeed2$ correspondent au choix de la bonne vitesse pour le moteur de l'essuie-vitre gauche.

Nous spécifions aussi deux propriétés d'invariance locales au composant $Left_R$:
- $lws = sp0 \Rightarrow lwR = down$ indique que si le moteur est en vitesse nulle, alors l'essuie-vitre est en position basse,
- $lwR = up \Rightarrow (lws = sp1 \vee lws = sp2)$ indique que si l'essuie-vitre est en position haute, alors la vitesse du moteur est $sp1$ ou $sp2$.

Grâce à SynCo, nous pouvons vérifier que $Left_R$ est un raffinement de $Left_A$ (et que $Right_R$ est un raffinement de $Right_A$).

Spécifications des interactions entre les différents composants

La figure 6.17 donne un ensemble de synchronisations contraintes csw_A^c pour les composants $Comodo_A$, $Sensor_A$, $Left_A$ et $Right_A$ afin de spécifier le système à composants

$(Comodo_A, Sensor_A, Left_A, Right_A, csw^c_A)$, c.à.d. une vue abstraite du module Essuyage avant.

(lwDown, rwDown) **when**	coManual **when**
(co = manual ∧ se = off)	lw = rw = down ∧ co = stop ∧ se = off,
∨ (co = auto ∧ se ∈ {tiny,strong}),	coStop **when**
(lwUp, rwUp) **when**	lw = rw = down ∧ co = manual ∧ se = off,
(co = manual ∧ se = off)	seTiny **when**
∨ (co = auto ∧ se ∈ {tiny,strong}),	lw = rw = down ∧ co = auto ∧ se != off,
(coAuto, seNot) **when**	seStrong **when**
se = off ∧ lw = rw = down,	lw = rw = down ∧ co = auto ∧ se != off,
(coManual, seOff) **when**	seNot **when**
co = auto ∧ lw = rw = down,	lw = rw = down ∧ co = auto ∧ se != off
(coStop, seOff) **when**	
co = auto ∧ lw = rw = down,	

Fig. 6.17: Ensemble de synchronisations csw^c_A

- Les essuie-vitres gauches et droits évoluent simultanément, c'est pourquoi nous synchronisons les transitions $(lwDown, rwDown)$ et $(lwUp, rwUp)$.
- On ne pourra modifier le comportement des essuies-vitres que quand ils seront en position basse. Cela explique que toutes les autres synchronisations soient gardées par $lw = rw = down$.
- Quand un conducteur choisira de passer en mode automatique de détection de pluie, il faut que le capteur soit allumé, c'est pourquoi nous synchronisons les transitions $(coAuto, seNot)$ et que nous avons comme garde $co = auto$.
- De la même manière, si le conducteur décide de revenir en mode manuel ou d'éteindre les essuie-vitres, il faut éteindre le capteur de pluie. Nous avons donc synchronisé les transitions $(coManual, seOff)$ et $(coStop, seOff)$ si l'on a $co = auto$.
- Si l'on n'est pas en mode automatique, le passage en mode manuel a lieu indépendamment, de même que l'arrêt des essuie-vitres n'a pas de raison d'influer sur les autres composants. C'est pourquoi nous avons $coManual$ et $coStop$.
- Le capteur de pluie fonctionne uniquement quand on est en mode automatique. Les transitions $seTiny$, $seStrong$ et $seNot$ ont lieu uniquement si l'on a $co = auto$.

L'ensemble raffiné de synchronisations contraintes csw^c_R est donné figure 6.18. Il permet de spécifier le système à composants synchronisés raffinés $(Comodo_R, Sensor_R, Left_R, Right_R, csw^c_R)$. Les nouvelles transitions sont, elles aussi, synchronisées. Les vitesses des essuie-vitres droit et gauche sont fixées simultanément : $(lwSpeed1, rwSpeed1)$ et $(lwSpeed2, rwSpeed2)$. Si le niveau limite de grande ou de petite pluie est atteinte, alors on en tient compte : $(seMaxTiny, lwSpeed1, rwSpeed1)$ et $(seMinStrong, lwSpeed2, rwSpeed2)$. Les gardes des synchronisations tiennent maintenant compte des nouvelles variables introduites par le raffinement des composants.

(lwDown, rwDown) **when**
(coR = manual ∧ cos = lws = rws
∧ seR = offR ∧ cos ∈ {sp1, sp2})
∨ (coR = auto ∧ lws = rws = sp1
∧ seR ∈ {tinyR, maxTinyR})
∨ (coR = auto ∧ lws = rws = sp2
∧ seR ∈ {strongR, minStrongR}),
(lwUp, rwUp) **when**
(coR = manual ∧ cos = lws = rws
∧ seR = offR ∧ cos ∈ {sp1, sp2})
∨ (coR = auto ∧ lws = rws = sp1
∧ seR ∈ {tinyR, maxTinyR})
∨ (coR = auto ∧ lws = rws = sp2
∧ seR ∈ {strongR, minStrongR}),
coManual **when**
lwR = rwR = down ∧ lws = rws = sp0
∧ coR = stop ∧ seR = offR,
coStop **when**
lwR = rwR = down ∧ lws = rws = sp0
∧ coR = manual ∧ seR = offR,
seTiny **when**
lwR = rwR = down ∧ lws = rws = sp0
∧ coR = auto ∧ seR != offR,
seStrong **when**
lwR = rwR = down ∧ lws = rws = sp0
∧ coR = auto ∧ seR != offR,
seNot **when**
lwR = rwR = down ∧ lws = rws = sp0
∧ coR = auto ∧ seR != offR,
(coManual, seOff) **when**
lwR = rwR = down ∧ lws = rws = sp0 ∧ coR = auto,
(coStop, seOff) **when**
lwR = rwR = down ∧ lws = rws = sp0 ∧ coR = auto,
(coAuto, seNot) **when**
lwR = rwR = down ∧ lws = rws = sp0 ∧ seR = offR,
seMaxTiny **when**
lwR = rwR = down ∧ lws = rws = sp0 ∧ coR = auto
∧ seR = tinyR,
seMinStrong **when**
lwR = rwR = down ∧ lws = rws = sp0 ∧ coR = auto
∧ seR = strongR,
coSelect **when**
lwR = rwR = down ∧ lws = rws = sp0 ∧ coR = auto
∧ seR = offR,
(lwSpeed1, rwSpeed1) **when**
(coR = manual ∧ cos = sp1 ∧ seR = offR)
∨ (coR = auto ∧ seR ∈ {tinyR, maxTinyR}),
(lwSpeed2, rwSpeed2) **when**
(coR = manual ∧ cos = sp2 ∧ seR = offR)
∨ (coR = auto ∧ seR ∈ {strongR, minStrongR}),
(seMaxTiny, lwSpeed1, rwSpeed1) **when**
lwR = rwR = down ∧ lws = rws = sp0 ∧ coR = auto
∧ seR = tinyR,
(seMinStrong, lwSpeed2, rwSpeed2) **when**
lwR = rwR = down ∧ lws = rws = sp0 ∧ coR = auto
∧ seR = strongR

FIG. 6.18: Ensemble de synchronisations csw_R^c

Vérification compositionnelle du raffinement par SynCo

Grâce à SynCo, on construit successivement chacun des composants sous-contexte raffiné et abstrait, puis on calcule leurs relations de raffinement affaiblis :

1. $[Comodo_R] \sqsubseteq_{\rho_w}^{D_{co}} [Comodo_A]$,
2. $[Sensor_R] \sqsubseteq_{\rho_w}^{D_{se}} [Sensor_A]$,
3. $[Left_R] \sqsubseteq_{\rho_w}^{D_{le}} [Left_A]$, et
4. $[Right_R] \sqsubseteq_{\rho_w}^{D_{ri}} [Right_A]$.

Le raffinement affaibli est vérifié pour chacun des composants sous-contexte. Les ensembles de blocages D_{co}, D_{se}, D_{le} puis finalement D_{ri} sont réduits. Comme de plus, le raffinement de chaque composant simple est vérifié, alors on est sûr que ($Comodo_R, Sensor_R, Left_R, Right_R, csw_R^c$) ne contient pas de τ-divergence. SynCo en déduit automatiquement que ($Comodo_A, Sensor_A, Left_A, Right_A, csw_A^c$) est bien raffiné par ($Comodo_R, Sensor_R, Left_R, Right_R, csw_R^c$). La figure 6.19 illustre ce résultat.

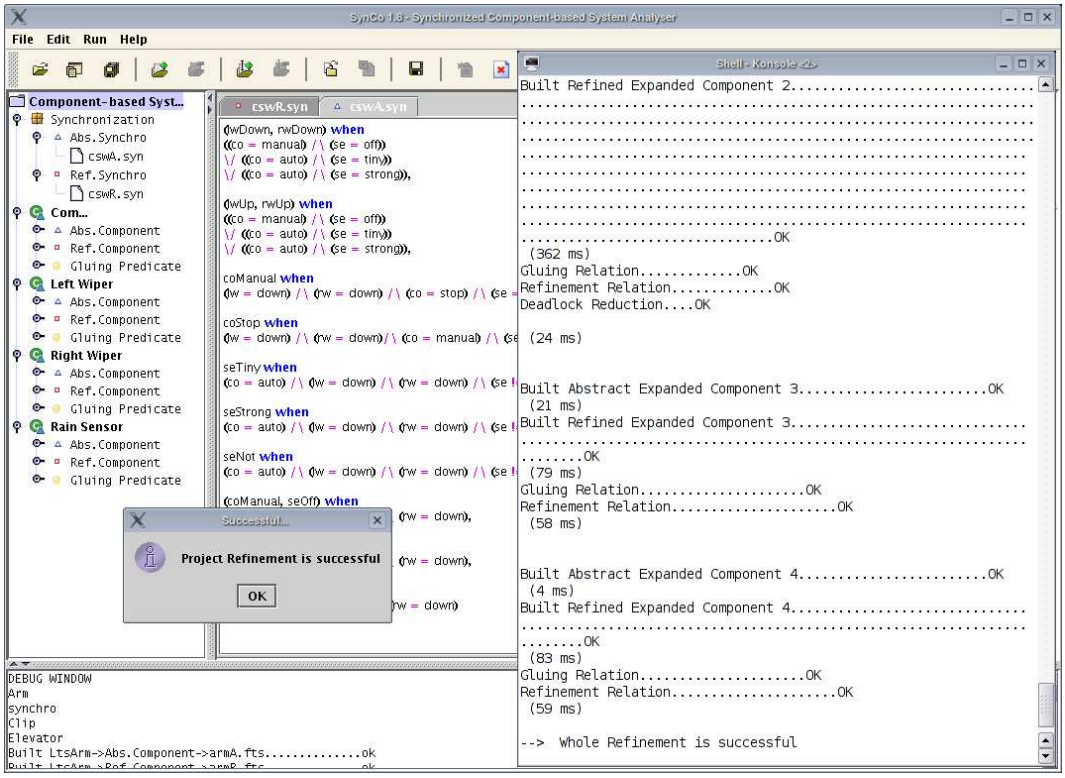


FIG. 6.19: Vérification du raffinement de $(Comodo_A, Sensor_A, Left_A, Right_A, csw^c_A)$

6.2.2 Porte-monnaie électronique CEPS

La norme CEPS[3] [CEP99, CEP00] définit un standard pour les applications de porte-monnaies électroniques embarqués sur carte à puce. Elle fournit les informations nécessaires à la conception d'un système de porte-monnaie électronique dont les entités sont interopérables. Cette norme contient, entre autre :

- les fonctionnalités requises par chacune des entités du système (carte CEP, terminaux d'achat, terminaux de chargement, etc.),
- la description de toutes les commandes et réponses qui constituent l'interface entre la carte et les périphériques qui la supportent,
- les opérations de cryptographie requises pour l'interopérabilité.

Différentes entités entrent en jeu dans un système de porte-monnaie électronique de type CEPS :

- *La Carte CEP* constitue l'élément central de la norme. Il s'agit d'une carte à puce munie de plusieurs slots mémoires. Chaque slot contient des unités correspondant à la monnaie dans une devise particulière. Suivant le terminal dans lequel on insère la carte, des commandes différentes permettent soit de consommer des unités, soit d'en recharger, soit d'en convertir en les faisant passer d'un slot à l'autre. La carte enregistre dans une

[3]pour *Common Electronic Purse Specification.*

table spécifique toutes les opérations effectuées avec succès.
- *Le Load Device* (*terminal de chargement*) permet de recharger des unités sur l'un des slots de la carte en débitant le compte bancaire auquel le porteur de la carte est rattaché. Il existe deux types de chargement selon que le fournisseur de la carte et le fournisseur des fonds sont deux entités différentes ou non.
- *Le POS Device* (*terminal d'achat*) constitue l'appareil installé chez le commerçant et qui permet d'effectuer ses règlements grâce à la carte CEP.
- *Le Scheme Provider* est l'autorité qui établit les règles régissant le fonctionnement et la sécurité du système CEPS. Elle distribue à toutes les entités des certificats leur permettant de s'authentifier.
- *Le Merchant Acquirer* est le responsable des POS devices. Il est chargé de collecter et de valider les transactions, puis d'accuser réception auprès des POS devices.
- *Le Load Acquirer* gère les requêtes lors d'une transaction d'échange de devises ou lors d'un chargement.
- *Le Card Issuer* est chargé de l'authentification lors des transactions et il autorise ou non la distribution des fonds lors d'un chargement.

Plusieurs transactions entrent en jeu entre ces différentes entités.
- *L'achat* est une transaction qui se déroule en plusieurs temps. D'abord, la transaction est initialisée par un POS device. Ensuite, la carte CEP et le POS device s'authentifient en échangeant leurs certificats, puis un premier débit est réalisé, éventuellement suivi par d'autres débits consécutifs.
- *L'annulation du dernier achat* permet de rectifier le dernier achat, à deux conditions : que l'on soit sur le même POS device et qu'il n'y ait eu aucune autre transaction entre temps.
- *Le chargement* permet de créditer la carte CEP. Une fois la carte insérée dans un Load device, ce dernier initialise une nouvelle transaction, en demandant l'authentification. Une fois le transfert de fonds validé par l'autorité compétente, le montant chargé sur la carte est mis à jour.
- *L'échange de devices* peut avoir lieu à partir de certains Load devices. Après initialisation et authentification, la carte essaye de réaliser l'échange. Si l'échange est possible, alors la carte transfère les fonds entre les slots concernés.

L'étude que nous réalisons ici ne tient pas compte de toutes les entités listées précédemment. Nous allons nous concentrer sur les transactions entre la carte, un terminal d'achat et un terminal de chargement. De même, nous ne considérerons qu'un seul slot sur la carte, avec un montant minimum et un montant maximum.

Nous spécifions ces transactions entre la carte CEP, un terminal d'achat et un terminal de chargement par un système à composants synchronisés mettant en jeu trois composants, avec chacun un niveau de raffinement : $Card_A$ et $Card_R$ pour la carte CEP, $Load_A$ et $Load_R$ pour le terminal de chargement, et, pos_A et pos_R pour le terminal d'achat. Nous spécifions ensuite les transactions entre les différents composants par des ensembles de synchronisations contraintes abstrait $ceps^c_A$ et raffiné $ceps^c_R$ avant de nous assurer grâce à SynCo que

$(Card_R, Load_R, Pos_R, ceps^c_R)$ est bien un raffinement correct de $(Card_A, Load_A, Pos_A, ceps^c_A)$.

Remarque. Les spécifications FTS complètes – abstraites et raffinées – pour chacun des composants ainsi que celles des ensembles de synchronisations ne sont pas présentées dans ce document, étant donné leurs tailles (environ 1000 lignes). Elles peuvent être trouvées dans [Col04a, Col04b].

Spécifications des composants

Au niveau abstrait, les trois composants $Card_A$, $Load_A$ et Pos_A n'ont qu'une variable indiquant leurs statuts (voir figure 6.20).

$$load\_device\_statusA \in \left\{ \begin{array}{l} Status\_Ready, Status\_LoadInit, \\ Status\_ReceiveId, Status\_IdAccept \end{array} \right\}$$

$$pos\_device\_statusA \in \left\{ \begin{array}{l} Status\_Ready, Status\_PurchaseInit, \\ Status\_PurchaseDebited, Status\_ReceiveId, \\ Status\_IdAccept, Status\_PurchaseDebitCancel \end{array} \right\}$$

$$card\_statusA \in \left\{ \begin{array}{l} Status\_On, Status\_Ready, Status\_LoadInit, \\ Status\_PurchaseInit, Status\_PurchaseDebited, \\ Status\_SendId, Status\_IdAccept, \\ Status\_PurchaseDebitCancel \end{array} \right\}$$

FIG. 6.20: Variables indiquant le satut des composants $Load_A$, Pos_A et $Card_A$

Des transitions dans chacun des composants permettent de modifier le statut du composant considéré. Ces transitions correspondent au chargement et à l'achat, ainsi qu'à l'échange de certificats nécessaires pour initialiser toute transaction. La transition *cardIdAccept*, donnée figure 6.21, fait passer le statut de la carte CEP de $Status\_SendId$ à $Status\_IdAccept$.

Transition cardIdAccept :
enable (card_statusA = Status_SendId) ;
assign card_statusA := Status_IdAccept

FIG. 6.21: Transition *cardIdAccept* du composant $Card_A$

Au niveau raffiné, nous ajoutons de nouvelles variables. Dans chacun des composants, trois nouvelle variables $card\_value$, $load\_device\_value$ et pos_device_value permettent d'indiquer la valeur échangée entre la carte et l'un des terminaux. Pour la carte, nous avons aussi des variables permettant de représenter le montant présent actuellement sur la carte $card\_balance$, le montant maximum $card\_balance\_max$, le montant en cours d'opération $card\_bal\_init$ ainsi que le montant du dernier achat effectué $card\_value\_last\_purchase$.

Au niveau raffiné, les transitions traitent maintenant les débits et les crédits, ainsi que l'annulation de la dernière opération effectuée. Elles permettent de modifier les variables

introduites par le raffinement. Par exemple, l'opération *cardPurchaseDebit*, figure 6.22, débite la carte CEP, lors d'un paiement dans un terminal d'achat et modifie en conséquence les variables nécessaires.

```
Transition cardPurchaseDebit :
    enable card_statusR = Status_PurchaseInit
        ∧ card_bal_init <= card_balance_max
        ∧ (card_balance - card_bal_init) >= 0 ;
    assign card_statusR := Status_PurchaseDebited,
        card_balance := card_balance - card_bal_init,
        card_value_last_purchase := card_bal_init,
        card_bal_init := 0
```

FIG. 6.22: Transition *cardPurchaseDebit* du composant $Card_R$

Dans chacun des composants des invariants locaux expriment des propriétés de sûreté que le système complet doit vérifier.

Spécifications des interactions entre les différents composants

Les transactions entre la carte CEP, un terminal d'achat et un terminal de chargement, c.à.d. les interactions entre les composants sont modélisées par deux ensembles de synchronisations $ceps^c_A$, au niveau abstrait, et $ceps^c_R$, au niveau raffiné.
Au niveau abstrait, chaque transition de la carte CEP est synchronisée avec une transition d'un des terminaux. Par exemple, les transitions *cardLoadCredit* de la carte CEP $Card_A$ et *loadDeviceLoadCredit* du terminal de chargement $Load_A$ sont synchronisées lorsqu'ils sont dans le statut "Initialisation du chargement OK" $Status\_LoadInit$ et quand le terminal d'achat est en attente (voir figure 6.23).

```
(cardLoadCredit, loadDeviceLoadCredit) when
    card_statusA = load_device_statusA = Status_LoadInit
    ∧ pos_device_statusA = Status_Ready
```

FIG. 6.23: Extrait de $ceps^c_A$

Au niveau raffiné, nous retrouvons les mêmes synchronisations, ainsi que des synchronisations pour les chargements et les prélèvements sur la carte CEP et le terminal considéré.

Nous avons aussi spécifié des propriétés d'invariance globale (au niveau abstrait et au niveau raffiné) qui seront vérifiées par SynCo simultanément à la construction des composants sous-contextes. Dans l'invariant global abstrait Inv_{ceps_A}, nous vérifions, par exemple, que si les terminaux reçoivent une demande d'identification de la carte CEP, la carte leur en a bien envoyé une (voir figure 6.24).

$$
\begin{array}{l}
(load\_device\_statusA = Status\_ReceivedId \Rightarrow \\
(card\_statusA = Status\_SendId \wedge pos\_device\_statusA = Status\_Ready)) \\
\wedge \\
(pos\_device\_statusA = Status\_ReceivedId \Rightarrow \\
(card\_statusA = Status\_SendId \wedge load\_device\_statusA = Status\_Ready))
\end{array}
$$

FIG. 6.24: Extrait de Inv_{ceps_A}

Vérification compositionnelle du raffinement par SynCo

Comme nous l'avons déjà précisé, les spécifications complètes pour chacun des composants ainsi que celles des ensembles de synchronisations sont disponibles dans [Col04a, Col04b].

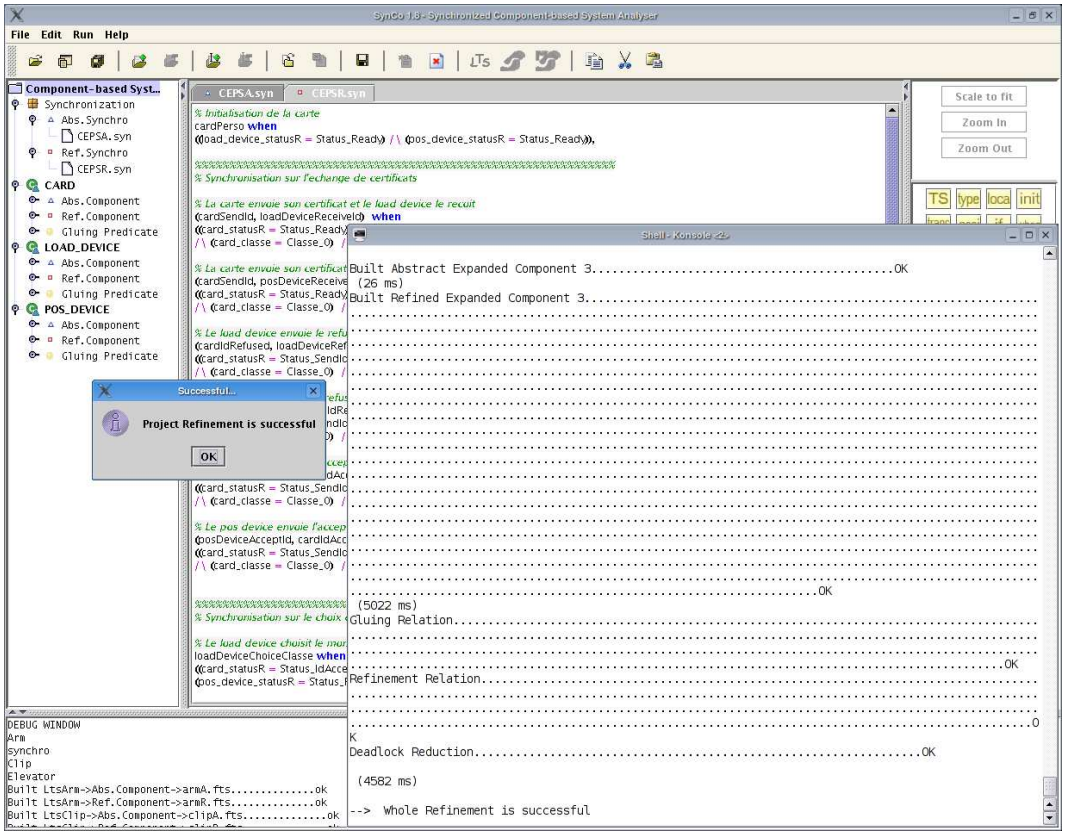


FIG. 6.25: Vérification du raffinement de $(Card_A, Load_A, Pos_A, ceps^c_A)$

Grâce à SynCo, nous avons vérifié que le système à composants raffinés $(Card_R, Load_R, Pos_R, ceps^c_R)$ est un raffinement correct de $(Card_A, Load_A, Pos_A, ceps^c_A)$. SynCo construit pour cela – successivement – chacun des composants sous-contexte raffiné et abstrait, puis calcule leurs relations affaiblies :

1. $[Card_R] \sqsubseteq^{D_{ca}}_{\rho_w} [Card_A]$,
2. $[Load_R] \sqsubseteq^{D_{lo}}_{\rho_w} [Load_A]$ et

3. $[Pos_R] \sqsubseteq_{\rho_w}^{D_{po}} [Pos_A]$.

Puisque le raffinement affaibli est vérifié pour chacun des composants sous-contexte et que les ensembles de blocages peuvent être réduits, alors, SynCo déduit que $(Card_A, Load_A, Pos_A, ceps_A^c)$ est bien raffiné par $(Card_R, Load_R, Pos_R, ceps_R^c)$ (voir figure 6.25).

6.3 Conclusion

Dans ce chapitre, nous avons présenté SynCo, le prototype développé afin de valider l'approche de la vérification compositionnelle du raffinement d'un système à composants synchronisés présentée dans le chapitre précédent.

	Composants			Composants sous-contexte			Système à composants
	Arm_A	$Clip_A$	$Elev_A$	$[Arm_A]$	$[Clip_A]$	$[Elev_A]$	$(Arm_A, Clip_A, Elev_A, Syn_A^c)$
$\|Q\|$	2	2	3	9	9	4	9
$\|T\|$	2	2	4	7	7	5	15
$\|Q\| + \|T\|$	**4**	**4**	**7**	**16**	**16**	**9**	**24**
	$Card_R$	$Load_R$	Pos_R	$[Card_R]$	$[Load_R]$	$[Pos_R]$	$(Arm_R, Clip_R, Elev_R, Syn_R^c)$
$\|Q\|$	4	4	7	19	19	11	24
$\|T\|$	4	4	8	19	20	12	40
$\|Q\| + \|T\|$	**8**	**8**	**15**	**38**	**39**	**23**	**64**

FIG. 6.26: Taille des ST2Es pour l'exemple du bras mobile robotisé

	Composants				Composants sous-contexte				Système à composants
	$Comodo_A$	$Sensor_A$	$Left_A$	$Right_A$	$[Comodo_A]$	$[Sensor_A]$	$[Left_A]$	$[Right_A]$	$(Comodo_A, Sensor_A, Left_A, Right_A, csw_A^c)$
$\|Q\|$	3	4	2	2	5	5	7	7	8
$\|T\|$	6	10	2	2	10	14	6	6	22
$\|Q\| + \|T\|$	**9**	**14**	**4**	**4**	**15**	**19**	**13**	**13**	**30**
	$Comodo_R$	$Sensor_R$	$Left_R$	$Right_R$	$[Comodo_R]$	$[Sensor_R]$	$[Left_R]$	$[Right_R]$	$(Comodo_R, Sensor_R, Left_R, Right_R, csw_R^c)$
$\|Q\|$	4	6	5	5	8	10	18	18	20
$\|T\|$	5	16	6	6	17	25	20	20	47
$\|Q\| + \|T\|$	**9**	**22**	**11**	**11**	**25**	**35**	**38**	**38**	**67**

FIG. 6.27: Taille des ST2Es pour l'exemple du module Essuyage avant

	Composants			Composants sous-contexte			Système à composants
	$Card_A$	$Load_A$	Pos_A	$[Card_A]$	$[Load_A]$	$[Pos_A]$	$(Card_A, Load_A, Pos_A, ceps_A^c)$
$\|Q\|$	8	4	6	10	5	7	10
$\|T\|$	15	6	11	18	6	11	18
$\|Q\| + \|T\|$	**23**	**10**	**17**	**28**	**11**	**18**	**28**
	$Card_R$	$Load_R$	Pos_R	$[Card_R]$	$[Load_R]$	$[Pos_R]$	$(Card_R, Load_R, Pos_R, ceps_R^c)$
$\|Q\|$	215	9	17	544	182	378	544
$\|T\|$	361	16	34	561	188	402	705
$\|Q\| + \|T\|$	**576**	**25**	**51**	**1105**	**370**	**780**	**1249**

FIG. 6.28: Taille des ST2Es pour l'exemple du porte-monnaie électronique CEPS

SynCo nous a permis de valider – en partie – notre approche sur quelques exemples. Nous avons présenté deux études de cas – le module Essuyage avant et le CEPS – réalisées

grâce à SynCo. Les figures 6.26, 6.27 et 6.28 donnent les nombres d'états et de transitions des différents systèmes de transitions – composants, composants sous-contexte et système complet – pour les différents exemples présentés dans ce document. L'objectif de notre démarche est de diminuer la taille des systèmes à explorer pour la vérification algorithmique du raffinement et des propriétés. La méthode proposée pour vérifier compositionnellement le raffinement consiste à explorer successivement chacun des composants sous-contexte, au lieu d'explorer directement le système complet. Même si le gain est minime et qu'il concerne plutôt le nombre de transitions que le nombre d'états, nous remarquons que dans les exemples étudiés le plus grand des composants sous-contexte raffinés est plus petit que le système complet.

Le cas de la carte CEP dans l'exemple de CEPS est édifiant : il s'agit du composant central de la modélisation réalisée, c'est la raison pour laquelle sa taille est si proche de celle du système complet. Pour que la démarche de vérification compositionnelle du raffinement soit la plus efficace possible il nous semble qu'il faille que la modélisation spécifie des composants tous aussi importants les uns que les autres. De plus, l'ensemble de synchronisations doit contenir des comportements synchronisés et des comportements indépendants.

Néanmoins, l'implantation actuelle de SynCo ne nous permet pas d'aborder les études de cas à échelle industrielle. Il faudrait pour cela améliorer la représentation mémoire des différents ST2Es et l'algorithme de parcours pour la vérification de la relation de raffinement. C'est dans cet objectif que nous avions mené une étude prospective vis-à-vis de CADP et particulièrement les outils ALDEBARAN et BCG.

Conclusion

L'objectif principal des travaux présentés dans ce document est le développement de méthodes formelles qui concilient les notions de composition et de raffinement, pour spécifier et vérifier des systèmes critiques.

Généralement, le raffinement suppose un processus de développement et de vérification "vertical" peu compatible avec la composition, vue comme une démarche de spécification "horizontale". La contribution scientifique de ces travaux concerne avant tout une démarche cherchant à concilier (dé)composition et raffinement, dans un cadre assez général pour pouvoir être étendu à d'autres approches.

L'intérêt principal de la démarche présentée dans ce document est que la composition et le raffinement ne contribuent pas seulement à la conception du modèle, mais aussi à la vérification des propriétés. Composition et raffinement sont rendus compatibles et permettent de diminuer l'impact du phénomène d'explosion combinatoire lors de la vérification, par exploration du modèle, du raffinement et des propriétés.

Synthèse et bilan

Nous fondons nos travaux sur une définition du raffinement en terme de τ-simulation préservant l'absence de τ-cycles et de nouveaux blocages [BJK00]. Cette relation de raffinement peut être calculée de manière algorithmique entre deux systèmes de transitions finis SR et SA. De plus, elle préserve la logique temporelle LTL : une propriété LTL satisfaite par un système abstrait SA est automatiquement satisfaite par un système plus concret SR, si la relation de raffinement est établie entre SA et SR. L'intérêt principal de la préservation est qu'elle simplifie le model-checking : dans une démarche de raffinement, il sera beaucoup moins coûteux de vérifier une propriété sur le système abstrait. C'est en effet à ce niveau que le nombre d'états à explorer est le plus petit. La satisfaction – par préservation – de la propriété est garantie sur les systèmes raffinés.

- Dans un premier temps, nous avons proposé au chapitre 4 une approche basée sur la *décomposition* du système pour vérifier de manière compositionnelle le raffinement ainsi que les propriétés du système. La décomposition d'un système en plusieurs sous-systèmes permet l'étude séparée de chacun des sous-systèmes. Plus précisément,
  – le raffinement du système complet est assuré – par décomposition – en vérifiant une

relation de raffinement "affaiblie" entre chaque sous-système abstrait et raffiné, en tenant compte des états de blocages introduit par la décomposition. Cela permet d'assurer le raffinement – affaibli – du système complet, recomposé ;
- plusieurs propriétés peuvent être vérifiées par décomposition, c.à.d., vérifiées sur les sous-systèmes et préservées automatiquement sur le système complet. Il s'agit de propriétés d'invariance, mais aussi des invariants dynamiques exprimant des propriétés de la forme $\Box(sp \Rightarrow \bigcirc sp')$ en LTL. Plus généralement, nous montrons que les propriétés LTL exprimant des sûretés qui ne sont pas satisfaites par au moins l'un des sous-systèmes, ne le seront pas non plus sur le système complet. Nous proposons aussi un algorithme permettant de s'assurer de manière compositionnelle de l'atteignabilié d'un état du système complet, en parcourant successivement les différents sous-systèmes.

- Nous étendons ensuite aux *systèmes à composants synchronisés* la démarche de vérification par décomposition. Dans le chapitre 5, nous proposons une extension du produit synchronisé d'Arnold et Nivat : les *systèmes à composants synchronisés*. Les différents composants sont donnés par des systèmes de transitions doublement étiquetés et les interactions entre composants sont quant-à-elles, exprimées par un ensemble de synchronisations : dans notre cas, nous contraignons l'activation des transitions synchronisées par des conditions d'activations.
  Afin d'étudier composant par composant le système à composants sans jamais calculer le produit complet, nous établissons un lien avec l'opérateur précédent de décomposition. Ce lien repose sur la notion de composant sous-contextes : il s'agit d'un composant enrichi par le contexte des autres composants, en restreignant l'ensemble de synchronisations aux comportements du composant considéré. Nous montrons ensuite que les composants sous-contexte forment une *décomposition* du système à composants synchronisés, obtenue automatiquement, sans avoir besoin de considérer le système dans son ensemble.
  Nous montrons ensuite que l'étude de la correction du raffinement d'un système à composants synchronisés se ramène, grâce aux composants sous-contexte, à réutiliser les résultats à propos du raffinement par décomposition :
  - la non τ-divergence dans un système à composants synchronisés raffinés est vérifiée de manière compositionnelle, puisqu'il suffit de vérifier l'absence de τ-cycles pour chacun des composants simples,
  - nous étudions successivement le raffinement affaibli de chacun des composants sous-contexte abstrait et raffiné,
  - pour chaque ensemble de blocages, nous calculons un ensemble de blocages réductibles, uniquement à partir de l'ensemble raffiné de synchronisations.

  Adapter le raffinement par décomposition au cas des systèmes à composants, nous permet d'étudier de manière compositionnelle le raffinement d'un système à composants synchronisés. Cette étude ne nécessite jamais de considérer le système à composants dans son ensemble, mais uniquement des composants sous-contexte, c.à.d. des systèmes plus "petits" que le système à composants complet.
  Plusieurs propriétés peuvent être vérifiées compositionnellement dans le cadre des systèmes à composants synchronisés. Certaines propriétés LTL de sûreté locales à l'un des

composants – en particulier les propriétés d'invariance – sont préservées automatiquement au niveau du système à composants complet. Grâce aux composants sous-contexte, nous pouvons aussi vérifier certaines propriétés globales à tout le système à composants en réutilisant les résultats à propos de la vérification par décomposition. Bien entendu, puisque le raffinement est vérifié, nous bénéficions de la préservation des propriétés par le raffinement. Nous pouvons envisager de vérifier compositionnellement des propriétés au niveau le plus abstrait, pour ensuite les préserver sur les systèmes complets raffinés.

- Finalement, dans le chapitre 6, nous présentons SynCo, le prototype implanté pour valider l'approche du raffinement des systèmes à composants synchronisés. SynCo permet de vérifier compositionnellement le raffinement d'un système à composants synchronisés, en construisant successivement chaque composant sous-contexte, en vérifiant son raffinement affaibli, puis en réduisant les blocages. De plus, SynCo permet actuellement de vérifier à la volée des propriétés d'invariances – locales ou globales – qui peuvent ensuite être préservées par le raffinement. SynCo nous a permis de valider notre approche sur quelques exemples.

Publications

Ce travail à donné lieu à plusieurs publications :
- La démarche de vérification compositionnelle du raffinement des systèmes à composants synchronisés est présentée dans [KL03a, KL03b].
- La vérification compositionnelle des propriétés d'invariance – locales et globales – est présentée dans [KL04].
- Le prototype SynCo qui met en oeuvre la démarche de vérification compositionnelle du raffinement des systèmes à composants est présenté dans [KL, Lan04b].
- D'autres publications présentent la démarche d'une manière plus générale [Lan04c, Lan04a].

Perspectives

Nous avons proposé une démarche pour la vérification compositionnelle du raffinement et des propriétés basée sur une décomposition du système complet, défini par des composants synchronisés. Les résultats obtenus sont suffisamment généraux pour pouvoir être transposés à d'autres domaines. En effet, pour établir nos résultats, nous nous plaçons, dans le cadre des systèmes de transitions – modèles sous-jacents à de nombreux langages de haut niveau – du produit synchronisé, des relations de simulation et de τ-simulation et de la logique temporelle. La portée de nos résultats dépasse donc le cadre de la décomposition et des systèmes à composants synchronisés et s'étend à tout formalisme ayant des caractéristiques identiques.

Dans le cadre de ces travaux, nous pouvons envisager plusieurs perspectives.

- Un des objectifs que nous visons est l'extension du cadre des systèmes à composants synchronisés à des *systèmes communicants*. Nous sommes pour cela en contact avec l'équipe *Architecture of Information Systems group* du département d'informatique de l'université *Eindhoven University of technology (TU/e)*. Cette équipe étudie principalement des systèmes communicants et leur vérification par des méthodes d'abstraction et de model-checking : une analyse de flot de données permet de détecter les variables d'un composant que l'on peut abstraire [ISS02b, ISS02a, SS02, ISS04].

- Le théorème de raffinement par décomposition pourrait être étendu aux *systèmes à variables partagées*. En effet, les sous-systèmes issus d'une décomposition peuvent être vus comme une restriction d'un système à variables partagées dans lequel toutes les variables seraient partagées. On doit pouvoir affiner le partage des variables, tout en restant dans le cadre d'applicabilité du raffinement par décomposition. L'idée est de rapprocher les notions de décomposition et de sous-système de celle de *modules* introduite par Kupferman, Vardi et Alur dans [KV96, KV98, AH99] pour décrire des systèmes à variables partagées (voir section 1.2.5).

- Plus généralement, il semble intéressant d'étudier une approche qui combinerait raffinement et composition par ajout de nouveaux composants à chaque niveau de raffinement. L'idée est de partir d'une spécification abstraite formée d'un unique composant S^1. Une étape de raffinement consiste alors à ajouter par composition un nouveau composant S^2. D'une manière générale, il est connu qu'une telle démarche préserve les propriétés de sûreté, puisque $S^1 \| S^2$ est τ-simulé par S^1. Il reste à définir des conditions sur l'opérateur de composition $\|$ permettant de garantir les propriétés de vivacité également, c.à.d. garantissant que $S^1 \| S^2$ est bien un raffinement de S^1, au sens de la relation de raffinement.

- Concernant SynCo, il est nécessaire d'améliorer la représentation mémoire des différents systèmes ainsi que l'algorithme de parcours pour la vérification de la relation de raffinement, afin de valider notre démarche sur des études de cas à échelle industrielle. Dans cet objectif, il serait intéressant de poursuivre notre étude prospective de différents outils comme ALDEBARAN et BCG.

Bibliographie

[ABL96] J.-R. Abrial, E. Bőrger, and H. Langmoeck. Specifying and programming the steam boiler control. In *Formal Methods for Industrial Applications (FMIA'96)*, volume 1165 of *LNCS*. Springer Verlag, 1996.

[Abr84] J.-R. Abrial. Spécifier ou comment matérialiser l'abstrait. *Technique et Science Informatique*, 3(3) :201–219, 1884.

[Abr96a] J.-R. Abrial. *The B Book*. Cambridge University Press, 1996.

[Abr96b] J.-R. Abrial. Extending B without changing it (for developing distributed systems). In *1st Conference on the B method*, pages 169–190, Nantes, France, November 1996.

[Abr97] J.-R. Abrial. Constructions d'automatismes industriels avec B. In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, Toulouse, France, Mai 1997. Invited lecture.

[Abr02] J.-R. Abrial. Discrete system models. Version 1.1, February 2002.

[AC] J.-R. Abrial and D. Cansell. Click'n'Prove. http ://www.loria.fr/ cansell/cnp.html.

[AdAG+01] R. Alur, L. de Alfaro, R. Grosu, T.A. Henzinger, M. Kang, R. Majumdar, F. Mang, C.M. Kirsch, and B.Y. Wang. Mocha : A model checker that exploits design structure. In *23rd International Conference on Software Engineering (ICSE'01)*, May 2001.

[AG00] R. Alur and R. Grosu. Modular refinement of hierarchic reactive machines. In *Principles of Programming Languages*, pages 390–402, 2000.

[AH99] R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design (FMSD)*, 15(1) :7–48, July 1999.

[AHI98] K. Ajami, S. Haddad, and J-M. Ilié. Exploiting symmetry in linear time temporal logic model checking : One step beyond. *Lecture Notes in Computer Science*, 1384, 1998.

[AL88] M Abadi and L. Lamport. The existence of refinement mappings. In *3th IEEE Logic in Computer Science*, pages 165–175, Edinburgh, 1988.

[AL91] M Abadi and L. Lamport. The existence of refinement mappings. *Theorical Computer Science*, 82(2) :253–284, 1991.

[AM97] J.-R. Abrial and L. Mussat. Specification and design of a transmission protocol by successive refinements using B. École d'été de Marketoberdorf, 1997.

[AM98] J.-R. Abrial and L. Mussat. Introducing dynamic constraints in B. In *Second Conference on the B method*, volume 1393 of *LNCS*, pages 83–128. Springer Verlag, April 1998.

[AN82] A. Arnold and M. Nivat. Comportements de processus. In *Actes du Colloque AFCET - Les Mathématiques de l'Informatique*, pages 35–68, 1982.

[And95] H.R. Andersen. Partial model checking (extended abstract). In *LICS'95*, pages 398–407. IEEE Computer Society Press, June 1995.

[Arn92] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants.* Collection Etudes et Recherches en Informatiques. Masson, Paris, 1992.

[AS85] B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4) :181–185, 1985.

[AS87] B. Alpern and F. B. Schneider. Recognizing safety and liveness. *Distributed Computing*, 2 :117–126, 1987.

[AS89] B. Alpern and F. B. Schneider. Verifying temporal properties without temporal logic. *ACM Trans. Prog. Lang. Sys.*, 11 :147–167, 1989.

[AS02] H. A. Andrade and B. Sanders. An approach to compositional model checking. In *International Parallel and Distributed Processing Symposium (IPDPS'02)*, Fort Lauderdale, Florida, April 2002. IEEE.

[Ate] Clerasy – Atelier B. http ://www.atelierb.societe.com.

[ATTa] ATT Labs-Research – graphviz. http ://www.research.att.com/sw/tools/ graphviz/refs.html.

[ATTb] ATT Labs-Research – grappa – a java graph package. http ://www.research.att.com/ ∼john/Grappa/.

[Att02] J.C. Attiogbé. Communicating B abstract systems. Research Report RR-IRIN 02.08, december 2002. updated july 2003.

[AY01] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. *ACM Trans. Prog. Lang. Syst.*, 23(3) :273–303, 2001.

[B4f] Clearsy – B4free. http ://www.b4free.com.

[Bac78] R. J. Back. *On the correctness of refinement in program development.* PhD thesis, University of Helsinki, 1978.

[Bac88] R. J. Back. A calculus of refinements for program derivations. *Acta Informatica*, (25) :593–624, 1988.

[BBC+98] N. Bjørner, A. Browne, M. Colón, B. Finkbeiner, Z. Manna, M. Pichora, H.B. Sipma, and T.E. Uribe. *STeP - The Stanford Temporal Prover - Educational Release - User's Manual.* Computer Science Department - Stanford University, Stanford, California, July 1998.

[BC00] D. Bert and F. Cave. Construction of finite labelled transition systems from B abstract systems. In T. Santen W. Grieskamp and B. Stoddart, editors, *Integrated Formal Methods (IFM'00)*, volume 1945 of *LNCS*, pages 235–255, Germany, November 2000. Springer Verlag.

[BCL02] F. Bouquet, S. Colin, and B. Legeard. Génération automatique de scénarii de test à partir de modèles b – ste visibilité générique. Rapport confidentiel TR-02-02, LIFC, - Université de Franche-comté, 2002.

[BCM+90] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking : 10^{20} states and beyond. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[BDJK00] F. Bellegarde, C. Darlot, J. Julliand, and O. Kouchnarenko. Reformulate dynamic properties during B refinement and forget variants and loop invariants. volume 1878 of *LNCS*, pages 230–249. Springer Verlag, September 2000.

[BDJK01] F. Bellegarde, C. Darlot, J. Julliand, and O. Kouchnarenko. Reformulation : a way to combine dynamic properties and B refinement. In *Formal Method Europe (FME'01)*, volume 2021 of *LNCS*, pages 2–19. Springer Verlag, March 2001.

[BDJM05] D. Bergamini, N. Descoubes, C. Joubert, and R. Mateescu. Bisimulator : A modular tool for on-the-fly equivalence checking. In *11th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2005)*, Edinburgh, Scotland, April 2005.

[BIJ97] G. Booch, Jacobson I., and Rumbaugh J. *Unified Modeling Language User Guide*. Addison Wesley, 1997.

[BJK00] F. Bellegarde, J. Julliand, and O. Kouchnarenko. Ready-simulation is not ready to express a modular refinement relation. In *Fundamental Aspects of Software Engineering (FASE'00)*, volume 1783 of *LNCS*, pages 266–283. Springer Verlag, April 2000.

[BJM99] F. Bellegarde, J. Julliand, and H. Mountassir. Model-based verification through refinement of finite B event systems. In *B User Group Meeting, Formal Methods (FM'99)*, CD-ROM publication, 1999.

[BKS83] R. J. Back and R. Kuirji-Suonio. Decentralisation of process nets with centralised control. In *2nd ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing*, pages 131–142, 1983.

[BLA+99] G. Behrmann, K. G. Larsen, H. R. Andersen, H. Hulgaard, and J. Lind-Nielsen. Verification of hierarchical state/event systems using reusability and compositionality. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*. LNCS, 1999.

[BLS96] S. Bensalem, Y. Lakhnech, and H. Saidi. Powerful techniques for the automatic generation of invariants. In R. Alur and T. A. Henzinge, editors, *Eighth*

International Conference on Computer Aided Verification (CAV'96), volume 1102, pages 323–335, New Brunswick, NJ, USA, 1996. Springer Verlag.

[BQ96] O. Burkart and Y.-M. Quemener. Model-checking of infinite graphs defined by graph grammars. Publication interne 995, IRISA, Mai 1996. 28 pages.

[BSZ00] A. Browne, H. Sipma, and T. Zhang. Linking STeP whith SPIN. volume 1885 of *LNCS*, pages 181–186. Springer Verlag, September 2000.

[Bto] BCore (UK) Ltd – B-Toolkit. http ://www.b-core.com/btoolkit.html.

[But96] M. J. Butler. Stepwise refinement of communicating systems. *Science of Computer Programming*, 27(2) :139–173, 1996.

[But00] M. J. Butler. CSP2B : A practical approach to combining CSP and B. *Formal Aspects of Computing*, 12 :182–198, 2000.

[BVW98] R. J. Back and J. Von Wright. Refinement calculus : a systematic introduction. In *Graduate texts in computer sciences*. Springer Verlag, 1998.

[CDGR04a] J.-F. Couchot, D. Déharbe, A. Giorgetti, and S. Ranise. Barvey, 2004. http ://lifc.univ-fcomte.fr/ couchot/soft/barvey/.

[CDGR04b] J.-F. Couchot, D. Déharbe, A. Giorgetti, and S. Ranise. Barvey : vérification automatique de consistance de machines abstraites B. In J. Julliand, editor, *Sessions Outils, Approches Formelles dans l'Assistance au Développement de Logiciels (Afadl'04)*, pages 369–372, Besançon, France, Juin 2004.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs Workshop*, volume 131 of *LNCS*, pages 52–71. Springer, May 1981.

[CEP99] Cepsco certification working group – common electronic purse specifications – functional requirements, September 1999. version 6.3.

[CEP00] Cepsco certification working group – common electronic purse specifications – business requirements, March 2000. version 7.0.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, 1986.

[CFJ93] E. M. Clarke, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. In *5th International Conference on Computer Aided Verification (CAV'93)*, pages 450–462, London, UK, 1993. Springer-Verlag.

[CG04] J.-F. Couchot and A. Giorgetti. Analyse d'atteignabilité déductive. In J. Julliand, editor, *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'04)*, pages 269–283, Besançon, France, Juin 2004.

[CGK97] S. C. Cheung, D. Giannakopoulou, and J. Kramer. Verification of liveness properties using compositional reachability analysis. In *ESECIFSE'97*, volume 1301 of *LNCS*, pages 227–243. Springer Verlag, September 1997.

[CGP00] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 2000.

[CGP03] J.-M. Cobleigh, D. Giannakopoulou, and C. Pasareanu. Learning assumptions for compositional verification. In *9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, Warsaw, Poland, April 2003. Springer-Verlag.

[Cha03a] C. Charlet. *Raffiner pour vérifier des systèmes paramétrés*. PhD thesis, Université de Franche-comté, December 2003.

[Cha03b] M. Charpentier. Composing invariants. In K. Araki, S. Gnesi, and Mandrioli D., editors, *Formal Methods (FM'03)*, volume 2805 of *LNCS*, pages 401–421, Pisa, Italy, September 2003. Springer Verlag.

[CIP04] M. Caporuscio, P. Inverardi, and P. Pelliccione. Compositional verification of middleware-based software architecture descriptions. In *19th International Conference on Software Engineering*, Edimburgh, Scotland, UK, may 2004.

[CK99] S. C. Cheung and J. Kramer. Checking safety properties using compositional reachability analysis. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8(1) :49–78, Janvier 1999.

[CL03] S. Campos and C. Lasalle. Conception d'un outil pour vérifier le raffinement de systèmes à composants synchronisés. Dess, UFR Sciences et Techniques - Université de Franche-comté, 2003.

[CMP92] E. Chang, Z. Manna, and A. Pnueli. Characterization of temporal property classes. In *ICALP'92*, volume 623 of *LNCS*, pages 474–486, Vienne, 1992.

[Col02] S. Colin. Génération automatique de scénarii de test – application de la méthode bztt à un cas industriel – amélioration du calcul du préambule. DEA, UFR Sciences et Techniques - Université de Franche-comté, Septembre 2002.

[Col04a] I. Colin. Spécification et raffinement de systèmes à composants synchronisés : application au porte-monnaie électronique. DESS, UFR Sciences et Techniques - Université de Franche-comté, 2004.

[Col04b] I. Colin. Vérification de propriétés PLTL des systèmes à composants synchronisés. DEA, UFR Sciences et Techniques - Université de Franche-comté, Juillet 2004.

[Coq] The coq proof assistant. http ://coq.inria.fr/.

[DAC98] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Property specification patterns for finite-state verification. In *2nd Workshop on formal Methods in Software Practice (FMSP-98)*, pages 7–15, New-York, 1998. ACM Press.

[DAC99] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *21st International Conference on Software Engineering*, May 1999.

[Dar02] C. Darlot. *Reformulation et vérification de propriétés temporelles dans le cadre du raffinement de systèmes d'événements.* PhD thesis, Université de Franche-comté, December 2002.

[DB01] J. Derrick and E. Boiten. *Refinement in Z and object-Z : foundations and advanced applications.* Springer Verlag, 2001.

[Dij72] E. W. Dijkstra. Notes on structured programming. In *Structured programming*, 1972.

[Dij75] E. W. Dijkstra. Guarded commands, nondeterminancy and formal derivation of programs. In *Communications of the ACM*, volume 18, pages 453–457, 1975.

[Dij76] E. W. Dijkstra. *A discipline of Programming.* Prentice-Hall, 1976.

[DJK03] C. Darlot, J. Julliand, and O. Kouchnarenko. Refinement preserves PLTL properties. In D. Bert, J. P. Bowen, S. C. King, and M. Walden, editors, *Formal Specification and Development in Z and B (ZB'2003)*, volume 2651 of *LNCS*, Turku, Finland, June 2003. Springer Verlag.

[DNV95] R. De Nicola and F. Vaandrager. Three logics for branching bisimulation. *Journal of ACM*, 42(2) :458–487, 1995.

[DS04] J. Derrick and G. Smith. Linear temporal logic and Z refinement. In Shankland C. Rattray C., Maharaj S., editor, *Algebraic Methodology and Software Technology (AMAST'04)*, volume 3116 of *LNCS*, pages 117–131, Stirling, Scotland, July 2004. Springer Verlag.

[DW96] J. Davies and J. C. P. Woodcock. *Using Z : Specification, Refinement and Proof.* Prentice-Hall, 1996.

[EH82] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. In *Symp. Theory of Computing (STOC'82)*, pages 169–180, May 1982.

[EH86] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited : on branching versus linear time temporal logic. *Journal of the ACM*, 33(1) :151–178, Janvier 1986.

[EH00] K. Etessami and G. J. Holzmann. Optimizing büchi automata. In *11th Int. Conf. on Concurrency Theory (CONCUR'2000)*, volume 1877 of *LNCS*, pages 153–167, University Park, PA, USA, Août 2000. Springer-Verlag.

[EL85] E. A. Emerson and C.-L. Lei. Modalities for model checking : Branching time strikes back. In *Principles of Programming Languages*, pages 84–96, 1985.

[ES93] F. A. Emerson and A. P. Sistla. Symmetry and model checking. In *5th Conference on Computer Aided Verification (CAV'93)*, June 1993.

[ES96] F. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2) :105–131, August 1996.

[FDR93] *Failures-Divergence Refinement. User Manual and Tutorial*, formal systems (europe) edition, August 1993.

[FDR97] *Failures-Divergence Refinement. FDR2 user manual*, formal systems (europe) edition, October 1997.

[Fer89a] J.-C. Fernandez. Aldebaran : A tool for verification of communicating processes. Rapport SPECTRE C14, Institut IMAG, Laboratoire de Génie Informatique - Grenoble, September 1989.

[Fer89b] J.-C. Fernandez. Aldebaran user's manual. Technical report, Institut IMAG, Laboratoire de Génie Informatique - Grenoble, January 1989.

[Fer90] J.-C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of computer programming*, 13(2-3) :219–236, May 1990.

[FFQS05] C. Flanagan, S. N. Freundb, S. Qadeerc, and S. A. Seshiad. Modular verification of multithreaded programs. *Theoretical Computer Science*, 338 :153–183, June 2005.

[FGM+92] J.-C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodríguez, and J. Sifakis. A toolbox for the verification of lotos programs. In *14th International Conference on Software Engineering (ICSE'92)*, pages 246–259, Melbourne, Australia, May 1992.

[FKM93] J.-C. Fernandez, A. Kerbrat, and L. Mounier. Symbolic equivalence checking. In *5th Workshop on Computer-Aided Verification*, Heraklion, Greece, June 1993.

[FM90] J.-C. Fernandez and L. Mounier. Verifying bisimulations "on the fly". In *3rd International Conference on Formal Description Techniques FORTE'90*, Madrid, Spain, November 1990.

[FM91a] J.-C. Fernandez and L. Mounier. "on the fly" verification of behavioural equivalences and preorder. In *3rd Workshop on Computer-Aided Verification*, Aalborg, Denmark, July 1991.

[FM91b] J.-C. Fernandez and L. Mounier. A toolset for deciding behavioral equivalences. In *CONCUR'91*, Amsterdam, The Netherlands, August 1991.

[FQ03] C. Flanagan and S. Qadeer. Thread-modular model checking. In *SPIN Workshop on Software Verification*, 2003.

[Gab89] D. M. Gabbay. The declarative past and imperative future. In H. Barringer, editor, *Colloquium on Temporal Logic and Specifications*, volume 398 of *LNCS*, pages 409–448. Springer Verlag, 1989.

[GB92] D. M. Gabbay and H. Barringer. The imperative future : Past successes implies future action : A survey position paper. In Y. N. Moschovakis, editor, *Logic from Computer Science*, pages 1–16. Springer-Verlag, 1992.

[GJM+97] H. Garavel, M. Jorgensen, R. Mateescu, C. Pecheur, M. Sighireanu, and B. Vivien. Cadp'97 - status, applications, and perspectives. In *International Workshop on Applied Formal Methods in System Design*, Zagreb, Croatia, June 1997.

[GL02a] H. Garavel and F. Lang. NTIF : A general symbolic model for communicating sequential processes with data. Technical Report RR-4666, INRIA, December 2002.

[GL02b] D. Giannakopoulou and F. Lerda. From states to transitions : Improving translation of ltl formulae to büchi automata. In *FORTE 2002*, LNCS. Springer Verlag, November 2002.

[GLM02] H. Garavel, F. Lang, and R. Mateescu. An overview of CADP 2001. In *European Association for Software Science and Technology (EASST)*, volume 4, pages 13–24, August 2002.

[GO01] P. Gastin and D. Oddoux. Past ltl to buchi automata translation. In *CAV'01*, volume 2102 of *LNCS*, pages 53–65. Springer Verlag, 2001.

[GPC04] D. Giannakopoulou, C. S. Pasareanu, and J. M. Cobleigh. Assume-guarantee verification of source code with design-level assumptions. In *26th International Conference on Software Engineering (ICSE'04)*, pages 211–220, Washington, DC, USA, 2004. IEEE Computer Society.

[GPSJ80] D. M. Gabbay, A. Pnueli, S. Shelah, and Stavi J. On the temporal analysis of fairness. In *7th Annual ACM Symposium on Principles of Programming Languages*, pages 163–173, Las Vegas, Nevada, 1980.

[GPVW95] R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Protocol Specification Testing and Verification (PSTV'95)*, pages 3–18, Warsaw, Poland, June 1995. Chapman & Hall.

[Gra] Graphviz - graph visualization software. http ://www.graphviz.org.

[GS02] G. Goessler and J. Sifakis. Composition for component-based modeling. In *FMCO'02*, volume 2852 of *LNCS*, pages 443–466, Leiden, the Netherlands, November 2002.

[GS03a] G. Goessler and J. Sifakis. Component-based construction of deadlock-free systems. In *FSTTCS 2003*, volume 2914 of *LNCS*, pages 420–433, Mumbai, December 2003. Invited talk.

[GS03b] G. Goessler and J. Sifakis. Priority systems. In *FMCO'03*, volume 3188 of *LNCS*, pages 314–329, Leiden, the Netherlands, November 2003.

[Har87] D. Harel. Statecharts : a Visual Formalism for Complex Systems. *Journal of Science of Computer Programming*, 8 :231–274, 1987.

[Har98] D. Harel. *Modeling Reactive Systems With Statecharts*. Mac Graw Hill, 1998.

[HD04] J.-L. Haberbusch and N. Delpoux. SynCo ou CADP pour vérifier le raffinement de systèmes. DESS, UFR Sciences et Techniques - Université de Franche-comté, 2004.

[HMP01] T. A. Henzinger, M. Minea, and V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In *Fourth International Workshop on Hybrid*

Systems : Computation and Control (HSCC), volume 2034 of *LNCS*, pages 275–290. Springer Verlag, 2001.

[HN96] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *The Journal of ACM Transaction on Software Engineering and Methodologies*, 5(4) :293–333, October 1996.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall Int., 1985.

[HOL] Automated reasoning group HOL page. http ://www.cl.cam.ac.uk/Research/HVG/HOL/.

[Hol85] G. J. Holzmann. Tracing protocols. *ATT Sequential Processes*, 64(10), 1985.

[Hol97] G. J. Holzmann. The model checker SPIN. In *IEEE Trans. on Software Engineering*, volume 23, pages 279–295, May 1997.

[Hol03] G.J. Holzmann. *The SPIN Model Checker*. Addison-Wesley, 2003.

[HQR98] T. A. Henzinger, S. Qadeer, and S. K. Rajamani. You assume, we guarantee : Methodology and case studies. In *10th International Conference on Computer Aided Verification (CAV '98)*, pages 440–451, London, UK, 1998. Springer Verlag.

[HyT] HyTech : The hybrid technology tool. http ://www-cad.eecs.berkeley.edu/∼tah/HyTech/.

[ISS02a] N. Ioustinova, N. Sidorova, and M. Steffen. Abstraction and flow analysis for model checking open asynchronous systems. In L.-H. Eriksson and P. A. Lindsay, editors, *9th Asia-Pacific Software Engineering Conference (APSEC 2002)*, pages 227–235. IEEE Computer Society, December 2002.

[ISS02b] N. Ioustinova, N. Sidorova, and M. Steffen. Closing open SDL-systems for model checking with DT Spin. In L.-H. Eriksson and P. A. Lindsay, editors, *Int. Symp. on Formal Methods Europe (FME'02)*, volume 2391 of *LNCS*, pages 531–548. Springer Verlag, 2002.

[ISS04] N. Ioustinova, N. Sidorova, and M. Steffen. Synchronous closing and flow analysis for model checking timed systems. (invited paper). In *2nd International Symposium on Formal Methods for Components and Objects (FMCO'03)*, LNCS. Springer Verlag, 2004.

[JM87] F. Jahanian and A. K.-L. Mok. A graph-theoretic approach for timing analysis and its implementation. *IEEE Trans. Comput.*, 36(8) :961–975, 1987.

[Jon83] C. B. Jones. Tentative steps toward a development method for interfering programs. *ACM Trans. Program. Lang. Syst.*, 5(4) :596–619, 1983.

[Jos88] M. B. Josephs. A state-based approach to communicating processes. *Distributed Computing*, 3 :9–18, 1988.

[JT96] B. Jonsson and Y.-K. Tsay. Assumption/guarantee specifications in linear-time temporal logic. *Theoretical Computer Science*, 167 :47–72, October 1996. An extended abstract appeared earlier in TAPSOFT '95, LNCS 915.

[Kam68] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, UCLA, Los Angeles, CA, USA, 1968.

[KL] O. Kouchnarenko and A. Lanoix. SynCo : a refinement analysis tool for synchronized component-based systems. In *Tool Exhibition Notes, Formal Methods (FM'03).*

[KL03a] O. Kouchnarenko and A. Lanoix. Raffinement de systèmes à composants synchronisés. In D. Méry, N. Rezg, and X. Xie, editors, *Modélisation des systèmes réactifs (MSR'03)*, volume RS/hors série 2003 of *Hermes*, pages 225–240, Metz, France, Octobre 2003. Lavoisier.

[KL03b] O. Kouchnarenko and A. Lanoix. Refinement and verification of synchronized component-based systems. In K. Araki, S. Gnesi, and Mandrioli D., editors, *Formal Methods (FM'03)*, volume 2805 of *LNCS*, pages 341–358, Pisa, Italy, September 2003. Springer Verlag.

[KL04] O. Kouchnarenko and A. Lanoix. Verifying invariants of component-based systems through refinement. In C. Rattray, S. Maharaj, and C. Shankland, editors, *Algebraic Methodology and Software Technology (AMAST'04)*, volume 3116 of *LNCS*, pages 289–303, Stirling, Scotland, July 2004. Springer Verlag.

[KLL+97] K. J. Kristoffersen, F. Larroussinie, K. G. Larsen, P. Pettersson, and W. Yi. A compositional proof of a real-time mutual exclusion protocol. In *7th Int. Joint Conf. on the Theory and Practice of Software Development*, April 1997.

[Kou04] O. Kouchnarenko. *Raffiner pour vérifier des propriétés des systèmes finis et infinis.* Habilitation à diriger des recherches, UFR Sciences et Techniques - Université de Franche-comté, Novembre 2004.

[Koz83] D. Kozen. Results on the porpositional μ-calculus. *Theoretical Computer Science*, 27(3) :333–354, 1983.

[Kro] The tool kronos. http ://www-verimag.imag.fr/TEMPORISE/kronos/.

[KV96] O. Kupferman and M. Y. Vardi. Module checking. In Rajeev Alur and T.A. Henzinger, editors, *Eighth International Conference on Computer Aided Verification CAV*, volume 1102, pages 75–86, New Brunswick, NJ, USA, 1996. Springer Verlag.

[KV97] O. Kupferman and M. Y. Vardi. Module checking revisited. In *9th International Computer Aided Verification Conference*, pages 36–47, 1997.

[KV98] O. Kupferman and M. Y. Vardi. Modular model checking. *LNCS*, 1536 :381–401, 1998.

[KV00] O. Kupferman and M. Y. Vardi. An automata-theoretic approach to modular model checking. *ACM Trans. Program. Lang. Syst.*, 22(1) :87–128, 2000.

[Lam] L. Lamport. TLA - The Temporal Logic of Actions. http ://www.research.microsoft.com/users/lamport/tla/tla.html.

[Lam73] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions of Software Engineering*, 27(2) :333–354, 1973.

[Lam83] L. Lamport. Specifying concurrent program modules. *ACM Trans. Program. Lang. Syst.*, 5(2) :190–222, 1983.

[Lam90] L. Lamport. A Temporal Logic of Actions. Research Report 57, DEC Systems Research, Palo Alto, CA, Avril 1990.

[Lam94] L. Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3) :872–923, May 1994.

[Lam96] L. Lamport. Refinement in state-based formalisms. Src-tn-1996-001, Systems Research Center, California, December 1996.

[Lan04a] A. Lanoix. A compositional framework using refinement. In *Modelling and Verifying Parallel Processes (MOVEP'04)*, Bruxelles, Belgique, December 2004.

[Lan04b] A. Lanoix. Synco : vérification du raffinement des systèmes à composants synchronisés. In *Sessions Outils, Approches Formelles dans l'Assistance au Développement de Logiciels (Afadl'04*, pages 373–376, Besançon, France, June 2004.

[Lan04c] A. Lanoix. Vérifier le raffinement de manière compositionnelle. In *MAnifestation de Jeunes Chercheurs en STIC (MAJECSTIC'04)*, Calais, France, Octobre 2004.

[LB03] M. Leuschel and B Butler. ProB : A model checker for B. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FM'03*, LNCS, pages 855–874. Springer-Verlag, 2003.

[LBa] M. Leuschel, M. Butler, and al. ProB. http ://www.ecs.soton.ac.uk/~mal/systems/prob.html.

[LL95] F. Laroussinie and K.G. Larsen. Compositional model checking of real time systems. In *Basic Research in Computer Science*, March 1995.

[LMA01] M. Leuschel, T. Massart, and Currie A. How to make fdr spin : Ltl model checking of csp by refinement. In *FME'2001*, LNCS, Berlin, Germany, March 2001.

[LMM99a] D. Latella, I. Majzik, and M. Massink. Automatic verification of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing*, 11(6) :637–664, 1999.

[LMM99b] D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML statechart diagrams. In *Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, pages 331–347, Florence, Italy, February 1999. Kluwer.

[LMS02] F. Laroussinie, N. Markey, and P. Schnoebelen. Temporal logic with forgettable past. In *Logic in Computer Science (LICS'02)*, pages 383–392, Copenhagen, Denmark, july 2002. IEEE Computer Society Press.

[LNAH+01] J. Lind-Nielsen, H. R. Andersen, H. Hulgaard, G. Behrmann, K. Kristoffersen, and K. G. Larsen. Verification of large state/event systems using compositionality and dependency analysis. *Formal Methods in System Design*, 18(1) :5–23, Janvier 2001.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Principles of Programming Languages (POPL '85)*, pages 97–107. ACM Press, 1985.

[LP99] J. Lilius and I. P. Paltor. Formalising uml state machines for model checking. In R. France and B. Rumpe, editors, *UML'99 - The Unified Modeling Language*, volume 1723 of *LNCS*, pages 430–445, Fort Collins, CO, USA, October 1999. Springer Verlag.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. D. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, London, UK, 1985. Springer Verlag.

[LS95] F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. *Theoretical Computer Science*, 148(2) :303–324, september 1995.

[LS99] J. Lilius and H. Sara. An implementation of uml state machine semantics for model checking. In *11th Nordic Workshop on Programming Theory*, Sweden, October 1999.

[LT04] A. Leclerc and C. Thevenot. Inplantation d'un partitionneur pour la vérification de systèmes d'évènements spécifiés par raffinement. DESS, UFR Sciences et Techniques - Université de Franche-comté, 2004.

[Mat03] R. Mateescu. A generic on-the-fly solver for alternation-free boolean equation systems. In H. Garavel and J. Hatcliff, editors, *9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003)*, volume 2619 of *LNCS*, pages 81–96, Warsaw, Poland), April 2003. Springer Verlag.

[MBB+95] Z. Manna, N. Bjørner, A. Browne, E. Chang, M. Colón, L. de Alfaro, H. Devarajan, A. Kapur, J. Lee, H.B. Sipma, and T.E. Uribe. STeP - the stanford temporal prover. *Theory and Practice of Software Developpement*, 915 :793–794, May 1995.

[MC81] J. Misra and K. M. Chandy. Proofs of networks of processes. *IEEE Transactions on software Engineering*, 7(4) :417–426, July 1981.

[McM00] K. L. McMillan. A methodology for hardware verification using compositional model-checking. *Science of Computer Programming*, 37 :279–309, 2000.

[Mil71] R. Milner. An algebraic definition of simulation between programs. In *2nd Int. Joint conf. on Artificial Intelligence*, pages 481–489, 1971.

[Mil89] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[MLS97] E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical automata as model for statecharts. In *Third Asian Computing Science Conference on Advances in Computing Science (ASIAN'97)*, pages 181–196, London, UK, 1997. Springer Verlag.

[MLSH97] E. Mikk, Y. Lakhnech, M. Siegel, and G. J. Holzmann. Implementing statecharts in promela/spin. Technical report, 1997.

[Mor87] J. M. Morris. A theoretical basis for stepwise refinement and programming calculus. *Science of computer programming*, 9 :287–306, 1987.

[Mor90] C. Morgan. *Programming from specifications*. Prentice-Hall, 1990.

[MP90] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *ninth annual ACM symposium on Principles of distributed computing (PODC'90)*, pages 377–410. ACM Press, 1990.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems : Specifications*. Springer Verlag, 1992.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems : Safety*. Springer Verlag, 1995.

[MPS04] X. Morselli, M.-L. Potet, and N. Stouls. Génésyst : Génération d'un système de transitions étiquetées à partir d'une spécification B événementiel. In J. Julliand, editor, *Sessions Outils, Congrès Approches Formelles dans l'Assistance au Développement de Logiciels, AFADL'04*, Besançon, France, June 2004.

[NT00] K. S. Namjoshi and R. J. Trefler. On the completness of compositional reasoning. In E. A. Emerson and A. P. Sistla, editors, *Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 139–153, Chicago, July 2000. Springer Verlag.

[Par81] D. Park. Concurrency and automata on infinite sequences. In *5th GI Conf. on Theorical Computer Science*, LNCS, pages 167–183. Springer Verlag, 1981.

[PDH99] C. S. Pasareanu, M. B. Dwyer, and M. Huth. Assume-guarantee model checking of software : A comparative case study. In *Theorical and Pratical Aspect of SPIN Model Checking*, volume 1680 of *LNCS*. Springer Verlag, 1999.

[Pnu77] A. Pnueli. The temporal logic of programs. In *18th IEEE Symposium Foundations of Computer Science (FOCS 1977)*, pages 46–57, 1977.

[Pnu81] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13 :1–20, 1981.

[Pnu85] A. Pnueli. In transition from global to modular temporal reasoning about programs. pages 123–144, 1985.

[Pnu92] A Pnueli. System specification and refinement in temporal logic. In *12th Conf. Found. of Software Technology and Theor. Comp. Sci.*, number 652 in LNCS, pages 1–38, New Delhi, India, 92. Springer Verlag.

[PS04] M.-L. Potet and N. Stouls. Explicitation du contrôle de développement B événementiel. In J. Julliand, editor, *Congrès Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'04)*, Besançon, France, Juin 2004.

[PT87] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6) :973–989, 1987.

[PVS] PVS specification and verification system. http ://pvs.csl.sri.com/.

[QS82] J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems is CESAR. In *International Symposium on Programming*, volume 137 of *LNCS*, pages 337–351. Springer Verlag, 1982.

[Rab01] A. Rabinovich. On compositional method and its limitations. Technical report, University of Edinburgh, Research Report EDI-INF-RR-0035, 2001.

[Ros97] A.W. Roscoe. *The theory and Practice of Concurrency.* Prentice-Hall, 1997.

[S+99] P. Schnoebelen et al. *Vérification de logiciels - Techniques et outils du model-checking.* Vuilbert, 1999.

[SB00] F. Somenzi and R. Bloem. Efficient buechi automata from LTL formulae. In *CAV'00*, pages 248–263. Springer Verlag, 2000.

[Sie97] M. Siegel. Relaxing property preservation in the refinement of concurrent systems. In *2nd BCS-FACS Northern Formal Methods Workshop*, Ilkley, UK, July 1997.

[Sif05] J. Sifakis. A framework for component-based construction. In *3rd IEEE International Conference on Software Engineering and Formal Methods (SEFM05)*, pages 293–300, Koblenz, September 2005. Keynote talk.

[Sis85] A. P. Sistla. On characterization of safety and liveness properties in temporal logic. In *Fourth annual ACM symposium on Principles of distributed computing*, pages 39–48, Minaki, Ontario, Canada, August 1985.

[Sis94] A. P. Sistla. Safety, liveness and fairness in temporal logic. *Formal Aspects in Computing*, 6 :495–511, 1994.

[SMV] Model checking @CMU – the SMV system. http ://www-2.cs.cmu.edu/ modelcheck/smv.html.

[SPI] On-the-fly LTL model checking with Spin. http ://spinroot.com/spin.

[Spi92] J. M. Spivey. *The Z notation : a reference manual.* Prentice-Hall, Inc., 1992. 2nd edition.

[SS02] N. Sidorova and M. Steffen. Synchronous closing of timed SDL systems for model checking. In A. Cortesi, editor, *Third Int. Wsh on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 2294 of *LNCS*, pages 79–93. Springer Verlag, 2002.

[ST02] S. Schneider and H. Treharne. Communicating B machines. In D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, editors, *Formal specification and development in Z and B (ZB 2002)*, volume 2272 of *LNCS*, pages 416–435. Springer Verlag, 2002.

[SW03] G. Smith and K. Winter. Proving temporal properties of Z specifications using abstraction. In D. Bert, J. P. Bowen, S. C. King, and M. Walden, editors, *Formal Specification and Development in Z and B (ZB'2003 :)*, volume 2651 of *LNCS*, pages 408–420, Turku, Finland, June 2003. Springer Verlag.

[Tar83] R. E. Tarjan. *Data structures and network algorithms.* SIAM, Philadelphia, 1983.

[TS99] H. Treharne and S. Schneider. Using a process algebra to control B OPERATIONS. In *1st International Conference on Integrated Formal Methods (IFM'99)*, pages 437–457, York, 1999. Springer Verlag.

[Tsa00] Y.-K. Tsay. Compositional verification in linear-time temporal logic. In J. Tiuryn, editor, *3rd Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS 2000)*, volume 1784 of *LNCS*, pages 344–358. Springer Verlag, 2000.

[TT03] J.-W. Teng and Y.-K. Tsay. Composing temporal-logic specifications with machine assistance. In K. Araki, S. Gnesi, and Mandrioli D., editors, *Formal Methods (FM'03)*, volume 2805 of *LNCS*, Pisa, Italy, September 2003. Springer Verlag.

[Upp] Uppaal home. http ://www.uppaal.com.

[Var88] M. Y. Vardi. A temporal fixpoint calculus. In *15th ACM symposium on Principles of programming languages (POPL'88)*, pages 250–259. ACM Press, 1988.

[Var94] M. Y. Vardi. Nontraditional applications of automata theory. In M. Hagiya and J. C. Mitchell, editors, *Theoretical Aspects of Computer Software*, pages 575–597. Springer Verlag, Berlin, Heidelberg, 1994.

[VAS] VASY. Construction and analysis of distributed processes (CADP). http ://www.inrialpes.fr/vasy/cadp/.

[vG90] R.J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR'90*, pages 278–297. Springer-Verlag, 1990.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *First IEEE Symp. on Logic in Computer Science*, pages 322–331, 1986.

[VW94] M. Y. Vardi and P. Wolper. Reasonning about infinite computations. *Information and Computation*, 115(1) :1–37, 1994.

[WVS83] P. Wolper, M. Y. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. In *24th IEEE Symp. on Foundation of Computer Science*, pages 185–194, Tuscan, 1983.

Résumé : L'augmentation en taille et en complexité des systèmes réactifs font que leur vérification est de plus en plus difficile à comprendre et à appréhender. Dans cette thèse, une approche est proposée pour spécifier et vérifier compositionnellement certains de ces systèmes.

Cette approche est basée sur un principe de décomposition supportant un raffinement compositionnel au niveau des composants et au niveau de leur produit synchronisé : une méthode est présentée pour vérifier le raffinement d'un système à composants à partir du raffinement de ses composants.

Les propriétés LTL sont préservées par le raffinement compositionnel présenté ici. De plus, certaines propriétés – comme les invariants et les propriétés LTL de sûreté – peuvent être vérifiées compositionnellement durant la phase de vérification du raffinement.

Un outil, nommé SynCo, implante cette approche de vérification compositionnelle. Les différents aspects de ce travail sont illustrés par plusieurs exemples : un robot industriel, un système d'essuyage et un porte-monnaie électronique.

Abstract : The increasing size and complexity of reactive systems make their verification more and more difficult to understand as well as to handle. In this thesis, an approach is proposed to specify and to verify some of these systems in a compositional way.

This approach is based on a decomposition principle that supports a compositional refinement for both components and their synchronized product : a method is presented in order to verify the refinement of a component-based system based on the weak refinement of its components.

LTL properties are preserved through the compositional refinement presented here. Furthermore, some properties – like invariants and LTL safety properties – can be compositionally verified during the refinement verification process.

An analysis tool, called SynCo, implements this compositional verification approach. The different aspects of this work are illustrated by some examples : an industrial robot, a windscreen wipers system and an electronic purse.