Techniques modulo pour les bisimulations

Damien Pous

▶ **To cite this version:**

Damien Pous. Techniques modulo pour les bisimulations. Informatique [cs]. ENS Lyon, 2008. Français. NNT : . tel-01441480

HAL Id: tel-01441480
https://hal.science/tel-01441480

Submitted on 19 Jan 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 453 N° attribué par la bibliothèque : 07ENSL0453

Thèse

en vue d'obtenir le grade de

Docteur de l'Université de Lyon – École Normale Supérieure de Lyon
spécialité : Informatique

Laboratoire de l'Informatique du Parallélisme
École doctorale de Mathématiques et Informatique Fondamentale

présentée et soutenue publiquement le 22 février 2008 par

Damien Pous

Techniques modulo pour les bisimulations

Directeurs de thèse :	Daniel	Hirschkoff
	Pierre	Lescanne
Après avis de :	Gérard	Boudol
	Vincent	van Oostrom

Devant la commission d'examen formée de :

Gérard	Boudol	(membre/rapporteur)
Cédric	Fournet	(membre)
Daniel	Hirschkoff	(membre)
Pierre	Lescanne	(membre)
Vincent	van Oostrom	(membre/rapporteur)
Davide	Sangiorgi	(membre/président)

à coulisse

Remerciements

Merci à Daniel Hirschkoff, pour la disponibilité exceptionelle dont il a fait preuve tout au long de cette thèse. Tout en me laissant une liberté fantastique dans mes recherches, Daniel a su me guider, m'encourager, et me corriger. Il m'a de plus transmis une vision et une éthique de la recherche que j'espère conserver longtemps. Sans lui, je n'aurai pas éprouvé autant de plaisir à travailler lors de ces trois années, et je n'aborderais pas aujourd'hui avec autant d'enthousiasme le monde de la recherche.

Merci à Pierre Lescanne, qui a accepté d'endosser le rôle ingrat du directeur administratif, et qui m'a indiqué de nombreuses références tout au long de ma thèse, toujours en lien direct avec mon travail.

Merci à Gérard Boudol et Vincent van Oostrom, qui ont relu cette thèse avec une attention inespérée, et dont les nombreux commentaires m'ont permis d'améliorer très nettement ce manuscrit.

Merci à Davide Sangiorgi : cette thèse n'est que le prolongement de son travail préliminaire sur les techniques modulo. Je le remercie aussi pour ses conseils, et pour les chouettes séjours à Bologne, qui ont chaque fois été extrêmement enrichissants.

Merci à Cédric Fournet et ces cinq personnes, qui m'ont fait l'honneur de former le jury.

Merci à Tom Hirschowitz, « ministre de l'alpha-conversion », pour les discussions fort stimulantes que nous avons pu avoir, dont ont découlé certains résultats présentés dans cette thèse.

Merci à Jean-Pierre Jouannaud, dont les cours de DEA sur la terminaison m'ont passioné, puis fortement influencé lors de ces trois années de thèse. Plus généralement, merci à tous les professeurs qui m'ont peu à peu tordu les neurones. En particulier Jacques Sauloy, qui n'y est pas allé de main morte ; son premier cours d'informatique théorique, littéralement envoûtant, est certainement à l'origine de mon orientation vers ce domaine.

Merci inversement à tous les étudiants que j'ai tenté de corrompre en travaux dirigés : votre perspicacité et votre ténacité m'ont forcé à clarifier de

nombreux points que je croyais acquis, et dont la compréhension m'a souvent permis d'avancer dans mes recherches.

Merci à Sylvie Boyer et Serge Torres, qui furent toujours disponibles et souriants.

Merci à Lydia Bézard, pour le prêt entre bibliothèques.

Merci à Éric Thierry (prononcez « treillis »).

Merci à Sylvain Périfel, conseiller ès méandres administratifs.

Merci à Lionel Vaux, camarade Marseillais.

Merci à Piotr, Mathilde et Melaine, comité d'accueil de Montrouge.

Merci à Yvon et Marlène, pour la menthe et les câpres.

Merci à Raphaële et Antoine, pour leur soutien inconditionnel.

Merci à Forest et Julien.

Merci à mes parents.

Merci à Julie.

Continuer ici l'énumération des personnes envers lesquelles je suis reconnaissant serait indécent ; merci à vous tous qui, m'ayant approché, me façonnez.

Sommaire

Chapitre 1

Introduction

Cette thèse s'inscrit dans le domaine de la *théorie de la concurrence* : l'étude *systèmes interactifs*, composés de plusieurs parties capables d'interagir et d'évoluer de manière indépendante. Une motivation provient du calcul parallèle : « comment utiliser plusieurs machines pour effectuer plus rapidement un calcul ? quelles primitives utiliser pour écrire des programmes destinés à être distribués sur un ensemble donné de machines ? ». Mais cette théorie permet aussi de s'intéresser à des systèmes biologiques : « comment modéliser certaines interactions entre organismes, molécules, ou protéines ? » ; ou de se pencher sur des questions moins liées à la notion de calcul : « comment définir et vérifier un protocole d'authentification ? ». Cette branche de l'informatique prend forme dans les années 1960, notamment avec les *réseaux de Petri* (Carl). Divers modèles ont été étudiés depuis, correspondant à des approches et des objectifs très différents. Par exemple, les *machines à mémoire parallèle* sont des machines abstraites qui permettent d'étudier des notions de complexité, tandis que la librairie *MPI*, plus pragmatique, fournit des primitives permettant d'écrire des programmes distribués qui communiquent par le biais de messages.

Pour notre part, c'est aux *algèbres de processus* que nous allons nous intéresser, qui permettent d'étudier formellement diverses primitives liées à la concurrence (synchronisation, distribution, mobilité, etc...). Cette notion est générique, il s'agit d'un moyen de représenter des *comportements* : à partir de processus de base, correspondant à des comportements atomiques (par exemple, l'émission d'une note, l'écriture d'un caractère), on représente des comportements plus complexes à l'aide de diverses constructions (composition séquentielle : « je chante puis j'écris une lettre » ; composition parallèle : « j'écris une lettre en chantant » ; choix exclusif : « soit je chante, soit j'écris une lettre » ; ...).

Indépendamment de la nature des processus considérés (réseaux de Petri, programmes, librairies, ensemble de molécules, auteur-interprète), la question suivante se pose naturellement : deux processus donnés dénotent-ils le

même comportement ? Par exemple, une machine est-elle équivalente à sa spécification ? un programme optimisé est-il équivalent au programme initial ? deux molécules ont-elles la même fonction ? Il nous faut ainsi définir une notion d'*équivalence*, et trouver des techniques permettant de prouver en pratique l'équivalence de deux processus donnés. Ce sont ces techniques de preuve que nous étudions dans cette thèse.

Dans la première section de ce chapitre introductif, nous montrons par une succession d'exemples standards, comment aboutir à l'équivalence *comportementale* couramment adoptée : la *bisimilarité*. Cette équivalence est comportementale car elle ne dépend pas de la façon dont sont décrits les processus : seul le comportement qu'ils dénotent intervient. Nous n'introduisons donc des processus qu'à la section 1.2, où nous définissons une algèbre de processus un peu simpliste, qui permet d'illustrer de manière plus concrète la méthodologie correspondante, et qui nous donne ensuite de la matière pour justifier l'utilité des *techniques modulo*, dans la section 1.3. Ces techniques y sont décrites dans le cas dit « fort », où elles ne posent pas de problème particulier ; c'est lorsque l'on passe au cas « faible », décrit à la section 1.4, que des irrégularités apparaissent : certaines techniques ne sont plus valides, et les techniques restantes ne sont pas toujours suffisantes. Dans la section 1.5, nous décrivons brièvement les contributions de cette thèse, dont un plan est finalement donné en dernière section.

Le lecteur déjà convaincu des bienfaits des algèbres de processus et de la bisimilarité pourra lire les deux premières sections en diagonale : mise à part la définition de l'algèbre de processus sur laquelle nous nous appuierons dans la suite de cette introduction, il n'y trouvera rien qui ne soit pas déjà présent sous une forme ou une autre dans les diverses références citées ci-dessous. Les deux sections suivantes, qui justifient les techniques modulo et décrivent la situation dans le cas faible, n'apportent rien non plus de très neuf ; mais elles permettent de mettre en place une approche et une terminologie qui nous sont parfois propres.

Repères bibliographiques et historiques

Les travaux fondateurs dans le domaine des algèbres de processus sont le *Calcul des Systèmes Communiquant (CCS)* de Robin Milner [Mil80] et les *Processus Séquentiels Communiquants (CSP)* de Tony Hoare [Hoa78]. Ces travaux ont donné lieu la publication de plusieurs livres dans la décennie qui suivit, sur lesquels nous nous appuyons fortement : Hoare publie [Hoa85] ; Matthew Hennessy insiste dans [Hen88] sur la présentation algébrique des processus, dont Milner fut le premier défenseur ; Milner publie un second livre sur CCS [Mil89], où il définit la *bisimilarité* à l'aide des outils développés indépendamment par David Park [Par81], en théorie des automates. Avec Joachim Parrow et David Walker, Milner crée ensuite le π-calcul [MPW92],

qui connut un succès considérable. Davide Sangiorgi réalise de nombreuses avancées dans la théorie de ce calcul, collectées dans un livre encyclopédique [SW01] (pour une introduction au π-calcul, on préférera celui de Milner [Mil99], qui nous semble être un très bon point d'entrée dans le domaine des algèbres de processus).

N'oublions pas l'école hollandaise, dont, entre autres : Jan A. Bergstra et Jan Willem Klop, qui ont travaillé dès le début sur les aspects algébriques de la théorie des processus [BK84, BK85] ; Rob J. van Glabbeek, à qui l'on doit l'analyse exhaustive de toutes les équivalences comportementales imaginées (imaginables), dans son « spectre du temps linéaire et du temps arborescent » [vG90, vG93] ; et plus récemment, Wan Fokkink, dont l'*introduction aux algèbres de processus* [Fok00] est d'une clarté remarquable.

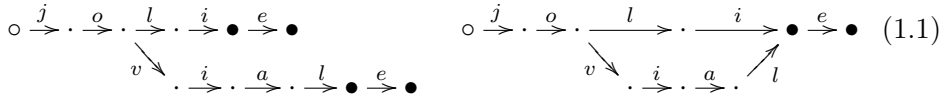
Les techniques modulo apparaissent dès le second livre de Milner [Mil89], elles ont ensuite été étudiées par Sangiorgi [SM92, San98, SW01]. Nous détaillerons ces références une fois la terminologie et les problèmes des techniques modulo introduits.

1.1 Interaction, équivalences comportementales

Tant que l'on considère des programmes qui attendent un argument, puis calculent un résultat, on a généralement une notion d'équivalence très naturelle : deux programmes sont égaux s'ils correspondent à la même fonction, i.e., s'ils associent à tout argument donné le même résultat. Il suffit donc dans ce genre de langages de définir les notions d'argument et de résultat, puis d'associer à toute phrase du langage (à tout programme) la fonction mathématique à laquelle elle correspond. Nous allons cependant voir que cette approche atteint ses limites lorsque l'on souhaite parler de « systèmes interactifs », où la notion de résultat n'est pas claire : c'est à l'ensemble des interactions possibles avec le système que l'on s'intéresse, à son « comportement ».

Interprétation fonctionnelle

Considérons par exemple les automates : ce sont des graphes étiquetés par les lettres d'un alphabet quelconque, dont un sommet, dit *initial*, est noté (○), et dont certains, dits *acceptants*, sont notés (●). Deux exemples sont dessinés ci-dessous :



(1.1)

L'idée initiale des automates consiste à définir des « langages » : un mot est *reconnu* par un automate lorsqu'il permet de relier l'état initial à l'un des états acceptants, en respectant les arêtes du graphe ; l'ensemble des mots

reconnu par un automate forme alors le *langage* reconnu par cet automate. Les deux automates précédents reconnaissent le même langage, et peuvent donc être considérés comme équivalents :

$$\{joli, jolie, jovial, joviale\} \quad .$$

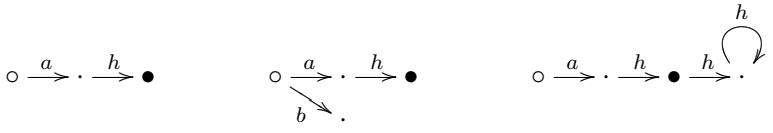
Notons que nous ne considérons que des mots finis, mais le langage reconnu peut être infini, même lorsque l'automate est fini, et que ces derniers ne sont pas nécessairement déterministes (d'un même état peuvent partir deux arêtes étiquetées par une même lettre). Ainsi, les deux automates ci-dessous, dont celui de droite n'est pas déterministe, reconnaissent le langage infini $\{a, ah, ahh, \ldots\} \cup \{a!, ah!, ahh!, \ldots\}$.

(1.2)

Cette interprétation des automates en termes de langage reconnu est « fonctionnelle » : un automate est représenté par la fonction caractéristique du langage auquel il correspond, qui à tout mot sur l'alphabet renvoie 1 ou 0 selon que le mot est reconnu ou pas. Cette sémantique repose sur une certaine idée de l'utilisation des automates : ils sont assimilés à des « boîtes noires » auxquelles on peut dicter un mot, et qui répondent oui ou non selon que le mot est reconnu ou non. Ainsi, pour un observateur extérieur, la seule possibilité pour distinguer deux automates consiste à trouver un mot reconnu par l'un mais pas par l'autre.

Traces

On peut cependant imaginer d'autres modes de fonctionnement : un automate pourrait être une « boîte noire parlante », capable de prononcer des lettres, en suivant aléatoirement les arêtes de son graphe. Il nous faut alors changer de notion d'équivalence : les trois automates ci-dessous reconnaissent le même langage, $\{ah\}$, bien qu'un observateur extérieur puisse différencier les boîtes parlantes leur correspondant : la seconde peut prononcer « b » et se bloquer, ce que ne fera jamais la première ; la troisième peut prononcer indéfiniment « h », ce que peuvent pas faire les deux premières.



(1.3)

Cette interprétation des automates ignore en fait la notion d'état acceptant : tous les états sont acceptants, ce qui nous laisse observer toutes les *traces*

(finies) possibles. On différencie bien ainsi les trois automates précédents, dont les traces sont les ensembles suivants (où ϵ représente le mot vide) :

$$\{\epsilon, a, ah\} \qquad \{\epsilon, a, b, ah\} \qquad \{\epsilon, a, ah, ahh, \ldots\}$$

Supposons maintenant que les différentes lettres de l'alphabet correspondent à des « actions » que peut effectuer une machine : afficher un caractère, émettre une note, éteindre l'écran, etc... Les automates donnent ainsi un moyen de décrire l'ensemble des exécutions possibles d'un programme non déterministe. Dans ce cadre, l'équivalence des traces semble raisonnable : deux programmes sont équivalents s'ils peuvent effectuer les mêmes séquences d'actions.

Interaction : les limites des traces

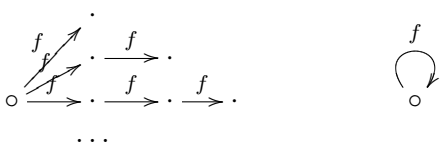
Dans notre première interprétation des automates, les lettres de l'alphabet jouent le rôle des « entrées » d'un programmes : elle sont dictées à l'automate, qui décide de son évolution en fonction de ces dernières. Dans la seconde interprétation, on a en fait inversé le sens de la communication : les lettres constituent la « sortie » du programme ; l'automate évolue en prononçant au fur et à mesure l'ensemble des décisions qu'il prend.

Une idée naturelle consiste maintenant à donner les deux possibilités en même temps : imaginons que certaines lettres correspondent à des actions qu'un observateur externe doit fournir, et que les autres lettres correspondent à des actions effectuées par la machine, qu'un tiers peut observer. On obtient ainsi des « systèmes interactifs » : l'automate est une boîte noire avec laquelle un observateur peut « communiquer ». Reprenant un exemple classique [Mil99], un distributeur de boissons peut être représenté par l'automate de gauche ci-dessous, où les lettres s, c, t correspondent respectivement aux actions de fournir un sou, d'appuyer sur le bouton « café » et d'appuyer sur le bouton « thé » – du point de vue de l'automate, ce sont des entrées – et les lettres C, T correspondent à l'action de servir respectivement un café ou un thé – toujours du point de vue de l'automate, ce sont des sorties.

(1.4)

L'équivalence des traces n'est désormais plus satisfaisante : l'automate de droite admet les mêmes traces que celui de gauche, mais on souhaite en général rejeter un distributeur de boissons correspondant à cet automate, qui ne permet pas de choisir soi-même si l'on prend un thé ou un café.

Les deux automates suivants illustrent une autre faiblesse de l'équivalence des traces : ils ont les mêmes traces (finies) : $\{\epsilon, f, ff, \ldots\}$, mais le premier finit toujours par « s'arrêter », tandis que le second peut toujours « continuer ».



(1.5)

Tant que l'on ne peut pas interagir avec les automates, la seule façon de comparer deux automates consiste à observer un comportement « global » sur l'un des deux (un mot accepté ou une trace empruntée), et à vérifier si l'autre admet ce comportement. Si l'on observe ainsi les deux distributeurs, on conclut effectivement qu'ils sont équivalents : après avoir observé la trace « stT » sur le premier, on vérifie bien que cette trace est *possible* sur le second. Lorsque l'on peut interagir, on gagne la possibilité d'accéder à l'état de l'automate après une observation : dans le cas des distributeurs de boissons, on vérifie que l'on peut introduire un sou dans les deux automates (observation de la trace « s »), mais on a ensuite la possibilité de comparer les deux automates à partir des deux états résultant de cette observation. C'est ce qui nous permet de les différencier : le premier nous laissera toujours le choix de prendre un thé ou un café, tandis que le second aura déjà effectué le choix à notre place.

Bisimilarité

La définition d'une équivalence raisonnable dans le cas de systèmes interactifs nécessite donc de se concentrer sur la notion d'état, plutôt que sur celle d'automate, qui est trop globale. Cela revient en fait à oublier l'état initial des automates, et à considérer de simple graphes étiquetés. Dénotons donc par $p, q \ldots$ les états d'un graphe étiqueté quelconque, et notons $p \xrightarrow{a} q$ lorsqu'il existe une arête étiquetée par a, menant de l'état p à l'état q (l'étiquette a pouvant correspondre à une action d'entrée comme à une action de sortie). Remarquons qu'il suffit de considérer un unique graphe : pour comparer deux automates, il suffira de considérer l'union disjointe de leurs graphes de transitions, et de comparer les états correspondant à leurs états initiaux dans ce graphe. Ainsi, pour comparer les deux distributeurs de boissons, il suffit de comparer les états p_1 et q_1 du graphe étiqueté suivant :

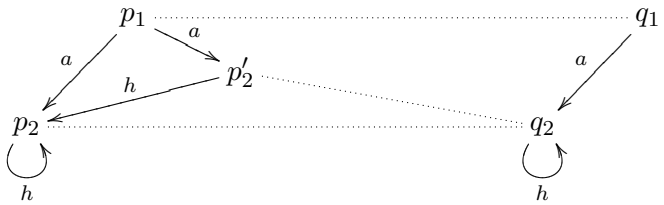
(1.4')

Suivant les intuitions précédentes, on souhaite définir une relation d'équivalence ($\sim$) entre les états, telle que deux états soient équivalents lorsqu'ils mènent, par des transitions selon les même étiquettes, à des états équivalents. On aboutit ainsi à la définition suivante : la *bisimilarité* est la plus grande relation ($\sim$) telle que si $p \sim q$, alors :

- $p \xrightarrow{a} p'$ implique $q \xrightarrow{a} q'$ pour un état q' tel que $p' \sim q'$; et
- $q \xrightarrow{a} q'$ implique $p \xrightarrow{a} p'$ pour un état p' tel que $p' \sim q'$.

(Il s'agit implicitement d'une définition co-inductive ; plus formellement, une *bisimulation* est une relation satisfaisant cette propriété, la *bisimilarité* est l'union de toutes les bisimulations, on vérifie alors que cette dernière est une bismimulation et qu'il s'agit d'une relation d'équivalence.)

Pour illustrer cette définition, on peut montrer que l'on a $p_1 \sim q_1$ dans le graphe ci-dessous, en prouvant que la relation $\{\langle p_1, q_1\rangle, \langle p_2, q_2\rangle, \langle p_2, q_2'\rangle\}$, représentée en pointillés, est une bisimulation : par exemple, lorsque p_1 effectue une transition vers p_2', q_1 peut répondre par une transition vers q_2, et les deux états ainsi obtenus sont bien mis en relation.



(1.6)

Dans cet exemple, la bisimilarité s'obtient en rajoutant le couple $\langle p_2, p_2'\rangle$, puis en prenant la fermeture réflexive symmétrique de la relation ainsi obtenue.

On associe généralement aux bisimulations le vocabulaire des jeux : les contraintes imposées par la définition de bisimulation forment les règles d'un jeu, où les états mis en relation font des *offres* et doivent y *répondre*. Dans la première clause de la définition de bisimulation par exemple, l'état de gauche (p) offre une transition selon a vers l'état p', à laquelle l'état de droite (q) doit répondre ; cette première clause décrit ainsi la partie « de gauche à droite » du jeu de bisimulation, tandis que la seconde clause décrit la partie « de droite à gauche », où les états de gauche doivent répondre aux offres des états de droite.

La bisimilarité est strictement plus fine que l'équivalence des traces : les traces partant de deux états bisimilaires sont nécessairement identiques, mais la bisimilarité distingue les deux distributeurs de boissons, ainsi que les deux derniers automates dessinés :

- pour les distributeurs (1.4), lorsque l'on joue la transition initiale selon s sur l'automate de gauche : $p_1 \xrightarrow{s} p_2$, la première clause de la définition de bisimulation demande que tout distributeur équivalent soit capable de répondre par une transition selon s vers un état équivalent. L'automate de droite peut faire deux transitions selon s, mais aucune ne

mène à un état équivalent à celui obtenu dans l'automate de gauche : l'un peut faire du thé (q_2), et l'autre du café (q_2'), mais aucun des deux ne peut choisir de faire l'un ou l'autre, comme le propose p_2.

– Pour les automates divergents (1.5), c'est en jouant à partir de celui de droite que l'on montre qu'ils ne sont pas bisimilaires : ce dernier joue sans changer d'état une transition selon f, à laquelle doit répondre celui de gauche, qui se retrouve forcé de choisir l'une de ses branches et de se faire ainsi piéger : cette branche étant finie, l'automate de droite n'a plus qu'à jouer suffisamment longtemps pour épuiser celui de gauche, qui finira par ne plus pouvoir répondre.

En termes de graphes, la bisimilarité ne fait qu'identifier les sous-graphes communs, et permet ainsi de ne pas tenir compte du « partage » potentiel de certains états. Par exemple, elle ne distingue pas les deux premiers automates que l'on a dessinés (1.1), où la seule différence provient du partage des deux états acceptants dans l'automate de droite.

Une façon courante de représenter les jeux de bisimulation consiste à dessiner des *diagrammes* : une relation R est une bisimulation si les deux diagrammes ci-dessous sont satisfaits pour toute étiquette a :

$$\begin{array}{ccc} p & R & q \\ {\scriptstyle a}\downarrow & & \vdots \\ & & {\scriptstyle a}\downarrow \\ p' & R & \exists q' \end{array} \qquad\qquad \begin{array}{ccc} p & R & q \\ \vdots & & \\ {\scriptstyle a}\downarrow & & {\scriptstyle a}\downarrow \\ \exists p' & R & q' \end{array} \tag{1.7}$$

Cette notation diagrammatique peut être rendue plus formelle : pour toutes relations binaires R, S, notons $R \cdot S$ leur composition, et notons $R \subseteq S$ lorsque R est incluse dans S ; pour toute étiquette a, l'ensemble des transitions étiquetées par a ($\xrightarrow{a}$) est une relation binaire, notons $\xleftarrow{a}$ sa converse. On a alors la caractérisation suivante : une relation R est une bisimulation si et seulement si pour toute étiquette a, $\xleftarrow{a} \cdot R \subseteq R \cdot \xleftarrow{a}$ et $R \cdot \xrightarrow{a} \subseteq \xrightarrow{a} \cdot R$; ce que l'on peut représenter par les deux diagrammes suivants, où seule la quantification universelle sur l'étiquette a est laissée implicite :

$$\begin{array}{ccc} \cdot & R & \\ {\scriptstyle a}\downarrow & \subseteq & \downarrow{\scriptstyle a} \\ & R & \cdot \end{array} \qquad\qquad \begin{array}{ccc} & R & \cdot \\ {\scriptstyle a}\downarrow & \supseteq & \downarrow{\scriptstyle a} \\ \cdot & R & \end{array} \tag{1.8}$$

Notons en particulier que l'on n'a pas besoin de parler des états pour définir bisimulation et bisimilarité : nous verrons au chapitre 3 que ces notions peuvent être définies à partir d'une axiomatisation abstraite des relations binaires.

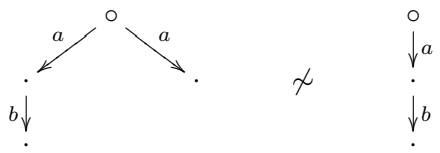
2-similarité

La bisimilarité est généralement considérée dans la littérature sur les processus comme l'équivalence comportementale la plus fine, l'équivalence des

traces remportant au contraire le titre de la plus grossière. Il existe de nombreuses équivalences intermédiaires : Rob J. van Glabbeek en étudie dix dans son premier « spectre du temps linéaire et du temps arborescent » [vG90]. Nous discutons rapidement de l'une d'entre-elles, la *2-similarité*, qui jouera un rôle particulièrement important par la suite.

Les deux clauses définissant le jeu de bisimulation sont complètement symétriques : si l'on retire la seconde, on obtient la notion de *simulation*, et une bisimulation peut alors être vue comme une simulation dont la converse est aussi une simulation.

Mais on peut aussi définir une autre équivalence : p, q sont *2-similaires* s'il existe deux simulations qui contiennent, l'une le couple $\langle p, q\rangle$, et l'autre le couple $\langle q, p\rangle$. On vérifie alors que deux processus bisimilaires sont toujours 2-similaires, mais la réciproque n'est pas vraie, comme le montre le contre-exemple standard suivant :



(1.9)

Intuitivement, en termes de jeux, la bisimilarité est très discriminante car elle permet de choisir à chaque étape du jeu le côté qui fait les offres. Au contraire, la 2-similarité est constituée de deux jeux indépendants où l'on joue toujours dans la même direction. Dans le contre-exemple, c'est ce qui permet d'offrir la branche a qui ne mène à rien, pendant la partie de gauche à droite : comme le côté droit ne posera pas de questions, le côté gauche ne se retrouvera pas coincé (dans l'autre partie, il suffit de ne pas répondre par cette branche).

Ce phénomène fait généralement de la 2-similarité une équivalence trop faible : comme illustré sur le contre-exemple, cette équivalence permet de « cacher une copie non achevée » du système : on pourrait ainsi rajouter au distributeur de boissons une transition qui accepte une pièce puis laisse choisir thé ou café, mais ne fait plus rien ensuite, sans que la 2-similarité ne s'en aperçoive.

Si la 2-similarité était satisfaisante, on pourrait immédiatement abandonner la bisimilarité, et se concentrer sur la notion plus primitive de similarité. Dans cette thèse, où l'on s'attachera justement autant que possible à ramener l'étude de la bisimulation à celle de la simulation, le couplage entre les deux jeux de simulation dont est composée la bisimulation – et qui en fait la force – nous obligera souvent à effectuer des calculs assez laborieux, qui ne seraient pas nécessaires si l'on considérait la 2-similarité.

1.2 Algèbres de processus, bisimulation

La bisimilarité ne s'intéresse pas à la façon dont est décrit le système : c'est une notion inhérente au graphe de transitions, indépendante de la nature des états et des interactions (symbolisées par les étiquettes). Définir la nature des états et des interactions, puis des règles permettant d'obtenir les transitions correspond ainsi à définir un *système de transitions étiquetées* (LTS, pour l'anglais « Labelled Transitions System »). Selon l'approche suivie pour définir ce LTS, on obtient alors un moyen de *modéliser* un système existant, ou un langage de programmation, permettant de *définir* des systèmes. Dans le premier cas, la définition des états sera plutôt descriptive : « un état est un triplet (pression,température,couleur) », et les règles de transitions seront déduites de l'observation du système modélisé ; dans le second cas, les états correspondront à l'ensemble des phrases du langage (les programmes), et ce sont les règles de transition qui définiront la sémantique du langage, en donnant la signification de chaque construction syntaxique, puis, par extension, de chaque programme.

Dans ce dernier cas, le concepteur du langage de programmation jouit d'une très grande liberté : selon le type de systèmes qu'il veut pouvoir définir, libre à lui de considérer les constructions qui lui plaisent lorsqu'il définit les ensembles d'états et d'interactions, et d'utiliser des règles aussi complexes que nécessaire lors de la définition des transitions. Nous nous prendrons à ce jeu au chapitre 6 ; pour l'instant, contentons-nous d'un LTS très simple (et peu utile en pratique), qui nous permet toutefois de montrer les problèmes qui peuvent se poser en pratique.

Une algèbre de processus minimaliste

Fixons un ensemble arbitraire d'interactions, notées $a, b \dots$ comme précédemment. On définit ensuite l'ensemble des états (que l'on appellera *processus*) par la grammaire suivante :

$$p, q, r ::= \mathbf{0} \;\mid\; a.p \;\mid\; p + q \;\mid\; p \parallel q \;\mid\; !p$$

Nous expliquons chacune de ces constructions syntaxiques ci-dessous, après avoir donné les règles d'inférence qui définissent le comportement de chacune d'entre elles (ces règles se lisent comme des implications de haut en bas : si la condition du haut est satisfaite, alors on en déduit celle du bas) :

$$[\textsc{Prf}]\ \frac{}{a.p \xrightarrow{a} p} \qquad [\textsc{Chx}_g]\ \frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \qquad [\textsc{Chx}_d]\ \frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'}$$

$$[\textsc{Par}_g]\ \frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q} \qquad [\textsc{Par}_d]\ \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'} \qquad [\textsc{Rpl}]\ \frac{p \xrightarrow{a} p'}{!p \xrightarrow{a} !p \parallel p'}$$

Un processus est soit :

- le processus *vide* (**0**), qui ne fait rien (il n'y a pas de règle d'inférence le concernant) ;
- un processus *préfixé* ($a.p$), qui propose l'interaction a, et qui devient le processus p lorsque cette interaction est jouée (règle [Prf]) ;
- un *choix* entre deux processus ($p + q$), qui se comporte comme l'un ou l'autre des deux processus p et q, selon la première interaction à fournir (règles [Chx$_g$] et [Chx$_d$]) ;
- la *mise en parallèle* de deux processus ($p \parallel q$), qui laisse p et q s'exécuter indépendamment l'un de l'autre (règles [Par$_g$] et [Par$_d$]) ;
- un processus *répliqué* ($!p$), qui correspond à la mise en parallèle d'une infinité de copies du processus p (la règle [Rpl] n'exprime pas clairement cette intuition, qui sera rapidement justifiée par la suite).

Dans la suite, on omettra le processus **0** lorsqu'il est préfixé ($a.b$ représentera ainsi le processus $a.b.\mathbf{0}$), et des priorités seront affectées aux constructions syntaxiques de telle sorte que $a.b \parallel c.d + e$ sera parenthésé comme suit : $(a.b) \parallel ((c.d) + e)$.

Les trois premières constructions permettent de représenter l'ensemble des arbres étiquetés finis : par exemple, l'état initial du premier automate que l'on a dessiné (1.1), peut être représenté par le processus suivant :

$$j.o.(l.i.e + v.i.a.l.e)$$

Si l'on se limite à ces trois constructions, on peut montrer que la bisimilarité se résume à la théorie équationnelle engendrée par les quatre lois suivantes, qui expriment l'associativité, la commutativité et l'idempotence de l'opérateur de choix, dont le processus vide est un élément neutre :

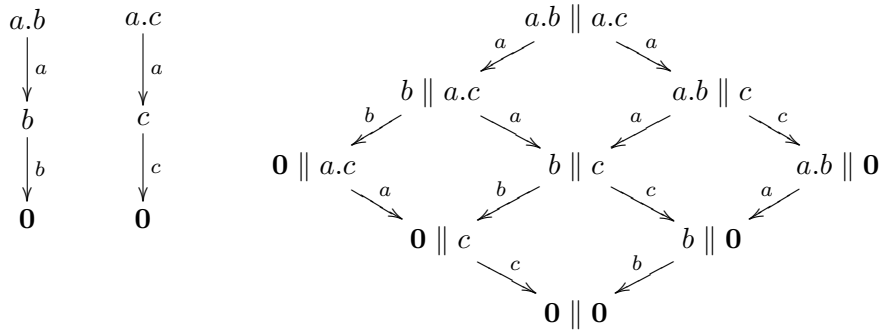
$$p + (q + r) \sim (p + q) + r \qquad p + q \sim q + p \qquad p + p \sim p \qquad p + \mathbf{0} \sim p$$

L'axiomatisation de ce fragment du calcul justifie le terme d'*algèbre* de processus : on a des générateurs (**0**, $a.$, $+$) et des axiomes. Cependant, dès que l'on passe à des systèmes plus expressifs, on perd généralement tout espoir de conserver une axiomatisation de la bisimilarité, et la notion de bisimilarité prend alors tout son intérêt.

L'opérateur de mise en parallèle est plus simple que celui que l'on trouve dans CCS [Mil80, Mil89] (ce calcul est aussi défini sur la figure 5.1 de cette thèse) : il permet simplement de considérer « l'entrelacement » de deux processus indépendants[1]. En termes de graphes de transitions, il s'agit d'un

1. La règle [Com] de la figure 5.1, qui permet à deux processus mis en parallèle de se synchroniser, est absente. Cet opérateur correspond en fait à l'opérateur « interleaving » (|||) de Hoare [Hoa85].

simple produit cartésien :



Comme annoncé, l'opérateur de réplication correspond à la mise en parallèle d'une infinité de copies d'un processus donné ; cela peut se formaliser en démontrant que pour tout processus p, on a $!p \sim !p \parallel p$. La signification précise de cet opérateur ne nous intéressera cependant pas par la suite : il nous sert simplement à obtenir des comportements non-triviaux, qui nous permettrons d'illustrer l'utilité des techniques modulo.

Contextes, congruence

Le fait de définir l'ensemble des processus par une grammaire nous permet de les considérer de façon très compositionnelle : à partir de processus élémentaires peuvent être construits des processus plus évolués ; et inversement, des systèmes complexes peuvent être analysés en se ramenant à l'étude des « briques élémentaires » dont ils sont composés.

Cela donne naturellement lieu à une notion de contexte : un *contexte* est un système dans lequel on peut placer un processus. Ainsi, tout processus peut être utilisé dans différents contextes : un distributeur de boissons peut être utilisé dans différents laboratoires ; et un contexte peut être amené à contenir différents processus : un laboratoire change parfois de distributeur de boissons. Dans ce dernier cas, on s'attend bien entendu à ce que le remplacement du distributeur par un distributeur équivalent n'affecte pas le comportement du laboratoire. Du point de vue des équivalences comportementales, cela se formalise par la propriété de congruence : une équivalence $\sim$ est une *congruence* lorsque pour tous processus p, q et tout contexte c, $p \sim q$ implique $c[p] \sim c[q]$ (où l'on note $c[p]$ le processus obtenu en plaçant le processus p dans le contexte c).

Cette propriété est fondamentale ; si nous avons ici présenté la bisimilarité comme le moyen de définir une équivalence comportementale raisonnable, c'est souvent l'approche suivante qui est suivie : on ne définit a priori ni interactions ni transitions étiquetées, mais seulement une relation de *réduction* (qui correspond à l'évolution autonome du système, sans interaction avec l'extérieur), et on choisit comme notion d'équivalence la plus grande

congruence satisfaisant certaines propriétés, dont la stabilité par réduction. L'avantage est que l'on obtient une définition plus canonique : au lieu de fixer arbitrairement les interactions d'un système avec l'extérieur, on s'appuie sur la notion de contexte, qui est interne au calcul, et la condition de congruence nous force de manière implicite à considérer l'ensemble des interactions possibles avec l'ensemble des contextes possibles. En revanche, ce type de définitions n'est pas facile à manipuler : lorsque l'on souhaite prouver l'équivalence de deux processus p, q, il nous faut démontrer que pour tout contexte c, $c[p]$ et $c[q]$ satisfont certaines propriétés ; la quantification universelle sur l'ensemble des contextes n'est pas raisonnable. Dans ce genre de situations, on cherche généralement ensuite à définir un ensemble d'interactions et de transitions étiquetées tel que la bisimilarité coïncide avec l'équivalence initiale, afin d'obtenir, comme nous allons le voir, des techniques de preuve efficaces.

Preuves par bisimulation

La définition même de la bisimilarité fournit directement une technique de preuve pour montrer l'équivalence de deux processus p, q : on a $p \sim q$ si et seulement si il existe une bisimulation qui met p en relation avec q. Afin d'illustrer cette méthode, nous allons prouver deux propriétés de la bisimilarité dans notre calcul, qui nous mèneront naturellement aux techniques modulo.

Commençons par montrer que la mise en parallèle préserve la bisimilarité (i.e., que pour tous processus p, q, r, si $p \sim q$, alors $r \parallel p \sim r \parallel q$). Il suffit pour cela de vérifier que la relation suivante est une bisimulation :

$$R \triangleq \{\langle r \parallel p,\ r \parallel q\rangle \mid \text{pour tous } r, p, q, \text{ tels que } p \sim q\} \quad .$$

Supposons que $r \parallel p \xrightarrow{a} p_0$, nous devons trouver un processus q_0 tel que $r \parallel q \xrightarrow{a} q_0$ et $p_0\ R\ q_0$. Il n'y a que deux règles qui permettent d'inférer une transition de la forme $r \parallel p \xrightarrow{a} p_0$:

[PAR$_g$] on a alors $p_0 = r' \parallel p$, avec $r \xrightarrow{a} r'$; dans ce cas, on déduit $r \parallel q \xrightarrow{a} r' \parallel q$ par la même règle, et on choisit $q_0 = r' \parallel q$, qui satisfait bien $p_0\ R\ q_0$.

[PAR$_d$] on a ici $p_0 = r \parallel p'$, avec $p \xrightarrow{a} p'$; dans ce cas, comme on a supposé $p \sim q$, on déduit q' tel que $q \xrightarrow{a} q'$ et $p \sim q'$. On prend alors $q_0 = r \parallel q'$: la règle [PAR$_d$] nous permet d'obtenir $r \parallel q \xrightarrow{a} r \parallel q'$, et on a bien $p_0\ R\ q_0$.

Introduisons une terminologie, qui sera souvent utilisée par la suite : on vient de vérifier la partie *de gauche à droite* du *jeu de bisimulation*, où les *processus de gauche* font des *offres* auxquelles les *processus de droite* doivent *répondre*. Le *candidat de bisimulation* (la relation R) étant symétrique, on a aussi implicitement vérifié la partie *de droite à gauche*, où les processus de gauche répondent aux offres des processus de droite.

Par des preuves similaires, on peut ainsi montrer les lois suivantes, qui expriment l'associativité et la commutativité de la mise en parallèle, et la neutralité du processus vide vis-à-vis de cette dernière. Les candidats de bisimulation permettant d'établir ces lois sont donnés sur la droite ; dans chaque cas, il s'agit simplement de l'ensemble des couples mis en relation par la loi. Cela est dû à la nature des règles de transitions pour l'opérateur de mise en parallèle, qui en font un opérateur « statique » [Mil89], préservé au fil des transitions ; ainsi, deux processus mis en relation par la loi d'associativité le resteront au fil des jeux de bisimulation.

$$\begin{aligned} p \parallel (q \parallel r) &\sim (p \parallel q) \parallel r & \{\langle p \parallel (q \parallel r),\ (p \parallel q) \parallel r\rangle \mid \forall p, q, r\} \\ p \parallel q &\sim q \parallel p & \{\langle p \parallel q,\ q \parallel p\rangle \mid \forall p, q\} \\ p \parallel \mathbf{0} &\sim p & \{\langle p \parallel \mathbf{0},\ p\rangle \mid \forall p\} \end{aligned}$$

Nous donnons ci-dessous les candidats de bisimulation utilisés pour montrer les lois concernant l'opérateur de choix. Ces candidats restent simples, mais prennent une forme très différente : il s'agit dans chaque cas de la relation identité, à laquelle on ajoute le singleton correspondant à une instance de la loi à prouver. Cela correspond naturellement au comportement « dynamique » [Mil89] de l'opérateur de choix, qui disparaît après une transition : deux processus mis en jeu par l'une des lois suivantes sont égalisés après une seule étape du jeu de bisimulation.

$$\begin{aligned} p + (q + r) &\sim (p + q) + r & \{\langle p + (q + r),\ (p + q) + r\rangle\} \cup \{\langle p,\ p\rangle \mid \forall p\} \\ p + q &\sim q + p & \{\langle p + q,\ q + p\rangle\} \cup \{\langle p,\ p\rangle \mid \forall p\} \\ p + \mathbf{0} &\sim p & \{\langle p + \mathbf{0},\ p\rangle\} \cup \{\langle p,\ p\rangle \mid \forall p\} \\ p + p &\sim p & \{\langle p + p,\ p\rangle\} \cup \{\langle p,\ p\rangle \mid \forall p\} \end{aligned}$$

1.3 Techniques modulo

Lorsque l'on cherche à montrer qu'une relation est une bisimulation, moins ce candidat contient de couples, moins il y a d'offres à considérer, mais plus il devient difficile de répondre à ces offres ; il y a donc un compromis à trouver. Notons qu'il y a aussi une notion de « régularité » qui intervient : bien que chacun des candidats précédemment utilisés contiennent une infinité de couples, la façon dont ils sont définis fait qu'il n'y a virtuellement qu'un couple à étudier.

L'idée des techniques modulo est d'autoriser plus de couples pour les réponses, de façon à ce qu'il soit plus facile de répondre (cela ayant pour effet secondaire la diminution du nombre d'offres à considérer : comme il sera plus facile de répondre, on pourra se permettre de mettre moins de couples dans le candidat…). On va ainsi chercher des fonctions f entre relations telles que si R est une *bisimulation modulo* f, c'est-à-dire si R satisfait les deux

diagrammes suivants, alors R est contenue dans la bisimilarité :

$$\begin{array}{ccc} \cdot & R & \\ {\scriptstyle a}\downarrow & \subseteq & \downarrow{\scriptstyle a} \\ & f(R) & \cdot \end{array} \qquad\qquad \begin{array}{ccc} & R & \cdot \\ {\scriptstyle a}\downarrow & \supseteq & \downarrow{\scriptstyle a} \\ \cdot & f(R) & \end{array} \tag{1.10}$$

Lorsque f est une fonction *extensive*, c'est-à-dire, qu'elle « agrandit » son argument, les deux diagrammes précédents sont plus faciles à vérifier que ceux définissant le jeu de bisimulation. Une telle fonction f sera dite *correcte* lorsque toute bisimulation modulo f est contenue dans la bisimilarité.

La fonction de fermeture réflexive ($\mathrm{id}^{=}_X : R \mapsto R^{=}$) dont la correction peut être prouvée très simplement, fournit par exemple une technique modulo, qui permet de ne pas avoir à rajouter soi-même la relation identité dans les candidats utilisés pour les lois sur l'opérateur de choix : chaque fois, le singleton formé de la loi elle-même est une bisimulation modulo réflexivité. Ainsi, les seules offres à considérer proviennent de la loi, et la fonction $\mathrm{id}^{=}_X$ nous permet de répondre dans $R^{=}$ plutôt que dans R, nous évitant ainsi d'avoir à rajouter tous les couples identité. Bien entendu, dans ce cas, l'usage de cette technique est plus un effet de style qu'autre chose : répondre aux offres provenant de l'identité ne rajoute pas vraiment beaucoup de travail.

Passons maintenant à une preuve où l'usage de techniques modulo est réellement utile : montrons que la réplication préserve la bisimilarité : fixons deux processus bisimilaires $p \sim q$, et montrons $!p \sim !q$. Commençons sans technique modulo, et partons du candidat singleton suivant :

$$R_1 \triangleq \{\langle !p,\ !q\rangle\} \quad .$$

Les seules offres possibles à gauche sont de la forme $!p \xrightarrow{a} !p \mid p'$, avec $p \xrightarrow{a} p'$. Pour répondre, on commence par utiliser notre hypothèse, $p \sim q$, qui nous donne q' tel que $q \xrightarrow{a} q'$ et $p' \sim q'$, et d'où on déduit $!q \xrightarrow{a} !q \parallel q'$ par la règle [Rpl]. Mais on se retrouve coincés : on n'a pas $!p \parallel p'\ R_1\ !q \parallel q' \ldots$

$$\begin{array}{ccc} !p & R_1 & !q \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle a} \\ !p \parallel p' & ? & !q \parallel q' \end{array}$$

Pour pouvoir répondre, on doit donc rajouter ce couple au candidat. En anticipant légèrement, on en rajoute au passage un peu plus, afin de pouvoir répondre à toutes les offres de $!p$ et $!q$:

$$R_2 \triangleq R_1 \cup \{\langle !p \parallel p_1,\ !q \parallel q_1\rangle \mid \forall p_1, q_1,\ p_1 \sim q_1\} \quad .$$

Mais il nous faut maintenant considérer les offres qui apparaissent avec ces nouveaux couples... Elles prennent deux formes possibles :

– par la règle [PAR$_d$], on a $!p \parallel p_1 \xrightarrow{a} !p \parallel p_1'$ dès que $p_1 \xrightarrow{a} p_1'$; ce cas est facile puisque l'on retombe sur un processus de la forme de ceux que l'on vient de rajouter : on peut répondre sans rajouter de couples.
– Par les règles [PAR$_g$] et [RPL], on a $!p \parallel p_1 \xrightarrow{a} (!p \parallel p_2) \parallel p_1$ dès que $p \xrightarrow{a} p_2$; ce cas nécessite de rajouter des couples : le candidat R_2 ne « parle » pas de processus de la forme $(!p \parallel p_2) \parallel p_1$.

On passe donc au candidat suivant :

$$R_3 \triangleq R_2 \cup \{\langle (!p \parallel p_2) \parallel p_1,\ (!q \parallel q_2) \parallel q_1 \rangle \mid \forall p_1, q_1, p_2, q_2,\ p_1 \sim q_1, p_2 \sim q_2\} \ ,$$

puis, en continuant d'ajouter ainsi les couples nécessaires, on obtient au bout d'un temps incertain la relation ci-dessous, pour laquelle il est possible, mais pénible, de montrer que c'est une bisimulation :

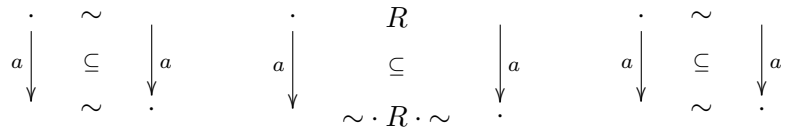
$$R_\infty \triangleq \{\langle (\dots((!p \parallel p_n) \parallel p_{n-1})\dots) \parallel p_1,\ (\dots((!q \parallel q_n) \parallel q_{n-1})\dots) \parallel q_1 \rangle \mid \forall n \in \mathbb{N},\ \forall i \leq n,\ p_i \sim q_i\} \ .$$

Nous montrons maintenant comment deux techniques modulo vont nous permettre de simplifier cette preuve en la ramenant à l'étude du candidat initial (R_1).

Modulo bisimilarité

Remarquons tout d'abord que lorsque l'on passe du candidat R_2 au candidat R_3, on ne rajoute « moralement » rien : la mise en parallèle étant associative, on a $(!p \parallel p_2) \parallel p_1 \sim !p \parallel (p_2 \parallel p_1)$ et $(!q \parallel q_2) \parallel q_1 \sim !q \parallel (q_2 \parallel q_1)$; comme elle préserve de plus la bisimilarité, on a $p_2 \parallel p_1 \sim q_2 \parallel q_1$ dès que $p_1 \sim q_1$ et $p_2 \sim q_2$. Les couples que l'on rajoute sont donc déjà présents « modulo bisimilarité » : on a $R_3 \subseteq \sim \cdot R_2 \cdot \sim$.

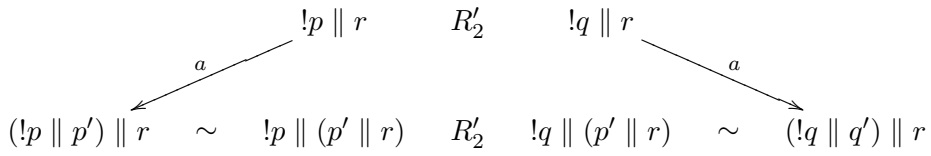
Définissons donc la fonction $\dot{\sim} : R \mapsto \sim \cdot R \cdot \sim$. Sa correction s'obtient en « assemblant » les trois diagrammes ci-dessous : si R est une bisimulation modulo $\dot{\sim}$, alors $\sim \cdot R \cdot \sim$ est une bisimulation (en utilisant la transitivité de $\sim$) ; cette dernière relation contenant en particulier R (réflexivité de $\sim$), on en déduit bien que R est contenue dans la bisimilarité.



Cette technique modulo, que l'on appellera « modulo bisimilarité » permet déjà de simplifier considérablement la preuve précédente : R_2 est une bisimulation modulo $\dot{\sim}$. En utilisant cette technique, on peut en fait simplifier encore un peu plus ce candidat : comme nous le montrons ci-dessous, la relation suivante est aussi une bisimulation modulo bisimilarité.

$$R_2' \triangleq \{\langle !p \parallel r,\ !q \parallel r \rangle \mid \forall r\} \ .$$

Le cas intéressant, où l'on utilise la technique modulo, est illustré ci-dessous ; l'offre provient d'une transition $p \xrightarrow{a} p'$, qui nous permet d'obtenir q' tel que $q \xrightarrow{a} q'$ et $p' \sim q'$ (pour montrer le résultat de bisimilarité utilisé à droite, on passe par le processus $!p \parallel (q' \parallel r)$, par transitivité).



La fonction $\dot{\sim}$ nous permet donc de raisonner modulo la bisimilarité elle-même, et d'utiliser lors des réponses des résultats préliminaires, indépendants du résultat que l'on est entrain de prouver ; ici l'associativité de la mise en parallèle, la transitivité de la bisimilarité, et le fait que cet opérateur préserve la bisimilarité.

Modulo contexte

Le candidat auquel on s'est ramené, R'_2, a une forme très particulière : les processus qu'il met en relation sont formés d'une partie « intéressante », qui n'est pas la même de part et d'autre de la relation ($!p$ et $!q$), et d'un contexte ($\parallel r$), identique de chaque côté. En conséquence, une partie de la preuve que R'_2 est une bisimulation modulo $\dot{\sim}$ est intéressante : elle consiste à vérifier que $!p$ et $!q$ ont le même comportement ; tandis qu'une autre partie de cette preuve consiste à prouver à nouveau la préservation de la bisimilarité par la mise en parallèle. Dans notre calcul, où la mise en parallèle ne permet pas aux processus de communiquer, cette preuve est suffisamment simple pour que l'on ne s'aperçoive presque pas qu'on la fait à nouveau... Ce n'est cependant pas le cas de calculs plus évolués, où la preuve de ce genre de résultats de congruence est bien souvent ardue (π-calcul d'ordre supérieur [SW01], « applicative bisimilarity »[Abr90, How96]). Il est donc utile dans ces situations d'avoir des techniques *modulo contexte*, qui permettent d'éviter qu'une preuve de bisimulation ne dégénère en une preuve de congruence.

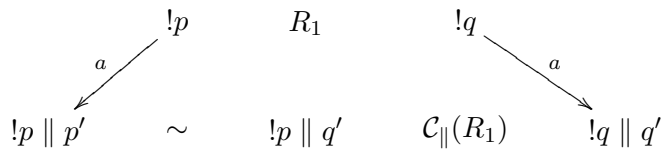
Dans notre calcul, on pourrait par exemple prouver une fois pour toutes la correction de la fonction de fermeture par contexte parallèle suivante :

$$\mathcal{C}_{\parallel} : R \mapsto \{\langle p \parallel r,\ q \parallel r\rangle \mid \forall p, q, r,\ p\ R\ q\} \quad ,$$

qui permet d'obtenir R'_2 à partir de notre singleton initial : $R'_2 = \mathcal{C}_{\parallel}(R_1)$.

On peut finalement combiner la technique modulo bisimilarité à cette dernière technique, c'est-à-dire, considérer la fonction composée $f = \dot{\sim} \circ \mathcal{C}_{\parallel}$: $R \mapsto \sim \cdot \mathcal{C}_{\parallel}(R) \cdot \sim$. On obtient ainsi une preuve élémentaire : il suffit de montrer que R_1 est une bisimulation modulo f. Reprenons donc la preuve concernant R_1 là où nous l'avions laissée : la seule offre possible (à gauche) est $!p \xrightarrow{a} !p \parallel p'$, et provient d'une transition $p \xrightarrow{a} p'$, qui nous permet de

trouver q' tel que $q \xrightarrow{a} q'$ et $p' \sim q'$. Puisque la bisimilarité est préservée par la mise en parallèle, on a $!p \parallel p' \sim !p \parallel q'$, et il suffit ensuite de « simplifier par q' » pour retomber sur l'unique couple de R_1, ce que l'on fait grâce à la fonction $\mathcal{C}_\parallel$:

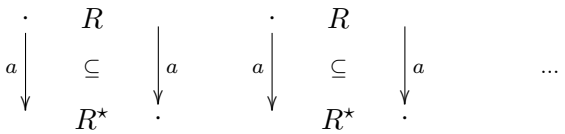


Le cas d'une offre à droite pouvant se résoudre de façon symétrique, R_1 est bien une bisimulation modulo f.

De façon un peu plus générale, ces techniques modulo contexte permettent de se focaliser sur les processus que l'on souhaite considérer, sans avoir à les placer dans leur contexte d'utilisation (les techniques modulo contexte ne sont correctes que dans le cas où la bisimilarité est une congruence, c'est-à-dire justement lorsque deux processus bisimilaires restent bisimilaires lorsqu'ils sont placés dans un contexte arbitraire). On réduit ainsi la complexité des processus à manipuler dans les candidats de bisimulation. Notons que la méthode des *paires critiques* en réécriture [TeR03] correspond intuitivement à ce genre de techniques : elle permet de se concentrer sur la partie « intéressante » de la relation.

Modulo transitivité

Dans la même veine que la technique modulo bisimilarité, on trouve la technique modulo *transitivité et réflexivité*, qui s'exprime à l'aide de la fonction de fermeture réflexive transitive ($\mathrm{id}^\star : R \mapsto R^\star$). Comme pour modulo bisimilarité, la preuve de correction est diagrammatique : on assemble les diagrammes suivants à l'aide d'une récurrence, pour montrer que $R^\star$ est une bisimulation dès que R est une bisimulation modulo transitivité :



Bien qu'il soit difficile de le démontrer sur de petits exemples, cette technique est vraiment intéressante car elle permet de « localiser » le candidat de bisimulation. Nous essayons d'expliquer cette intuition à l'aide d'un exemple très informel, provenant de l'exemple concret (et formel) que nous étudierons au chapitre 6. Imaginons pour cela un système où les états sont représentés par des arbres portant des entiers sur leurs nœuds, et dans lequel on souhaite montrer que deux arbres portant les mêmes ensembles d'entiers sont bisimilaires (nous ne spécifions ni les interactions ni les transitions ; supposons

qu'elles sont suffisamment complexes pour que le problème soit difficile...).

Une première solution consiste à considérer la relation E ci-dessous :

$$E \triangleq \{\langle A_1, A_2\rangle \mid A_1, A_2 \text{ sont deux arbres portant les mêmes entiers}\} \ .$$

Mais comme la structure arborescente de A_1 peut être très différente de celle de A_2, il peut être très difficile de mettre en correspondance leurs différentes transitions, afin de vérifier que E est une bien bisimulation (en particulier si les transitions dépendent fortement de la forme de l'arbre). Une autre solution consiste à essayer de raisonner par transitivité, en considérant la relation suivante :

$$E_1 \triangleq \{\langle A_1, A_2\rangle \mid \ A_2 \text{ s'obtient à partir de } A_1 \text{ en descendant ou en remontant d'un cran un nœud quelconque}\} \ .$$

Dans l'exemple ci-dessus, l'arbre de droite peut s'obtenir à partir de celui de gauche en remontant d'un cran les trois nœuds $2, 3, 4$, puis en descendant le nœud 2 sous le nœud 4. Plus généralement, en partant de la relation E_1, on atteint par transitivité la relation E toute entière : on a $E = E_1^\star$.

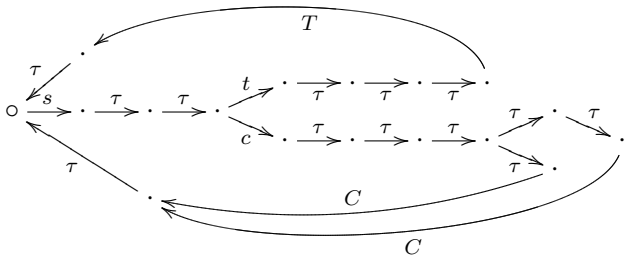
L'avantage majeur de la relation E_1 est sa « localité » : elle met en relation des arbres qui ne diffèrent que très peu, de telle sorte qu'il est a priori bien plus facile de faire correspondre les transitions partant de chacun des arbres mis en jeu. Ensuite, si les règles de transitions sont telles que le léger décalage introduit entre les deux arbres initiaux se préserve au fil des transitions, on peut espérer montrer sans technique modulo que E_1 est une bisimulation (puis que $E \subseteq \sim$, la bisimilarité étant transitive). Si au contraire ce léger décalage « dégénère » – c'est-à-dire qu'après une ou plusieurs transitions, il faut descendre ou remonter plusieurs nœuds pour pouvoir égaliser les arbres obtenus – alors on aura besoin de la technique modulo transitivité pour pouvoir répondre dans le jeu de bisimulation : cette technique permet de ne considérer que les offres de la relation E_1, tout en ayant la possibilité de répondre en utilisant la relation E.

Alors que les techniques modulo contexte nous permettaient de localiser le candidat en simplifiant les processus (ou états) mis en jeu, la technique modulo transitivité nous permet de passer d'un candidat défini de façon globale, à un candidat défini par une transformation locale des états : il s'agit d'une seconde forme de localisation.

1.4 Le cas « faible »

Le fait de représenter un système par le graphe de ses transitions suppose implicitement que ces transitions correspondent à des actions « atomiques » : lorsque qu'un processus effectue une transition, on ne peut pas observer les états intermédiaires par lesquels passe ce processus avant de l'avoir terminée. Si ce niveau d'abstraction est satisfaisant lorsque l'on considère une spécification (celle du distributeur de boissons par exemple, où l'on ne veut pas observer les étapes intermédiaires correspondant à l'infusion progressive du thé), il ne l'est plus lorsque l'on considère une implantation (un distributeur de boissons « réel » passe par l'étape où le thé n'est pas encore infusé).

Pour rendre compte de ces états intermédiaires, une idée naturelle consiste à distinguer l'une des étiquettes, dite *silencieuse* (ou *interne*) et notée τ, et à l'utiliser pour étiqueter certaines transitions « non-observables », ne correspondant pas à des interactions avec l'extérieur. Un distributeur de boissons pourra alors être décrit par le graphe étiqueté suivant, où deux transitions internes sont nécessaires pour compter la monnaie, trois pour préparer le thé, quatre ou cinq pour du café, et une pour nettoyer avant d'accepter à nouveau de la monnaie :

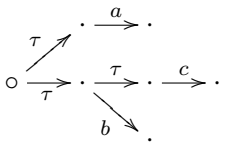


(1.11)

La notion de bisimilarité précédemment définie (qui correspond dans la littérature à la notion de bisimilarité *forte*) est dans ce cas relativement contraignante : comme elle considère l'étiquette τ comme les autres étiquettes, elle permet de compter les transitions silencieuses, et distingue ainsi la machine précédente de la spécification initiale (1.4). La bisimilarité forte correspond ainsi à une observation chronométrée des processus. Si l'on souhaite ignorer les questions d'efficacité, il nous faut passer à une notion d'équivalence plus faible, capable d'ignorer les transitions silencieuses. En termes de traces, on peut par exemple égaliser deux processus dès qu'ils ont les mêmes ensembles de traces « nettoyées », où le nettoyage d'une trace consiste à en retirer toutes les occurrences de l'étiquettes silencieuse, τ. Mais cette équivalence souffre des même faiblesses que dans le cas fort : en présence de non-déterminisme, le comportement interactif du système n'est pas bien pris en compte.

Il est difficile de définir une opération similaire de nettoyage directement sur les graphes, pour se ramener au cas de la bisimilarité forte : si cela semble naturel sur l'exemple précédent, où le déterminisme des transitions silencieuses les rend relativement inintéressantes, comment nettoyer le graphe

suivant, tout en conservant sa signification ?



Cet exemple montre notamment que les transitions silencieuses ne sont pas toujours anodines, et peuvent faire changer de classe d'équivalence : chacune des transitions silencieuses de l'exemple précédent change l'ensemble des interactions éventuellement accessibles, et correspond ainsi à un *choix* irrémédiable vis-à-vis du comportement extérieur du système.

Bisimilarité faible

L'équivalence la plus répandue dans la littérature est la *bisimilarité faible*, dont l'idée consiste à considérer des relations de transition dérivées, qui permettent d'ignorer plus ou moins les transitions silencieuses. Notons $\stackrel{\widehat{\tau}}{\Rightarrow}$ la fermeture réflexive transitive de la relation $\stackrel{\tau}{\longrightarrow}$, et pour chacune des interactions a restantes, définissons la relation $\stackrel{\widehat{a}}{\Rightarrow}$ comme l'ensemble des séquences de transitions comportant un nombre arbitraire de transitions silencieuses, et une unique transition étiquetée a :

$$\stackrel{\widehat{\tau}}{\Rightarrow} \triangleq \stackrel{\tau}{\longrightarrow}^{\star} \qquad\qquad \stackrel{\widehat{a}}{\Rightarrow} \triangleq \stackrel{\tau}{\longrightarrow}^{\star} \cdot \stackrel{a}{\longrightarrow} \cdot \stackrel{\tau}{\longrightarrow}^{\star} \ . \tag{1.12}$$

L'ensemble de ces transitions étiquetés forme un nouveau LTS, dit *faible*, qui nous permet de définir la *bisimilarité faible*, notée $\approx$, comme étant la bisimilarité forte dans le LTS faible. Les jeux de bisimulation faible correspondent ainsi aux diagrammes suivants, que satisfait la bisimilarité faible ; la lettre grecque (α) y dénote une étiquette quelconque, quantifiée universellement : nous réserverons dans la suite les lettres romaines ($a, b, \dots$) pour dénoter les étiquettes *visibles*, celles qui sont distinctes de τ.

$$\begin{array}{ccc} \cdot & R & \\ {\scriptstyle\widehat{\alpha}}\Downarrow & \subseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ \cdot & R & \cdot \end{array} \qquad\qquad \begin{array}{ccc} & R & \cdot \\ {\scriptstyle\widehat{\alpha}}\Downarrow & \supseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ \cdot & R & \end{array} \tag{1.13}$$

Le distributeur de boissons précédent (1.11) est par ainsi faiblement bisimilaire à sa spécification (1.4) : le jeu de bisimulation faible permet d'ignorer les transitions silencieuses. Dans le calcul présenté précédemment, les seules transitions silencieuse proviennent des processus de la forme $\tau.p$, et l'on peut facilement montrer que l'on a $\tau.p \approx p$, pour tout processus p.

Le choix de cette équivalence faible est discutable : van Glabbeek en étudie cent cinquante cinq dans son second « spectre » [vG93]... L'objectif

de cette thèse n'étant pas de justifier cette équivalence, contentons-nous du fait qu'elle se situe parmi les équivalences les plus fines (entre celles qui ne comptent pas les transitions silencieuses – la plus fine étant la *bisimulation arborescente* [vG93]), et que ce soit la plus utilisée en pratique.

Techniques de preuve

Si nous avons présenté et justifié les techniques modulo dans le cas de la bisimilarité forte par soucis de clarté, c'est pour la bisimilarité faible qu'il est le plus important d'avoir des techniques de preuve performantes. C'est effectivement lors de l'étude de systèmes suffisamment élaborés que ces techniques sont requises (implantations distribuées, validation d'optimisations, de protocoles cryptographiques, etc...), où la bisimilarité forte est bien trop discriminante pour pouvoir être utile. Dans le cas d'une optimisation, on s'attend typiquement à ce que le système optimisé effectue moins de transitions internes que le système initial et ne soit ainsi pas fortement bisimilaire à ce dernier.

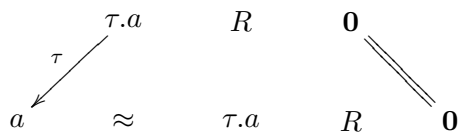
La technique de preuve associée à la définition précédente de bisimilarité faible n'est pas satisfaisante : les jeux de bisimulation définis par les deux diagrammes précédents sont redondants, et nous forcent à considérer plus d'offres qu'il n'est réellement nécessaire. En effet, les deux diagrammes suivants mènent à la même notion de bisimulation, mais sont bien plus faciles à vérifier : les offres y sont constituées de transitions élémentaires, et non pas de séquences arbitraires.

$$\begin{array}{ccc} \cdot & R & \\ {\scriptstyle\alpha}\downarrow & \subseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ \cdot & R & \cdot \end{array} \qquad\qquad \begin{array}{ccc} & R & \cdot \\ {\scriptstyle\widehat{\alpha}}\Downarrow & \supseteq & \downarrow{\scriptstyle\alpha} \\ \cdot & R & \end{array} \tag{1.14}$$

Remarquons qu'il s'agit en quelque sorte d'une technique modulo transitivité « verticale » : tout comme un assemblage horizontal validait la technique modulo transitivité dans le cas de la bisimulation forte, c'est ici un assemblage vertical des diagrammes précédents (1.14) qui aboutit aux diagrammes initiaux (1.13).

Si l'on garde la première définition, les techniques modulo que l'on a vues pour la bisimilarité forte restent trivialement valides dans le cas faible : on a juste changé de LTS. Malheureusement, le fait de simplifier les jeux de bisimulation faible (i.e., d'utiliser les diagrammes (1.14)) nous fait perdre la correction de certaines de ces techniques modulo : la technique modulo contexte reste correcte, mais pas les techniques fondées sur des arguments diagrammatiques, telle que modulo bisimilarité ou modulo transitivité. Le contre-exemple standard est le suivant, où $R = \{\langle \tau.a, \mathbf{0}\rangle\}$ est une bisimulation faible modulo bisimilarité faible ($\approx$), sans être contenue dans la bisimilarité faible ($\tau.a$ peut effectuer une transition visible selon a, mais pas

0) :



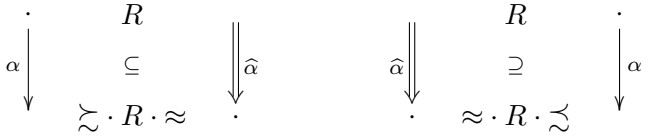
(1.15)

Intuitivement, le fait que la bisimilarité faible ne compte pas les transitions silencieuses lui permet de « remonter le temps », et d'annuler ainsi certaines offres silencieuses dans les jeux de bisimulation faible modulo $\approx$. Sur le contre-exemple, c'est ce qui permet de ne jamais avoir à considérer l'offre visible selon a. Naturellement, si l'on considère les premiers jeux de bisimulation faible (1.13), ce problème n'apparaît pas : comme les offres sont des transitions faibles, il nous faut considérer l'offre $\tau.a \stackrel{\widehat{a}}{\Rightarrow} \mathbf{0}$, que la bisimilarité faible ne permet pas d'annuler, et à laquelle le processus **0** ne pourra pas répondre.

Pour contourner ce problème, il faut donc renforcer la relation autorisée pour réécrire les processus résultant des offres silencieuses (sur le contre-exemple, le processus a – on appellera dans la suite *position de transitivité* la place occupée par ces processus), de façon à empêcher cette relation d'annuler les transitions silencieuses. La solution standard [SM92] consiste à utiliser au lieu de la bisimilarité faible un préordre plus fort, l'*expansion* $(\succsim)$ [2]. Ce préordre comportemental est défini en renforçant la partie de gauche à droite des jeux de bisimulation, de telle sorte que le processus de droite doive toujours être au moins aussi rapide que celui de gauche :

(1.16)

(la seule différence entre une transition de la forme $\stackrel{\widehat{\alpha}}{\longrightarrow}$ et une transition de la forme $\stackrel{\alpha}{\longrightarrow}$ est que la première autorise le processus à ne pas bouger lorsque α est l'action silencieuse : $\stackrel{\widehat{\tau}}{\longrightarrow} \triangleq \stackrel{\tau}{\longrightarrow}^{=}$; $\stackrel{\widehat{a}}{\longrightarrow} \triangleq \stackrel{a}{\longrightarrow}$). Ce préordre donne lieu à la technique (correcte) nommée « bisimulation faible modulo expansion », qui a largement été utilisée en pratique, et que l'on peut représenter intuitivement par les deux diagrammes suivants :



(1.17)

2. Ce préordre a été découvert indépendemment par S. Arun-Kumar et Hennessy [AKH92], qui cherchaient à caractériser une notion d'efficacité sur les processus ; et par Milner et Sangiorgi, qui essayaient de rattraper le problème de la bisimulation faible « modulo faible » [SM92].

Les techniques modulo apparaissent dès le second livre de Milner [Mil89], dont la première édition contient notamment la technique incorrecte dans le cas faible. Cette erreur est corrigée dans la seconde édition (1991), après que Gunnar Sjödin et Bengt Jonsson l'ont découverte. Sangiorgi tombe indépendamment sur cette erreur et publie avec Milner l'article où ils proposent d'utiliser l'expansion [SM92] (pour la seconde édition du livre de Milner, la solution consistant à utiliser faible modulo faible, mais en considérant toutes les offres faibles, s'avère suffisante). Sangiorgi définit ensuite une théorie des technique modulo pour la bisimulation forte [San98], dans laquelle il définit, entre autres, les techniques modulo contexte ; cette théorie est reprise dans son livre sur le π-calcul [SW01]. Ces diverses techniques ont été largement utilisées, et se sont souvent avérées essentielles (notamment pour les encodages de stratégies du λ-calcul dans le π-calcul [SW01]).

1.5 Contributions

L'expansion n'est pas toujours suffisante

La nécessité de trouver des techniques modulo plus puissantes que modulo expansion est apparue lors de mon travail de DEA, où nous cherchions avec Daniel Hirschkoff et Davide Sangiorgi, à optimiser une machine abstraite (la Pan [SV01]), permettant d'exécuter de manière distribuée le calcul des « Safe Ambients » [LS00]. Cette machine utilise des *messagers* : des agents chargés de rediriger indéfiniment des messages entre deux localités ; l'optimisation a consisté à définir un algorithme distribué permettant de retirer ces messagers après qu'ils sont devenus inaccessibles ; on a ainsi obtenu la GcPan [HPS05].

Vis-à-vis du comportement extérieur de la machine, ces messagers ont un rôle très transparent : il peuvent intuitivement être remplacés par l'application d'une substitution globale, qui remplace toute occurrence de l'adresse retransmise par l'adresse de destination. Si cette substitution est irréaliste du point de vue d'une implantation distribuée (il faudrait parcourir tout internet), elle permet de raisonner bien plus facilement. Ainsi, dans la preuve de correction initiale [SV01], la technique modulo expansion est utilisée afin de retirer tous les messagers qui apparaissent lors des jeux de bisimulation, en appliquant la substitution qu'ils représentent. Cela permet de considérer un candidat de bisimulation sans messagers, bien plus simple à étudier puisque tout message est nécessairement arrivé à destination.

Mais certains mécanismes de blocage introduits pour notre optimisation font tomber le résultat d'expansion : dans la machine optimisée, un messager est seulement faiblement bisimilaire à la substitution qu'il représente. Ce résultat négatif provient de la nature trop « contraignante » du jeu d'expansion : le processus de droite doit être aussi rapide que celui de gauche *à chaque étape*. L'expansion ne permet donc pas de caractériser des optimi-

sations dont le gain d'efficacité ne se mesure que de façon amortie. Ainsi, l'approche suivie dans la preuve initiale n'étant plus possible, nous avons dû donner une preuve de correction directe et laborieuse, s'appuyant sur un candidat défini de façon très globale. C'est cet « échec » qui nous a conduit à chercher de nouvelles techniques modulo pour la bisimulation faible.

Au dernier chapitre de cette thèse, nous étudions en détail une version simplifiée de cette optimisation, qui nous permet de montrer pourquoi l'expansion ne suffit pas, et comment les nouvelles techniques permettent de simplifier la preuve de correction.

Techniques modulo et hypothèses de terminaison

La correction de la technique modulo transitivité entraîne celle de modulo bisimilarité ; par contraposée, on peut donc déduire du contre-exemple de faible modulo faible (1.15) la non-correction de la technique faible modulo transitivité. En effet, la relation $\{\langle \tau.a,\ \mathbf{0}\rangle, \langle a,\ \tau.a\rangle, \langle a,\ a\rangle, \langle \mathbf{0},\ \mathbf{0}\rangle\}$ est une bisimulation modulo transitivité, bien qu'elle contienne le couple $\langle \tau.a,\ \mathbf{0}\rangle$. On retrouve exactement le phénomène qui intervenait avec la technique modulo $\approx$: la transitivité permet aussi d'annuler certaines offres silencieuses, et nous empêche ainsi de fermer certains diagrammes. Par exemple, lorsque l'on essaie de fermer le diagramme suivant, l'hypothèse sur R nous permet de fermer deux sous-diagrammes, mais nous ramène inéluctablement au diagramme initial :

$$\begin{array}{ccccccccc} a & R & \tau.a & & & R & & \mathbf{0} & \\ \Big\downarrow{\scriptstyle a} & & \Big\downarrow{\scriptstyle \tau} & & & & & \| & \\ & & a & R & \tau.a & R & & \mathbf{0} & \\ & & \Big\downarrow{\scriptstyle a} & & & & & & \\ \mathbf{0} & R & \mathbf{0} & & & & & ? & \end{array} \tag{1.18}$$

On montrera au chapitre 4 que la non-correction de cette technique correspond très précisément à un problème bien connu dans le domaine de la réécriture [TeR03] [3] : « la confluence locale n'entraîne pas la confluence ». Une réponse classique à ce problème est le lemme de Newman [New42], qui exploite une hypothèse supplémentaire de *terminaison*.

Nous avons donc cherché à utiliser ce genre d'outils, du domaine de la réécriture, pour définir des techniques permettant de contourner le problème de modulo transitivité dans le cas faible. Une adaptation naïve du lemme de Newman ne donne cependant pas lieu à des techniques modulo réellement exploitables : l'hypothèse de terminaison est trop forte pour être satisfaite dans les situations courantes. En cherchant à affaiblir cette hypothèse, nous avons finalement obtenu une hypothèse bien plus précise, qui exprime par une notion de terminaison l'interdiction de remonter ou d'annuler les offres silencieuses (théorèmes 4.14 puis 4.30). A notre grande surprise, cette technique,

3. Le contre-exemple (1.18) correspond exactement à [BKvO98, figure 10].

une fois ramenée dans le domaine de la réécriture, fournit une généralisation du lemme de Newman (corollaire 4.15).

Cependant, la preuve de cette nouvelle technique est assez complexe : plusieurs inductions imbriquées sont nécessaires, et le résultat en lui-même souffre de problèmes de modularité ; il est incompatible avec certaines techniques standard (notamment modulo expansion), qui ont facilement tendance à « détruire » l'argument de terminaison. De plus, pour les techniques auxquelles il peut être combiné, la preuve initiale doit systématiquement être refaite pour valider la correction de la combinaison. Si cela ne pose pas vraiment de problème pour des techniques simples, telle que modulo réflexivité, lorsque la preuve de correction de la technique à rajouter est elle aussi difficile (modulo contexte), la preuve de correction globale devient ingérable.

Théorie générale des techniques modulo pour la co-induction

Sangiorgi définit dans [San98] une théorie des techniques modulo pour la bisimulation forte, où une sous-famille de l'ensemble des techniques modulo correctes est isolée (les fonctions « sûres »), qui bénéficie de bonnes propriétés en termes de modularité : les techniques modulo de cette famille peuvent être combinées librement, ce qui permet d'obtenir la correction de techniques sophistiquées à partir de la correction de techniques plus simples. Ainsi, modulo contexte peut être combinée à modulo transitivité (dans le cas fort), sans avoir à faire une preuve de correction mêlant contextes et transitivité.

Cette théorie s'étend naturellement à la bisimilarité faible [SW01], où la technique modulo expansion prend la place de modulo bisimilarité, cette dernière n'étant plus correcte. L'ensemble des techniques modulo « standard » rentre dans le cadre des fonctions sûres, dans le cas fort comme dans le cas faible, mais ce n'est pas le cas de notre nouvelle technique, qui ne s'exprime pas par le biais d'une fonction sûre, et ne peut donc pas être combinée automatiquement aux techniques standard.

Dans [Pou05b], où nous décrivons une première ébauche de cette technique, nous donnons des conditions qui permettent de réaliser ce genre de combinaison. L'ensemble obtenu étant très ad hoc, nous avons ensuite cherché de manière plus générale comment combiner une fonction correcte et une fonction sûre quelconques. Ce faisant, il est apparu que la théorie des techniques modulo pour la bisimilarité gagnait à être définie à un niveau d'abstraction bien plus élevé, où l'on ne parle même plus de processus, ni même de transitions : on peut se concentrer dans un premier temps sur l'aspect *co-inductif* de la bisimilarité, vue comme le plus grand point fixe d'une fonction croissante (la *génératrice*) sur un *treillis complet*.

On obtient ainsi une théorie des techniques modulo pour la co-induction en général, et non plus seulement pour la bisimilarité, et cela nous permet de mieux distinguer ce qui fait partie de la théorie de la co-induction elle même, de ce qui est plus spécifique aux relations binaires, ou à la bisimilarité.

Le résultat principal de cette théorie est le théorème 2.52, qui donne une condition suffisante pour que la combinaison d'une technique *compatible* (notre équivalent des fonctions sûres) et d'une technique correcte arbitraire reste correcte. Un autre point relativement important, mais plus technique, est la capacité de ramener l'étude des techniques pour les jeux à deux cotés (bisimulation, expansion) à celle des techniques pour des différents jeux de simulation. C'est important, puisque cela permet de ne pas avoir à dupliquer chacune des preuves dans la suite. Bien qu'il soit immédiat de déduire les techniques correspondant à un jeu de droite à gauche à partir des techniques du même jeu, de gauche à droite, il est moins facile d'en déduire les techniques correspondant au jeu à deux côtés (bisimulation). Nous nous attachons donc à définir au niveau abstrait des outils qui permettront ensuite de se concentrer sur la simulation.

Abstraction : vers des preuves calculatoires

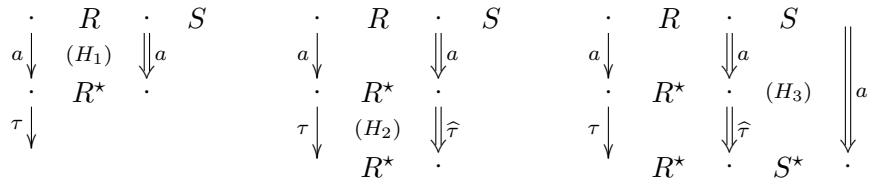
A ce stade, on peut naturellement instancier le treillis complet par celui des relations binaires sur un ensemble de processus quelconque, définir les génératrices concrètes des différents jeux de simulation, en chercher les techniques compatibles, puis les techniques seulement correctes (celles fondées sur des hypothèses de terminaison), et enfin, composer et « symétriser » le tout, pour passer de la simulation à la bisimulation.

C'est effectivement la stratégie que l'on adoptera, mais on prendra soin de ne pas instancier le treillis complet : en effet, lorsque l'on considère les diagrammes (1.8), qui définissent le jeu de bisimulation, on s'aperçoit que l'on peut considérer le candidat de bisimulation (R) et les relations de transition ($\xrightarrow{a}$) comme des objets abstraits, que l'on peut *comparer* ($\subseteq$), *composer* ($\cdot$), et dont on peut prendre les *converses*. Les processus mis en relation ne sont notamment pas mentionnés, de telle sorte que l'on peut se placer dans une *algèbre relationnelle* (un treillis complet muni des deux opérations précédentes, et satisfaisant certains axiomes). L'idée d'une telle structure algébrique, permettant de représenter les relations binaires sans mentionner les « variables d'individu », est due à Alfred Tarski [Tar41, TG87].

On gagne ainsi en généralité : la théorie obtenue s'applique dans des situations où les bisimulations ne sont pas de simples relations binaires (c'est le cas par exemple des « bisimulations typées » [SW01, HR04], où les processus sont mis en relation à un certain type ou dans un certain environnement de typage ; ou encore, des récentes « bisimulations à environnement » [SKS07a], où un environnement est utilisé pour garder une trace de la connaissance de l'observateur lorsque les processus sont comparés).

Ce niveau d'abstraction est de plus très bien adapté aux preuves diagrammatiques. Par exemple, une preuve qui consiste à appliquer trois hypothèses pour fermer un diagramme selon les trois étapes ci-dessous se traduit for-

mellement par le « calcul » suivant :



$$
\begin{aligned}
& \xleftarrow{\tau} \cdot \xleftarrow{a} \cdot R \cdot S \\
\subseteq\ & \xleftarrow{\tau} \cdot R^\star \cdot \xLeftarrow{a} \cdot S && (H_1) \\
\subseteq\ & R^\star \cdot \xLeftarrow{\widehat{\tau}} \cdot \xLeftarrow{a} \cdot S && (H_2) \\
\subseteq\ & R^\star \cdot S^\star \cdot \xLeftarrow{a} && (H_3)
\end{aligned}
$$

Le raisonnement tenu est ainsi décrit de manière bien plus synthétique que dans une preuve usuelle, où l'on passe son temps à manipuler la quantification existentielle qui se cache dans la définition de la composition relationnelle.

Nous utiliserons de plus les résultats de Henk Doornbos, Roland Backhouse et Jaap van der Woude [DBvdW97] pour caractériser la notion de terminaison dans ce cadre abstrait. Cela nous permettra de conserver l'approche calculatoire précédente pour fermer des diagrammes, même lorsque la preuve initiale repose sur des inductions bien fondées sur l'ensemble des processus : ces inductions bien fondées pourront être menées de façon calculatoire, en restant à ce niveau d'abstraction.

Retour sur les techniques modulo contexte

Le fait de définir une théorie générale des techniques modulo pour la co-induction nous a permis de réaliser que les fonctions compatibles elles-mêmes peuvent être vues comme des objets co-inductifs, pour lesquels des techniques modulo existent. En utilisant ces techniques « de second ordre », nous proposons une méthode pour prouver la validité des techniques modulo contexte, bien plus simple et modulaire que les méthodes existantes. Nous illustrons cette méthode en l'appliquant afin de retrouver l'ensemble des techniques modulo contexte disponibles dans le cas de CCS.

La validité de ce genre de techniques est généralement difficile à établir, notamment lorsque l'on considère des calculs d'ordre supérieur (λ-calcul [Las98], HOπ [SW01]) ; c'est en fait un problème plus difficile encore que celui de la congruence d'une bisimilarité donnée. Avoir une méthode efficace pour prouver de tels résultats est donc crucial.

1.6 Plan

Nous définissons la théorie générale des techniques modulo pour la coinduction au chapitre 2, qui débute par une étude rapide des structures ou notions généralement standard que nous utiliserons dans la suite (préordre, treillis complet, fonctions, etc... – section 2.1). La théorie est tout d'abord présentée en utilisant les génératrices (section 2.2), qui ont l'avantage de permettre une présentation très uniforme des résultats. Nous donnons ensuite une présentation alternative, fondée sur la notion de *progression*, qui nous permet de mieux se situer par rapport à la théorie de Sangiorgi [San98], et dont la formulation nous paraît plus adaptée à la pratique (section 2.3).

Nous appliquons cette théorie au cas particulier de la bisimilarité au chapitre 3 : après avoir défini la notion d'algèbre relationnelle (section 3.1), nous définissons les génératrices des différents jeux de simulation et nous étudions les techniques modulo standard pour ces jeux (section 3.2). Nous étendons alors le tout aux différentes notions de bisimilarité en utilisant les résultats du chapitre 2 (section 3.3). Ce faisant, nous préparons le terrain pour le chapitre suivant, en étudiant la notion de *préordre contrôlé*, dont le but est d'être utilisé à la place de l'expansion.

Le chapitre 4 est dédié aux nouvelles techniques, qui s'appuient sur des hypothèses de terminaison : nous mettons dans un premier temps en évidence les liens entre les problèmes auxquels nous nous trouvons confrontés et ceux de la théorie de la réécriture, et nous donnons notre généralisation du lemme de Newman (section 4.1). Nous montrons ensuite comment caractériser la notion de terminaison dans les algèbres relationnelles [DBvdW97], puis nous généralisons à ce cadre abstrait le résultat de commutation précédent (section 4.2). Nous en déduisons finalement la technique modulo pour la bisimulation faible, et nous montrons comment utiliser la théorie des deux chapitres précédents pour combiner ce résultat aux techniques standard (section 4.3). Une autre technique, moins utile en pratique, est étudiée à la section 4.4, et nous donnons finalement à la section 4.5 quelques conséquences importantes des résultats précédents dans le cas particulier des systèmes dits *réactifs*.

Nous étudions les techniques modulo contexte au chapitre 5 : nous montrons comment obtenir des techniques modulo « de second ordre » pour montrer la compatibilité de techniques modulo « de premier ordre » (section 5.1). Nous appliquons ensuite ces techniques afin de définir la méthode « des contextes initiaux » (section 5.2), que nous illustrons en retrouvant les techniques modulo contexte disponibles pour les différentes équivalences dans le cas particulier de CCS (section 5.3).

Le chapitre 6 contient un exemple d'application de nos techniques modulo, issu de notre travail de DEA sur la Pan : nous définissons un modèle d'exécution pour des programmes distribués, capables de *migrer* entre

différentes localités (section 6.1), puis nous montrons comment utiliser les techniques présentées dans cette thèse pour valider deux implantations de ce modèle d'exécution : l'une très simple, et l'autre, optimisée (sections 6.2 et 6.3).

Nous concluons par quelques pistes de recherche au chapitre 7, où nous indiquons par ailleurs les travaux que nous avons réalisés pendant cette thèse qui n'apparaissent pas dans ce document.

Publication des résultats

Une première version des techniques modulo reposant sur des hypothèses de terminaison a été présentée à ICALP'05 [Pou05b]. Les résultats concernant l'*élaboration* (section 4.5.1) ont été présentés à CONCUR'06 [Pou06b]. La théorie générale des techniques modulo pour la co-induction, et son application aux techniques modulo contexte a été présentée à APLAS'07 [Pou07a]. Le travail sur la GcPan a été présenté à COORD'05 [HPS05] ; le chapitre 6 de cette thèse, qui reprend ces travaux de façon plus générale, et en utilisant les nouvelles techniques modulo, provient de la version longue d'un article présenté à TGC'06 [Pou06a].

La présentation abstraite dans les algèbres relationnelles (notamment les preuves abstraites par induction bien fondée [DBvdW97]), l'extension du lemme de Newman, ainsi que les résultats sur le déterminisme, la τ-inertie et leurs liens avec la 2-simulation et la bisimulation, n'ont pas encore été publiés.

Chapitre 2

Co-Induction dans les treillis complets

2.1 Structures algébriques et propriétés de base

Nous introduisons dans cette section les diverses structures et notions qui seront utilisées dans ce chapitre, puis légèrement spécialisées dans les chapitres suivants. Ces notions étant communément répandues, nous nous restreignons aux résultats qui nous serviront par la suite, et nous ne cherchons pas à en donner les formes les plus générales. Bien que la plupart de ces résultats apparaissent sous des formulations plus ou moins différentes dans la littérature que l'on cite, ou font partie du folklore, nous donnons quelques unes des démonstrations, qui sont généralement élémentaires.

Nous commençons par définir les treillis complets, que nous enrichissons dans un second temps par une structure de monoïde. Nous étudions ensuite les propriétés des fonctions sur ce genre d'espaces.

2.1.1 Treillis complets

La théorie des ensembles ordonnés et des treillis est relativement ancienne, et a été longuement étudiée par les mathématiciens, notamment Garrett Birkhoff [Bir40]. Ces structures se sont révélées d'autant plus importantes ces dernières décennies, avec l'essor qu'a connu l'informatique. Le livre de Brian Davey et Hilary Priestley [DP90], qui est plus accessible que celui de Birkhoff, est considéré comme une très bonne référence dans ce domaine.

Définition 2.1 (Préordre, Ordre partiel)**.** Un *préordre* $\langle X, \sqsubseteq \rangle$ est un ensemble X muni d'une relation binaire ($\sqsubseteq$) réflexive et transitive :

$$\forall x \in X,\ x \sqsubseteq x\ , \tag{2.1}$$

$$\forall x, y, z \in X,\ x \sqsubseteq y,\ y \sqsubseteq z \Rightarrow x \sqsubseteq z\ . \tag{2.2}$$

Un *ordre partiel* est un préordre antisymétrique :

$$\forall x, y \in X,\ x \sqsubseteq y,\ y \sqsubseteq x \Rightarrow x = y\ . \tag{2.3}$$

Étant donnés deux éléments $x, y \in X$, on dira dans la suite que x *minore* y (ou encore, que y *majore* x) lorsque $x \sqsubseteq y$.

Définition 2.2 (Treillis complet)**.** Un *treillis complet* $\langle X, \sqsubseteq, \bigvee \rangle$ est un ordre partiel $\langle X, \sqsubseteq \rangle$, tel que toute partie Y de X admet une *borne supérieure*, notée $\bigvee Y$:

$$\forall y \in Y,\ y \sqsubseteq \bigvee Y\ , \tag{2.4}$$

$$\forall x \in X,\ (\forall y \in Y,\ y \sqsubseteq x) \Rightarrow \bigvee Y \sqsubseteq x\ . \tag{2.5}$$

Les bornes supérieures d'un ordre partiel sont déterminées de manière unique, de telle sorte que dans un treillis complet, $\bigvee$ peut être vue comme une application de $\mathscr{P}(X)$ dans X. Notons aussi que tout treillis complet admet un plus petit et un plus grand élément : $\bot \triangleq \bigvee \emptyset$ et $\top \triangleq \bigvee X$. Les bornes supérieures d'ensembles à deux éléments, $\{x, y\}$, seront notées $x \vee y$.

Exemple 2.3 (Treillis complet des parties)**.** L'ensemble $\mathscr{P}(\mathcal{Q})$ des parties d'un ensemble quelconque $\mathcal{Q}$, muni de la relation d'inclusion ensembliste ($\subseteq$), forme un treillis complet, dont les bornes supérieures sont données par la réunion ensembliste ($\bigcup$).

On fixe dans la suite un treillis complet $\langle X, \sqsubseteq, \bigvee \rangle$. Les propriétés (2.6) et (2.7) établies par le lemme suivant nous permettent de comparer des bornes supérieures, la troisième (2.8) exprime l'associativité des bornes supérieures.

Lemme 2.4. *Soient Y, Z deux parties de X. On a :*

$$(\forall y \in Y, \exists z \in Z,\ y \sqsubseteq z) \Rightarrow \bigvee Y \sqsubseteq \bigvee Z\ , \tag{2.6}$$

$$Y \subseteq Z \Rightarrow \bigvee Y \sqsubseteq \bigvee Z\ , \tag{2.7}$$

$$\bigvee (Y \cup Z) = \bigvee Y \vee \bigvee Z\ . \tag{2.8}$$

Dans un treillis complet, les *bornes inférieures* existent et s'expriment à partir des bornes supérieures :

Définition 2.5 (Bornes inférieures)**.** Soit Y une partie de X. On note Y^l l'ensemble des éléments de X majorés par tous ceux de Y. La *borne inférieure* de Y, notée $\bigwedge Y$, se définit comme la borne supérieure de Y^l :

$$Y^l \triangleq \{x \in X \mid \forall y \in Y,\ x \sqsubseteq y\}\ , \qquad \bigwedge Y \triangleq \bigvee Y^l\ .$$

Le lemme suivant montre que l'on a bien défini des bornes inférieures ; les bornes inférieures d'ensembles à deux éléments, $\{x, y\}$, seront notées $x \wedge y$.

Lemme 2.6. *Pour toute partie Y de X, on a :*

$$\forall y \in Y,\ \bigwedge Y \sqsubseteq y\ , \tag{2.9}$$

$$\forall x \in X,\ (\forall y \in Y,\ x \sqsubseteq y) \Rightarrow x \sqsubseteq \bigwedge Y\ . \tag{2.10}$$

L'ensemble Y^l étant potentiellement infini, même lorsque Y est fini, cette définition n'est possible que dans les treillis complets : dans un simple treillis, où seules les bornes supérieures d'ensembles finis sont exigées, il faut demander indépendamment l'existence des bornes inférieures (finies). Dans le treillis des parties, les bornes inférieures sont données par l'intersection ensembliste ($\bigcap$).

Principe de dualité [DP90]. Tout treillis complet $\langle X, \sqsubseteq, \bigvee\rangle$ peut être « dualisé » en renversant l'ordre partiel ($x \sqsupseteq y$ si et seulement si $y \sqsubseteq x$) et en remplaçant les bornes supérieures par les bornes inférieures : $\langle X, \sqsupseteq, \bigwedge\rangle$ est aussi un treillis complet, dont les bornes inférieures sont les bornes supérieures de X. De manière similaire, tout énoncé Φ peut être dualisé en un énoncé Φ^∂, vrai sur $\langle X, \sqsupseteq, \bigwedge\rangle$ dès que Φ est vrai sur $\langle X, \sqsubseteq, \bigvee\rangle$. On obtient ainsi le « principe de dualité » [DP90, pages 15 et 39] :

> *« Pour tout énoncé Φ, vrai dans tout treillis complet, l'énoncé dual, Φ^∂, est vrai dans tout treillis complet ».*

On peut par exemple appliquer ce principe pour obtenir immédiatement le lemme suivant, qui est exactement le dual du lemme 2.4. Bien que l'on utilise peu ce principe dans la suite, il s'avère parfois utile afin de guider l'intuition.

Lemme 2.7. *Pour toutes parties Y, Z de X, on a :*

$$(\forall y \in Y, \exists z \in Z,\ z \sqsubseteq y) \Rightarrow \bigwedge Z \sqsubseteq \bigwedge Y\ , \tag{2.11}$$

$$Y \subseteq Z \Rightarrow \bigwedge Z \sqsubseteq \bigwedge Y\ , \tag{2.12}$$

$$\bigwedge (Y \cup Z) = \bigwedge Y \wedge \bigwedge Z\ . \tag{2.13}$$

On n'aura pas besoin de la notion de *treillis distributif*, c'est-à-dire tel que les bornes supérieures distribuent sur les bornes inférieures. On a cependant l'inégalité suivante, qui s'apparente à une forme de « semi-distributivité », et qui sera utile dans la section 2.2.3.

Lemme 2.8. *Soit $\mathcal{Y}$ un ensemble de parties de X.*

$$\bigvee \bigcap_{Y \in \mathcal{Y}} Y \sqsubseteq \bigwedge_{Y \in \mathcal{Y}} \bigvee Y\ . \tag{2.14}$$

Démonstration. Posons $Z = \bigcap_{Y \in \mathcal{Y}} Y$, on a :

$$\begin{aligned} \forall Y \in \mathcal{Y},\ & Z \subseteq Y \\ \forall Y \in \mathcal{Y},\ & \bigvee Z \sqsubseteq \bigvee Y && \text{(par (2.7))} \\ & \bigvee Z \sqsubseteq \bigwedge_{Y \in \mathcal{Y}} \bigvee Y \ . && \text{(par (2.10))} \end{aligned}$$

■

2.1.2 Monoïde, treillis complet monoïdal

Une autre structure très naturelle est celle de monoïde :

Définition 2.9 (Monoïde)**.** Un *monoïde* $\langle X, \cdot, 1 \rangle$ est un ensemble (X) muni d'une $(\cdot)$ associative et possédant un *élément neutre* (1) :

$$\forall x, y, z \in X,\ x \cdot (y \cdot z) = (x \cdot y) \cdot z \ , \tag{2.15}$$

$$\forall x \in X,\ 1 \cdot x = x \cdot 1 = x \ . \tag{2.16}$$

La loi de composition interne sera aussi appelée *produit*. L'associativité (2.15) nous permettra d'omettre les parenthèses.

Définition 2.10 (Produit de parties)**.** Soit $\langle X, \cdot, 1 \rangle$ un monoïde. On définit le *produit de deux parties* Y et Z de X comme suit :

$$Y \cdot Z \triangleq \{y \cdot z \mid y \in Y, z \in Z\} \ .$$

Définition 2.11 (Itération)**.** Pour tous $x \in X$ et $n \in \mathbb{N}$, on définit l'*itéré* n *fois de* x par récurrence sur n :

$$x^0 \triangleq 1 \ , \qquad\qquad x^{n+1} \triangleq x \cdot x^n \ .$$

L'associativité nous permet d'obtenir le lemme suivant :

Lemme 2.12. *Pour tous entier* $n \in \mathbb{N}$ *et élément* $x \in X$, $x \cdot x^n = x^n \cdot x$.

Dans la suite, ce sont les interactions entre un monoïde et un treillis complet qui vont nous intéresser ; on introduit donc la notion de *treillis complet monoïdal*, qui combine les deux structures.

Définition 2.13 (Treillis complet monoïdal)**.** Un *treillis complet monoïdal* est un quintuplet $\langle X, \sqsubseteq, \bigvee, \cdot, 1 \rangle$ tel que :

- $\langle X, \cdot, 1 \rangle$ est un monoïde ;
- $\langle X, \sqsubseteq, \bigvee \rangle$ est un treillis complet ;
- Le produit *respecte* l'ordre partiel :

$$\forall x, x', y, y' \in X,\ x \sqsubseteq x',\ y \sqsubseteq y' \Rightarrow x \cdot y \sqsubseteq x' \cdot y' \ . \tag{2.17}$$

Afin d'alléger les notations et lorsque le contexte le permettra, on dénotera un treillis complet monoïdal $\langle X, \sqsubseteq, \bigvee, \cdot, 1\rangle$ par son domaine, X. On fixe dans la suite une telle structure. Le produit ne distribue généralement pas sur les bornes supérieures, mais le respect de l'ordre partiel (2.17) entraîne néanmoins l'inégalité suivante :

Lemme 2.14. *Pour toutes parties $Y, Z \subseteq X$, on a :*

$$\bigvee(Y \cdot Z) \sqsubseteq \bigvee Y \cdot \bigvee Z \ .$$

Nous aurons cependant besoin de l'égalité au chapitre 3 ; on introduit donc la notion suivante :

Définition 2.15 (Treillis complet monoïdal continu)**.** Le treillis complet monoïdal est *continu* lorsque son produit distribue sur les bornes supérieures :

$$\forall Y, Z \subseteq X,\ \bigvee Y \cdot \bigvee Z = \bigvee(Y \cdot Z) \ . \tag{2.18}$$

Dans la suite, nous parlerons simplement de *continuité du produit*, lorsque le treillis complet monoïdal correspondant à ce produit peut être déduit du contexte.

Cette propriété de distributivité peut effectivement être vue comme une forme de continuité : le produit doit respecter les « limites » que forment intuitivement les bornes supérieures. Le terme *continuité* est relativement standard, mais correspond parfois seulement à la propriété (2.18) restreinte au cas où Y et Z sont des parties *dirigées* (définition 2.31 plus bas). Nous préférons garder ce vocabulaire légèrement ambigu afin d'éviter toute confusion avec la notion de distributivité, qui fait généralement référence à une relation entre les bornes supérieures et inférieures lorsque l'on parle de treillis, et que nous n'imposons pas. Notons aussi que le fait d'être continu est indépendant du fait d'être *co-continu*, notion duale que l'on obtient en demandant la distribution du produit sur les bornes inférieures.

En prenant un singleton $\{x\}$ et l'ensemble vide dans (2.18), le plus petit élément ($\bot$) d'un treillis complet monoïdal continu est absorbant pour le produit : $\forall x \in X,\ x \cdot \bot = \bot \cdot x = \bot$.

La combinaison d'un treillis et d'un monoïde permet de définir à un niveau abstrait les notions standard de réflexivité, transitivité, ainsi que les opérations de fermeture qui leurs sont associées :

Définition 2.16 (Réflexivité, transitivité, fermetures)**.** Soit x un élément de X.

- x est *réflexif* si $1 \sqsubseteq x$;
- x est *transitif* si $x \cdot x \sqsubseteq x$;
- pour tout prédicat P sur X, on appelle *P-fermeture de x* le plus petit élément y tel que $x \sqsubseteq y$ et $P(y)$, s'il existe.

Comme l'indique le lemme suivant, il faut cependant que le produit soit continu pour obtenir les fermetures transitives :

Lemme 2.17. *Si le produit* $(\cdot)$ *est continu, alors tout élément* $x \in X$ *admet une* fermeture réflexive*, une* fermeture transitive *et une* fermeture réflexive transitive*, respectivement dénotées par* $x^=$*,* x^+ *et* $x^\star$ *; on a de plus les caractérisations et propriétés suivantes :*

$$x^= = x \vee 1 \qquad x^+ = \bigvee_{n>0} x^n \qquad x^\star = \bigvee_{n\in\mathbb{N}} x^n$$
$$x^+ = x \cdot x^\star = x^\star \cdot x \qquad x^\star = (x^=)^+ \ .$$

Exemple 2.18 (Treillis complet monoïdal des relations binaires)**.** L'ensemble des relations binaires sur un ensemble donné forme un treillis complet monoïdal continu (mais non co-continu), tel que les notions précédentes coïncident avec les notions usuelles. Ce cas particulier fondamental de treillis complet monoïdal sera décrit formellement à la section 3.1.1.

Exemple 2.19 (Treillis complets monoïdaux sur les entiers)**.** L'ensemble des entiers naturels, muni de la relation d'ordre usuelle et complété d'un élément $(\top)$ déclaré supérieur à tous les entiers, forme un treillis complet. Couplé au monoïde formé par l'extension naturelle à $\overline{\mathbb{N}}$ de l'addition sur $\mathbb{N}$, on obtient un treillis complet monoïdal continu : $\langle \overline{\mathbb{N}}, \leq, \sup, +, 0\rangle$. On obtient également un treillis complet monoïdal en utilisant la multiplication : $\langle \overline{\mathbb{N}}, \leq, \sup, \times, 1\rangle$; notons cependant qu'il y a plusieurs façons d'étendre la multiplication à $\overline{\mathbb{N}}$ de façon à obtenir un treillis complet monoïdal (on peut choisir arbitrairement les valeurs de $0 \times \top$ et $\top \times 0$) ; ce n'est que dans le cas commutatif où l'on déclare $0 \times \top = \top \times 0 = 0$ que l'on obtient la continuité.

Etant commutatifs, ces modèles particuliers nous intéresserons peu par la suite (cf. remarque 3.6) ; cependant, par leur simplicité, ils fournissent une certaine intuition, et nous permettront de donner des contre-exemples.

A notre connaissance, la notion de treillis complet monoïdal n'a pas été spécialement étudiée dans la littérature ; par contre, lorsque l'on rajoute l'hypothèse de continuité, on retrouve presque les « monoïdes ordonnés par un treillis » (en anglais, *l*-monoids pour lattice-ordered monoids) [Bir40, chapitre XIV] : la seule différence provient du fait que l'on considère un treillis complet, où les bornes arbitraires existent, au lieu d'un simple treillis.

Une propriété courante dans les monoïdes est celle de « commutation » : deux éléments x, y *commutent* lorsque $x \cdot y = y \cdot x$. L'ajout d'un ordre partiel (le treillis complet), nous permet de considérer une propriété intermédiaire de « semi-commutation » : x *semi-commute sur* y lorsque $x \cdot y \sqsubseteq y \cdot x$. C'est cette dernière propriété qui correspond aux jeux de simulation ; nous parlerons donc dans la suite simplement de commutation, bien qu'il s'agisse à

proprement parler de « semi-commutation » (notons que si tous les éléments semi-commutent, alors ils commutent tous, par antisymétrie).

Le lemme suivant, qui sera utilisé à maintes reprises dans les chapitres 3 et 4, nous permet d'itérer une propriété de commutation, afin d'obtenir la commutation d'une fermeture réflexive transitive sur un autre élément :

Lemme 2.20. *Supposons le produit* $(\cdot)$ *continu ; pour tous* $x, y \in X$*, on a :*

$$x \cdot y \sqsubseteq y \cdot x^\star \quad \Rightarrow \quad x^\star \cdot y \sqsubseteq y \cdot x^\star \tag{2.19}$$

$$x \cdot y \sqsubseteq y^\star \cdot x \quad \Rightarrow \quad x \cdot y^\star \sqsubseteq y^\star \cdot x \tag{2.20}$$

Démonstration. On démontre seulement (2.19), (2.20) étant similaire. Pour la seconde, on démontre par récurrence $\forall n \in \mathbb{N},\ x^n \cdot y \sqsubseteq y \cdot x^\star$:

- le cas $n = 0$ provient de la réflexivité de $x^\star$: $x^0 \cdot y \triangleq 1 \cdot y = y \cdot 1 \sqsubseteq y \cdot x^\star$.
- pour le cas inductif, on a : $x^{n+1} \cdot y \triangleq x \cdot x^n \cdot y \sqsubseteq x \cdot y \cdot x^\star \sqsubseteq y \cdot x^\star \cdot x^\star \sqsubseteq y \cdot x^\star$, par récurrence, hypothèse, puis transitivité de $x^\star$.

(notons dans les deux cas l'usage implicite du respect de l'ordre partiel par le produit (2.17)). On conclut grâce à la continuité du produit : on a $x^\star \cdot y \triangleq \left(\bigvee_{n\in\mathbb{N}} x^n\right) \cdot y = \bigvee_{n\in\mathbb{N}} (x^n \cdot y) \sqsubseteq y \cdot x^\star$. ■

Avant de considérer les fonctions, nous donnons un dernier lemme, qui permet de simplifier certaines fermetures réflexives et transitives (dans la suite, nous donnerons la priorité au produit sur les bornes supérieures et inférieures) :

Lemme 2.21. *Si le produit* $(\cdot)$ *est continu, alors on a :*

$$\forall x, y, z \in X,\ 1 \sqsubseteq y \sqsubseteq z \Rightarrow (x \cdot y \vee z)^\star = (y \cdot x \vee z)^\star = (x \vee z)^\star \ , \tag{2.21}$$

$$\forall x, z \in X,\ 1 \sqsubseteq z, z \cdot z \sqsubseteq z \Rightarrow z \cdot (x \cdot z)^\star = (x \vee z)^\star \ . \tag{2.22}$$

Démonstration.(2.21) On ne démontre que $(y \cdot x \vee z)^\star = (x \vee z)^\star$: l'autre égalité s'obtient de manière similaire. On raisonne par double inégalité : on a tout d'abord $y \cdot x \sqsubseteq (x \vee y)^\star \sqsubseteq (x \vee z)^\star$, ce qui mène à l'inégalité directe : $(y \cdot x \vee z)^\star \sqsubseteq ((x \vee z)^\star \vee z)^\star = ((x \vee z)^\star)^\star = (x \vee z)^\star$. L'autre inégalité est immédiate par la réflexivité de y qui entraîne en particulier $x \sqsubseteq y \cdot x$.

(2.22) On prouve chaque inégalité par une récurrence. ■

2.1.3 Extension aux fonctions

Nous considérons maintenant l'ensemble des fonctions sur un treillis complet monoïdal éventuellement continu. Cet ensemble est lui-même muni d'une structure de treillis complet monoïdal, en étendant point à point les opérations de la structure de base. Cela nous permet dans la suite de raisonner de façon algébrique sur les objets comme sur les fonctions.

Nous nous concentrons sur les fonctions *croissantes* : cette propriété très simple joue un rôle primordial dans la suite, notamment pour le théorème de point fixe de Knaster-Tarski (théorème 2.38 dans la section suivante). Se restreindre à cette classe de fonctions permet de simplifier drastiquement la présentation des résultats et s'avère parfaitement anodin en pratique : bien que très courantes, les fonctions non croissantes sur un treillis nous apparaissent comme des objets relativement farfelus, que l'on n'a pas besoin de considérer (les fonctions non-*monotones* – ou non-*compatibles* – auxquelles nous faisons allusion dans l'introduction, et que nous aurons à manipuler par la suite, seront quand-même croissantes).

Définition 2.22 (Monoïde des fonctions)**.** L'ensemble X^X des fonctions de X dans lui même, muni de la *composition fonctionnelle* ($\circ$) et de l'*identité* (id_X), forme un monoïde ; pour tout élément y de X, on note $\widehat{y}$ la *fonction constante* égale à y :

$$\mathrm{id}_X : \begin{array}{l} X \to X \\ x \mapsto x \end{array} \qquad f \circ g : \begin{array}{l} X \to X \\ x \mapsto f(g(x)) \end{array} \qquad \widehat{y} : \begin{array}{l} X \to X \\ x \mapsto y \end{array}$$

L'*image d'une partie* $Y \subseteq X$ par une fonction $f \in X^X$ est la partie suivante de X :

$$f(Y) \triangleq \{f(y) \mid y \in Y\} \quad .$$

Définition 2.23 (Extension des opérations aux fonctions)**.** On étend point à point la relation $\sqsubseteq$, les bornes supérieures et inférieures, et le produit ($\cdot$) aux fonctions : pour toutes fonctions f, g, et toute famille de fonctions F,

$$f \sqsubseteq g \text{ si } \forall x \in X,\ f(x) \sqsubseteq g(x)$$

$$\bigvee F : \begin{array}{l} X \to X \\ x \mapsto \displaystyle\bigvee_{f \in F} f(x) \end{array} \qquad \bigwedge F : \begin{array}{l} X \to X \\ x \mapsto \displaystyle\bigwedge_{f \in F} f(x) \end{array} \qquad f \mathbin{\hat{\cdot}} g : \begin{array}{l} X \to X \\ x \mapsto f(x) \cdot g(x) \end{array}$$

Comme pour les éléments de X, on dit que f *minore* (resp. *majore*) g si $f \sqsubseteq g$ (resp. $g \sqsubseteq f$).

Bien que nous ne changions pas les notations de l'ordre partiel et des bornes inférieures et supérieures dans l'espace des fonctions, nous notons ($\hat{\cdot}$) l'extension point à point du produit de la structure initiale ($\cdot$) ; cela nous permet d'éviter certaines ambiguïtés.

Définition 2.24 (Fonction croissante)**.** Une fonction f est *croissante* si :

$$\forall x, y \in X,\ x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y) \quad . \tag{2.23}$$

Suivant [DP90], on note $X^{\langle X \rangle}$ l'ensemble des fonctions croissantes sur X.

Dans la suite, nous dirons de manière assez générique qu'un opérateur *préserve* une propriété, lorsque cet opérateur, appliqué à des objets satisfaisant la propriété, retourne un objet qui la satisfait aussi.

La proposition suivante est standard : les fonctions croissantes sur un treillis complet forment un treillis complet. Les deux théorèmes qui suivent le sont moins : ce treillis complet peut s'étendre naturellement de deux façons en un treillis complet monoïdal.

Proposition 2.25. *Soit $\langle X, \sqsubseteq, \bigvee \rangle$ un treillis complet. L'ensemble $X^{\langle X \rangle}$ des fonctions croissantes forme un treillis complet : $\langle X^{\langle X \rangle}, \sqsubseteq, \bigvee \rangle$.*

Démonstration.
- L'extension de l'ordre partiel ($\sqsubseteq$) aux fonctions est clairement réflexive, transitive et antisymétrique.
- L'extension des bornes supérieures de X aux fonctions préserve la croissance : soit F une famille de fonctions croissantes et supposons $x \sqsubseteq y$. Pour tout $f \in F$, $f(x) \sqsubseteq f(y)$, en utilisant (2.6) on obtient :

$$\bigvee F(x) = \bigvee_{f \in F} f(x) \sqsubseteq \bigvee_{f \in F} f(y) = \bigvee F(y) \ ,$$

  $\bigvee F$ est donc croissante.
- Enfin, cette extension définit bien des bornes supérieures sur $X^{\langle X \rangle}$: pour toute famille F de fonctions, on a :

$$\forall f \in F,\ \forall x \in X,\ f(x) \sqsubseteq \bigvee_{f \in F} f(x) = \bigvee F(x) \ ,$$

$$\forall g \in X^X,\ (\forall f \in F,\ f \sqsubseteq g) \Rightarrow \forall x \in X,\ \bigvee F(x) = \bigvee_{f \in F} f(x) \sqsubseteq g(x) \ .$$

■

Par dualité, l'extension aux fonctions des bornes inférieures de X préserve la croissance, et coïncide donc avec les bornes inférieures inhérentes à $X^{\langle X \rangle}$. Les notations $\bigvee F$ et $\bigwedge F$ sont donc bien justifiées.

Théorème 2.26. *Soit $\langle X, \sqsubseteq, \bigvee, \cdot, 1 \rangle$ un treillis complet monoïdal.*

Le treillis complet $X^{\langle X \rangle}$ des fonctions croissantes, muni de l'extension du produit $(\hat{\cdot})$ et de la fonction constante $\widehat{1}$: $\langle X^{\langle X \rangle}, \sqsubseteq, \bigvee, \hat{\cdot}, \widehat{1} \rangle$, forme un treillis complet monoïdal. Si de plus $(\cdot)$ est continu, alors $(\hat{\cdot})$ est continu.

Démonstration.
- La fonction constante $\widehat{1}$ est croissante, et le respect de l'ordre partiel par le produit sur X (2.17) nous assure de la stabilité par produit des fonctions croissantes, $X^{\langle X \rangle}$ est donc un monoïde.
- Il reste à vérifier (2.17) au niveau de $X^{\langle X \rangle}$: supposons $f \sqsubseteq f'$ et $g \sqsubseteq g'$. Pour tout $x \in X$, $f(x) \sqsubseteq f'(x)$ et $g(x) \sqsubseteq g'(x)$, d'où $f(x) \cdot g(x) \sqsubseteq f'(x) \cdot g'(x)$ par (2.17) sur le treillis initial. On a donc bien $f \cdot g \sqsubseteq f' \cdot g'$.

– $\langle X^{\langle X\rangle}, \sqsubseteq, \bigvee, \hat{\cdot}, \widehat{1}\rangle$ est donc un treillis complet monoïdal, montrons qu'il est continu lorsque $(\cdot)$ l'est : soit F et G deux familles de fonctions croissantes, pour tout $x \in X$, on a

$$\begin{aligned}\bigvee F(x) \cdot \bigvee G(x) &= \bigvee_{f\in F} f(x) \cdot \bigvee_{g\in G} g(x) \\ &= \bigvee_{f\in F,\ g\in G} f(x)\cdot g(x) \qquad \text{(continuité de } X\text{)} \\ &= \bigvee_{f\in F,\ g\in G} (f \mathbin{\hat{\cdot}} g)(x) = \left(\bigvee F \mathbin{\hat{\cdot}} G\right)(x)\ ,\end{aligned}$$

d'où $\bigvee F \mathbin{\hat{\cdot}} \bigvee G = \bigvee F \mathbin{\hat{\cdot}} G$. ■

Le théorème précédent établit les propriété de la loi de composition dite « horizontale » ; le théorème suivant s'intéresse au contraire aux propriétés de la loi de composition « verticale » :

Théorème 2.27. *Soit $\langle X, \sqsubseteq, \bigvee\rangle$ un treillis complet. L'ensemble $X^{\langle X\rangle}$ des fonctions croissantes, muni de la composition fonctionnelle $(\circ)$ et de l'identité : $\langle X^{\langle X\rangle}, \sqsubseteq, \bigvee, \circ, \mathrm{id}_X\rangle$, forme un treillis complet monoïdal.*

Démonstration. L'identité et la composée de deux fonctions croissantes sont bien croissantes, il suffit donc de vérifier que la composition fonctionnelle respecte l'ordre partiel (2.17). Soient $f \sqsubseteq f'$ et $g \sqsubseteq g'$ quatre fonctions croissantes, pour tout élément $x \in X$, on a $g(x) \sqsubseteq g'(x)$ par hypothèse, et

$$(f \circ g)(x) \triangleq f(g(x)) \sqsubseteq f(g'(x)) \sqsubseteq f'(g'(x)) \triangleq (f' \circ g')(x),$$

par croissance puis hypothèse sur f. On a donc bien $f \circ g \sqsubseteq f' \circ g'$. ■

La proposition 2.25 et le théorème 2.26 sont en fait valides pour des fonctions quelconques : $\langle X^X, \sqsubseteq, \bigvee, \hat{\cdot}, \widehat{1}\rangle$ forme aussi un treillis complet monoïdal ; la croissance est par contre réellement nécessaire pour le théorème 2.27 : $\langle X^X, \sqsubseteq, \bigvee, \circ, \mathrm{id}_X\rangle$ est seulement un treillis complet couplé à un monoïde. En effet sans la croissance des fonctions, la composition fonctionnelle $(\circ)$ ne respecte pas nécessairement l'ordre partiel (2.17).

La composition fonctionnelle $(\circ)$ n'est généralement pas continue (il faudrait pour cela renforcer notre notion de continuité sur les fonctions). On a néanmoins toujours la « continuité à gauche » de la composition fonctionnelle :

Lemme 2.28. *Pour toutes fonction f et famille G de fonctions, on a :*

$$\bigvee_{g\in G} (g \circ f) = \bigvee G \circ f\ . \tag{2.24}$$

Le treillis complet des fonctions croissantes sur X admet donc deux notions de produit distinctes, qui l'étendent en un treillis complet monoïdal :
- l'extension point à point $(\hat{\cdot})$ du produit de la structure de base (X), qui est continue dès que le produit initial l'est ;
- et la composition fonctionnelle $(\circ)$ qui n'est continue qu'à gauche.

Afin de différencier les notions qui en découlent, on conserve les notations f^n, $f^\star$, $f^=$ et f^+ pour dénoter les itérations et fermetures d'une fonction f au sens du treillis complet monoïdal $\langle X^{\langle X\rangle}, \sqsubseteq, \bigvee, \hat{\cdot}, \widehat{1}\rangle$:

$$f^0 = \widehat{1} \qquad f^{n+1} = f \mathbin{\hat{\cdot}} f^n \qquad f^= = f \vee \widehat{1} \qquad f^\star = \bigvee_{n\in\mathbb{N}} f^n \qquad f^+ = \bigvee_{n>0} f^n \ ,$$

et on pose $f^{(n)} \triangleq f^n$, $f^\omega \triangleq f^\star$ au sens de $\langle X^{\langle X\rangle}, \sqsubseteq, \bigvee, \circ, \mathrm{id}_X\rangle$ pour dénoter les itérations de f et sa *fermeture par itération* (les deux autres fermetures ne seront pas utilisées) :

$$f^{(0)} = \mathrm{id}_X \qquad f^{(n+1)} = f \circ f^{(n)} \qquad f^\omega = \bigvee_{n\in\mathbb{N}} f^{(n)} \ .$$

Le lemme suivant exprime la distributivité à droite de la composition fonctionnelle sur le produit point à point, et relie les fermetures point à point d'une fonction aux fermetures d'un élément :

Lemme 2.29. *Soient f, g, h trois fonctions et x un élément de X, on a :*

$$(f \mathbin{\hat{\cdot}} g) \circ h = (f \circ h) \mathbin{\hat{\cdot}} (g \circ h) \tag{2.25}$$
$$f^=(x) = f(x)^= \tag{2.26}$$
$$f^+(x) = f(x)^+ \tag{2.27}$$
$$f^\star(x) = f(x)^\star \tag{2.28}$$
$$f^\star = \mathrm{id}_X^\star \circ f \ . \tag{2.29}$$

On n'a en général ni la distributivité à gauche de $(\circ)$ sur $(\hat{\cdot})$, ni celle de $(\hat{\cdot})$ sur $(\circ)$, à gauche comme à droite. Les équations (2.26), (2.27) et (2.28) nous permettent d'appeler respectivement *fonction de fermeture réflexive, fonction de fermeture transitive et fonction de fermeture réflexive transitive* les fonctions $\mathrm{id}_X^=$, id_X^+ et $\mathrm{id}_X^\star$.

Remarque 2.30 (Manipulation algébrique des fonctions). Chacun des opérateurs précédemment définis sur les fonctions préserve la croissance. Ces opérateurs nous permettrons de raisonner sur les fonctions de façon « algébrique » plutôt que « relationnelle ». Prenons par exemple la fonction $x \mapsto (y \cdot x)^\star \vee z$, qui est décrite de façon relationnelle. Cette fonction peut aussi s'écrire de manière algébrique : $(\widehat{y} \mathbin{\hat{\cdot}} \mathrm{id}_X)^\star \vee \widehat{z}$, et cette décomposition en plusieurs fonctions élémentaires permet d'en établir certaines propriétés de manière triviale. Les résultats de ce chapitre seront donnés de façon algébrique, afin de pouvoir exploiter cette méthode de raisonnement dans les chapitres suivants.

La notation algébrique étant cependant moins lisible lorsque les fonctions deviennent compliquées, les résultats des chapitres suivants, où l'on s'intéresse justement à des fonctions suffisamment riches, seront souvent énoncés avec la notation relationnelle, bien que leur preuves soient essentiellement algébriques.

On introduit maintenant la notion de *fonction continue*, qui nous permettra principalement de simplifier certains calculs au chapitre 5, et qui admet la définition standard suivante [DP90] :

Définition 2.31 (Partie dirigée, fonction continue)**.** Une partie $Y \subseteq X$ de X est dite *dirigée* si Y est non-vide et si pour tous $y_1, y_2 \in Y$, il existe $y \in Y$ tel que $y_1 \sqsubseteq y$ et $y_2 \sqsubseteq y$.

Une fonction f est *continue* lorsque pour toute partie dirigée $Y \subseteq X$,

$$f\left(\bigvee Y\right) = \bigvee f(Y) \ . \tag{2.30}$$

On note $X^{[X]}$ l'ensemble des fonctions continues.

Notons que notre notion de continuité sur les produits est bien plus forte que celles présentée ici sur les fonctions : le produit doit distribuer sur des bornes supérieures arbitraires, est pas seulement celles qui sont dirigées. Cette restriction aux parties dirigées est nécessaire, afin que les fonctions de fermeture par contextes polyadiques soient continues (cf. section 5.2).

Comme l'exprime le lemme suivant, la continuité entraîne la croissance, qui n'implique qu'une « moitié » de la continuité :

Lemme 2.32. *Toute fonction continue est croissante ; pour toute fonction croissante $f \in X^{\langle X \rangle}$, on a :*

$$\forall Y \subseteq X,\ \bigvee f(Y) \sqsubseteq f\left(\bigvee Y\right) \ , \tag{2.31}$$

$$\forall Y \subseteq X,\ f\left(\bigwedge Y\right) \sqsubseteq \bigwedge f(Y) \ . \tag{2.32}$$

Démonstration. Soit f une fonction continue, et supposons $x \sqsubseteq y$. $\{x, y\}$ est une partie dirigée de X, on a donc :

$$f(x) \sqsubseteq f(x) \vee f(y) = f(x \vee y) = f(y) \ .$$

Soit f une fonction croissante, et Y une partie de X ; on a :

$$\begin{aligned}
\forall y \in Y,\ y \sqsubseteq \bigvee Y && \text{(par (2.4))} \\
\forall y \in Y,\ f(y) \sqsubseteq f\left(\bigvee Y\right) && \text{(croissance de } f\text{)} \\
\bigvee f(Y) = \bigvee_{y \in Y} f(y) \sqsubseteq f\left(\bigvee Y\right) \ . && \text{(par (2.5))}
\end{aligned}$$

(2.32) s'obtient par dualité. ■

Tout comme la croissance, la continuité est une propriété « modulaire », au sens où elle est préservée par la plupart des opérateurs permettant de former des fonctions à partir de fonctions de base (proposition 2.33 ci-dessous). Suivant la remarque 2.30, cela nous permet de décomposer des preuves de continuité en plusieurs étapes élémentaires. C'est cette idée de modularité que l'on recherchera dans la section suivante pour les techniques modulo, avec la famille des fonctions « compatibles » (cf. proposition 2.50).

Proposition 2.33. *1. L'identité et les fonctions constantes sont continues.*

2. Les bornes supérieures et la composition fonctionnelle préservent la continuité.

3. Si le produit $(\cdot)$ est continu, son extension aux fonctions $(\hat{\cdot})$ préserve la continuité.

Démonstration. 1. Immédiat.

2. Pour la composition fonctionnelle, il suffit de vérifier que l'image d'une partie dirigée par une fonction croissante est dirigée.

3. Soient $f, g \in X^{[X]}$ deux fonctions continues, et Y une partie dirigée de X ; on a :

$$\begin{aligned}
(f \mathbin{\hat{\cdot}} g)\left(\bigvee Y\right) &= f\left(\bigvee Y\right) \cdot g\left(\bigvee Y\right) \\
&= \bigvee f(Y) \cdot \bigvee g(Y) && \text{(continuité de } f \text{ et } g\text{)} \\
&= \bigvee_{y_1, y_2 \in Y} f(y_1) \cdot g(y_2) \\
&\sqsubseteq \bigvee_{y \in Y} f(y) \cdot g(y) && \text{(la partie } Y \text{ est dirigée)} \\
&= \bigvee (f \mathbin{\hat{\cdot}} g)(Y) \ .
\end{aligned}$$

L'autre inégalité est toujours satisfaite (2.31). ■

Les deux premiers points de la proposition précédente nous indiquent en fait que $\langle X^{[X]}, \sqsubseteq, \bigvee, \circ, \mathrm{id}_X \rangle$ est un treillis complet monoïdal (non continu) Notons par contre que la borne inférieure de deux fonctions continues n'est pas nécessairement continue, de sorte que les bornes inférieures du treillis complet $\langle X^{[X]}, \sqsubseteq, \bigvee \rangle$ ne sont pas celles de $\langle X^{\langle X \rangle}, \sqsubseteq, \bigvee \rangle$ (un raisonnement dual au second point de la proposition 2.33 mène à la préservation de la *co-continuité* par les bornes inférieures, mais pas de la continuité). Remarquons finalement que le troisième point entraîne que $\langle X^{[X]}, \sqsubseteq, \bigvee, \hat{\cdot}, \widehat{1} \rangle$ est aussi un treillis complet monoïdal.

On conclut cette section par la définition de plusieurs propriétés sur les fonctions, qui seront utiles par la suite :

Définition 2.34 (Autres propriétés des fonctions, fermeture)**.** Une fonction f est dite
- *extensive* lorsque $\mathrm{id}_X \sqsubseteq f$;
- *contractante* lorsque $f \sqsubseteq \mathrm{id}_X$;
- *involutive* lorsque $f \circ f = \mathrm{id}_X$;
- *idempotente* lorsque $f \circ f = f$.

Une *fermeture* est une fonction croissante, extensive et idempotente.

La notion de fermeture est celle que l'on trouve en topologie (ce sont les « closure operators » [Bir40, DP90]). En se plaçant dans le treillis complet monoïdal $\langle X^{\langle X\rangle}, \sqsubseteq, \bigvee, \circ, \mathrm{id}_X\rangle$, on s'aperçoit que la notion d'extensivité correspond à celle de réflexivité et que toute fonction réflexive et transitive est idempotente. Dans la suite, on réservera toutefois les notions de réflexivité et transitivité aux éléments de la structure de base (X).

Le lemme 2.17 indique en particulier que les fonctions de fermeture réflexive, transitive et réflexive transitive sont des fermetures, dès que le produit est continu. La notion de fonction continue nous sera utile pour obtenir d'autres fermetures (section 5.2) ; l'avantage majeur des fermetures est qu'elles sont stables par itération.

Proposition 2.35. *Soit f une fonction croissante.*

1. *Si f est extensive et continue, alors f^ω est une fermeture.*
2. *La fonction f est une fermeture si et seulement si $f^\omega = f$.*

Démonstration. 1. f étant extensive, on a $\forall n \in \mathbb{N},\ f^{(n)} \sqsubseteq f^{(n+1)}$, d'où l'on déduit que pour tout $x \in X$, $(f^{(n)}(x))_{n\in\mathbb{N}}$ forme une partie dirigée. Par continuité de f, on a alors :

$$f \circ f^\omega(x) = f\left(\bigvee_{n\in\mathbb{N}} f^{(n)}(x)\right) = \bigvee_{n\in\mathbb{N}} f^{(n+1)}(x) \sqsubseteq f^\omega(x),$$

d'où $f \circ f^\omega \sqsubseteq f^\omega$. On en déduit $\forall n,\ f^{(n)} \circ f^\omega \sqsubseteq f^\omega$ par récurrence puis $f^\omega \circ f^\omega \sqsubseteq f^\omega$ par (2.24). L'extensivité de f^ω nous donne l'autre inégalité.

2. Pour l'implication directe, on montre $\forall n \in \mathbb{N},\ f^{(n)} = f$ par récurrence ; le cas de base provient de l'extensivité, le cas inductif de l'idempotence. Pour la réciproque, l'extensivité est triviale, et l'on a $f \circ f \sqsubseteq f^\omega = f$, ce qui mène à l'idempotence par extensivité. ∎

Enfin, remarquons qu'il existe des fonctions croissantes et extensives (mais non continues) telles que f^ω ne soit pas idempotente : considérons par exemple la fonction suivante, définie sur le treillis complet des ensembles

d'entiers naturels [1] :

$$Y \mapsto \begin{cases} \mathbb{N} & \text{si } (\mathbb{N} \setminus \{0\}) \subseteq Y, \\ Y \cup \{\min((\mathbb{N} \setminus \{0\}) \setminus Y)\} & \text{sinon .} \end{cases}$$

C'est une fonction croissante et extensive, mais on a :

$$f^\omega(\{1\}) = \mathbb{N} \setminus \{0\} \neq \mathbb{N} = f(f^\omega(\{1\})) = (f^\omega \circ f^\omega)(\{1\}) \ .$$

Si l'on regarde les fonctions idempotentes comme des calculateurs de points fixes (pour tout x, $f(x)$ est toujours un point fixe d'une telle fonction f), ce genre de cas pathologiques correspond à des fonctions qu'il faut fermer plusieurs fois par itération $(((f^\omega)^\omega)^\cdots)$ avant qu'elles ne se « stabilisent ». La proposition 2.35(1) fournit donc une façon de simplifier les calculs et les énoncés : stabiliser des fonctions continues ne nécessite qu'une seule fermeture par itération.

2.2 Points fixes, co-induction, techniques modulo

On s'intéresse maintenant aux objets définis par *co-induction* dans un treillis complet. Cette méthode, sur laquelle repose la définition de la bisimilarité, s'appuie sur le théorème de Knaster–Tarski (établi par Bronisław Knaster et Alfred Tarski en 1927 [Kna28] dans le cas particulier de fonctions d'ensembles, puis généralisé aux fonctions d'un treillis complet en 1939 par Tarski [Tar55]) :

> *« Dans un treillis complet, l'ensemble des points fixes d'une fonction croissante forme un treillis complet. En particulier, toute fonction croissante admet un plus petit et un plus grand point fixe ».*

La preuve de ce théorème passe par la notion de *post-point fixe*, qui est en fait celle qui nous intéresse. Nous commençons cette section en retrouvant la notion de co-induction à travers celle de post-point fixe d'une fonction croissante, puis nous développons une théorie des techniques modulo, permettant de raffiner la méthode de preuve par co-induction. Cette théorie est définie dans un treillis complet quelconque, qui est enrichi dans un second temps par une structure de monoïde.

On suppose désormais fixé un treillis complet $\langle X, \sqsubseteq, \bigvee \rangle$. Dans la suite, nous ne considérons que des fonctions croissantes : nous travaillons implicitement dans le treillis monoïdal $\langle X^{\langle X \rangle}, \sqsubseteq, \bigvee, \circ, \mathrm{id}_X \rangle$.

On considère dans la suite une fonction croissante s, que l'on appellera *génératrice*. Afin de guider l'intuition, on utilise le vocabulaire des algèbres de processus pour nommer les post-points fixes de cette fonction :

1. Merci à Émmanuel Jeandel pour ce contre-exemple.

Définition 2.36 (Simulation, similarité)**.** On appelle *s-simulation* tout post-point fixe de s, c'est-à-dire, tout élément $x \in X$ satisfaisant $x \sqsubseteq s(x)$.

On note X_s l'ensemble des s-simulations. La *s-similarité*, notée $\boldsymbol{\nu} s$, est la borne supérieure de cet ensemble :

$$X_s \triangleq \{x \in X \mid x \sqsubseteq s(x)\} \ ,$$
$$\boldsymbol{\nu} s \triangleq \bigvee X_s \ .$$

On dira dans la suite que s *génère* les s-simulations (X_s) et la s-similarité $(\boldsymbol{\nu} s)$, d'où le nom attribué à cette fonction.

Proposition 2.37. *La borne supérieure d'un ensemble de s-simulations est une s-simulation.*

Démonstration. Soit $Y \subseteq X_s$. Pour tout $y \in Y$, on a :

$$\begin{aligned} y &\sqsubseteq \bigvee Y \\ s(y) &\sqsubseteq s\left(\bigvee Y\right) && (s \text{ est croissante}) \\ y \sqsubseteq s(y) &\sqsubseteq s\left(\bigvee Y\right) && (\text{définition de } X_s) \end{aligned}$$

d'où $\bigvee Y \sqsubseteq s\left(\bigvee Y\right)$, c'est-à-dire $\bigvee Y \in X_s$. ■

$\langle X_s, \sqsubseteq, \bigvee \rangle$ est donc un treillis complet. Attention cependant : la borne inférieure de deux s-simulations n'étant pas nécessairement une s-simulation, les bornes inférieures prises dans X ne coïncident pas avec celles prises dans X_s. Dans le théorème de Knaster–Tarski, qui considère les points fixes et non pas les post-points fixes, cela se traduit par le fait que le treillis complet des points fixes n'est pas un *sous-treillis* : seuls son plus grand élément et son plus petit élément sont toujours obtenus avec les bornes supérieures et inférieures de X.

Le théorème suivant est une reformulation de l'étape principale de la preuve du théorème de Knaster–Tarski : le plus grand point fixe d'une fonction croissante est obtenu en considérant la borne supérieure de l'ensemble de ses post-points fixes :

Théorème 2.38 (Knaster-Tarski)**.** *La s-similarité est une s-simulation,*

$$\boldsymbol{\nu} s = s(\boldsymbol{\nu} s) \ . \tag{2.33}$$

Démonstration. Le fait que la s-similarité soit une s-simulation est une conséquence directe de la proposition 2.37 ; on en déduit l'inégalité directe. Montrons l'autre inégalité :

$$\begin{aligned} \boldsymbol{\nu} s &\sqsubseteq s(\boldsymbol{\nu} s) && (\text{point précédent}) \\ s(\boldsymbol{\nu} s) &\sqsubseteq s(s(\boldsymbol{\nu} s)) && (\text{croissance de } s) \\ s(\boldsymbol{\nu} s) &\in X_s && (\text{par définition de } X_s) \end{aligned}$$

d'où $s(\boldsymbol{\nu} s) \sqsubseteq \bigvee X_s \triangleq \boldsymbol{\nu} s$. ■

La conséquence immédiate qui nous intéresse dans la suite est que la s-similarité est la plus grande s-simulation : la définition de la s-similarité est donc une définition *co-inductive*, qui mène à la méthode de preuve par co-induction donnée ci-dessous. La formulation de cette méthode dans le cadre abstrait d'un treillis complet quelconque peut sembler étrange : il faut se convaincre qu'il est utile de caractériser l'ensemble des éléments majorés par la s-similarité. Sa formulation dans le treillis complet des relations binaires est plus intuitive : *une relation est contenue dans la s-similarité si et seulement si elle est contenue dans une s-simulation.*

Corollaire 2.39 (Preuve par co-induction)**.** *Un élément est majoré par la s-similarité si et seulement si il est majoré par une s-simulation.*

Démonstration. Si $x \sqsubseteq \boldsymbol{\nu} s$, x est majoré par une s-simulation : $\boldsymbol{\nu} s$ en est une. Réciproquement, si $x \sqsubseteq y \sqsubseteq s(y)$, alors $y \sqsubseteq \boldsymbol{\nu} s$ par définition de $\boldsymbol{\nu} s$, et $x \sqsubseteq \boldsymbol{\nu} s$ par transitivité. ■

Exemple 2.40 (Génératrice de la bisimilarité forte)**.** Afin de définir la bisimilarité forte, dans le cas de relations binaires, on peut considérer la génératrice suivante :

$$b : R \mapsto \big\{ \langle p, q\rangle \mid \forall \alpha, p',\ p \xrightarrow{\alpha} p' \text{ implique } \exists q', q \xrightarrow{\alpha} q' \text{ et } p'\ R\ q', \\ \text{et } \ \forall \alpha, q',\ q \xrightarrow{\alpha} q' \text{ implique } \exists p', p \xrightarrow{\alpha} p' \text{ et } p'\ R\ q' \ \big\}$$

On vérifie aisément que cette fonction est croissante, et que les b-simulations sont les bisimulations fortes (au sens usuel, donné dans l'introduction, et que l'on définira formellement au chapitre 3).

La définition suivante nous servira dans la remarque suivante, puis assez fréquemment tout au long de la thèse :

Définition 2.41 (Elément clos)**.** Un élément $x \in X$ est *clos* par une fonction f si c'est un pré-point fixe de f : $f(x) \sqsubseteq x$.

Remarque 2.42 (Induction)*.* De manière duale, si l'on considère le plus petit pré-point fixe de s, couramment noté $\boldsymbol{\mu} s$, on obtient naturellement une définition *inductive* : dans le treillis des parties par exemple, l'ensemble $\boldsymbol{\mu} s$ est le plus petit ensemble E clos par s ; on retrouve donc bien la méthodologie employée pour définir un ensemble de termes par une grammaire, ou une relation de réduction par des règles d'inférence.

La méthode de *preuve par induction* peut alors s'exprimer ainsi : *tout élément clos par s majore $\boldsymbol{\mu} s$.* Dans le treillis des parties, où un énoncé φ peut être considéré comme la partie des objets qui le satisfont, on obtient exactement le principe d'induction : *si φ est clos vis-à-vis de s, alors φ est vrai sur tout l'ensemble inductif $\boldsymbol{\mu} s$.*

Dans la suite, nous allons développer une théorie des techniques modulo pour la co-induction. Par dualité, cette théorie fournit donc aussi des techniques modulo pour l'induction... il n'est cependant pas clair que cela ait un quelconque intérêt.

2.2.1 Techniques modulo

On vient de voir qu'à toute fonction croissante s est associée un objet coinductif, ainsi qu'une méthode de preuve. Nous rentrons maintenant dans le vif du sujet, en étudiant comment raffiner cette méthode. Remarquons tout d'abord que plus la génératrice s est « contraignante » plus la s-similarité est « petite » :

Proposition 2.43. *Soit s' une seconde fonction croissante.*

$$\textit{Si } s' \sqsubseteq s \ , \ \textit{alors } X_{s'} \subseteq X_s \ \textit{et } \boldsymbol{\nu} s' \sqsubseteq \boldsymbol{\nu} s \ .$$

Démonstration. Soit x une s'-simulation. On a $x \sqsubseteq s'(x) \sqsubseteq s(x)$, ce qui fait de x une s-simulation, d'où $X_{s'} \subseteq X_s$. On conclut avec (2.7). ■

Cela correspond à l'intuition que la génératrice s est utilisée pour caractériser une propriété ($\boldsymbol{\nu} s$) (en pratique, le domaine (X) sera l'ensemble des relations binaires sur un ensemble d'états, et les génératrices caractériserons de telles relations binaires : elle généreront les différents préordres et équivalences comportementaux entre les états). Ainsi, lorsque $s' \sqsubseteq s$, il est plus difficile d'exhiber des s'-simulations que des s-simulations, et l'on caractérise donc a priori une propriété plus forte avec s' qu'avec s. L'idée des techniques modulo va dans l'autre sens : elle consiste à définir des génératrices moins contraignantes, mais correspondant néanmoins à des propriétés au moins aussi fortes que la propriété initiale. Plus formellement, on cherche à remplacer la génératrice s par une seconde génératrice s', telle que :

(i) $s \sqsubseteq s'$, afin qu'il y ait plus de s'-simulations que de s-simulations ; et

(ii) $\boldsymbol{\nu} s' \sqsubseteq \boldsymbol{\nu} s$, afin que la méthode reste correcte.

Dans un premier temps, on se restreint à des fonctions de la forme $s \circ f$ et on concentre notre attention sur la fonction f :

Définition 2.44 (Simulation modulo)**.** Soit f une fonction croissante. On appelle *s-simulation modulo f* toute $(s \circ f)$-simulation.

Pour satisfaire (i), il suffit de prendre f extensive, ce qui n'est pas problématique ; on s'attache donc à trouver des conditions sur f telles que $s \circ f$ satisfasse (ii).

Définition 2.45 (Fonction correcte, correcte via)**.** Soient f, f' deux fonctions croissantes.

– f est *correcte pour s* si $\boldsymbol{\nu}\,(s \circ f) \sqsubseteq \boldsymbol{\nu} s$,

– f est *correcte pour s, via f'* si f' est extensive et $f'(X_{s\circ f}) \subseteq X_s$,

La notion de correction correspond exactement à (ii) : f est correcte si et seulement si la similarité définie par $s' = s \circ f$ est majorée par la s-similarité, c'est-à-dire, si toute simulation modulo f est majorée par la s-similarité. Comme le montre la proposition suivante, la nuance apportée par la notion de « correction via » est minime :

Proposition 2.46. *Toute fonction correcte pour s via une fonction quelconque est correcte pour s. Toute fonction correcte pour s est correcte pour s via la fonction* $\mathrm{id}_X \vee \widehat{\boldsymbol{\nu}s}$.

Démonstration. – Soit f une fonction correcte pour s via une seconde fonction f'. On a :

$$\begin{aligned} \boldsymbol{\nu}\,(s \circ f) &\in X_{s\circ f} \ , && \text{(théorème 2.38)} \\ f'(\boldsymbol{\nu}\,(s \circ f)) &\in X_s \ , && \text{(correction de } f \text{ pour } s \text{ via } f') \\ f'(\boldsymbol{\nu}\,(s \circ f)) &\sqsubseteq \boldsymbol{\nu}s \ , && \text{(par définition de } \boldsymbol{\nu}s) \end{aligned}$$

L'extensivité de f' (requise dans la définition de correction via) nous permet de conclure : $\boldsymbol{\nu}\,(s \circ f) \sqsubseteq f'(\boldsymbol{\nu}\,(s \circ f)) \sqsubseteq \boldsymbol{\nu}s$.

– Inversement, posons $f' = \mathrm{id}_X \vee \widehat{\boldsymbol{\nu}s}$, qui est une fonction extensive. Si f est correcte pour s, alors pour tout $x \in X_{s\circ f}$, on a

$$x \sqsubseteq \boldsymbol{\nu}\,(s \circ f) \sqsubseteq \boldsymbol{\nu}s \ ,$$

d'où $f'(x) = x \vee \boldsymbol{\nu}s = \boldsymbol{\nu}s \in X_s$. ■

Intuitivement, une fonction est correcte pour s via f' si sa correction pour s peut être prouvée en utilisant f' comme une fonction « témoin », qui exhibe la s-simulation contenant x. Cette intuition est illustrée par la preuve de la proposition 2.49. La fonction témoin exhibée dans la proposition précédente est en quelque sorte maximale : elle n'apporte pas d'information. L'intérêt de cette fonction apparaîtra lorsque cette fonction sera plus précise : ce sera essentiel dans la suite, pour établir les théorèmes 2.57 et 2.62, où différentes fonctions témoins doivent coïncider.

Le lemme trivial suivant sera parfois utilisé ; notons que cette propriété ne tient pas du côté de la fonction témoin.

Lemme 2.47. *Soient f, g, h trois fonctions, si $f \sqsubseteq g$ et si g est correcte pour s via h, alors f est correcte pour s via h.*

Définition 2.48 (Fonction compatible)**.** Une fonction croissante f est *compatible avec s* si

$$f \circ s \sqsubseteq s \circ f \ .$$

La notion de compatibilité est extrêmement simple (elle ne fait notamment pas intervenir l'opérateur de point fixe, $\boldsymbol{\nu}$), et s'éloigne de la condition (ii). La proposition suivante montre néanmoins que c'est une condition suffisante :

Proposition 2.49. *Toute fonction f compatible avec s est correcte pour s via f^ω.*

Démonstration. Soit f une fonction compatible avec s ; f^ω est extensive par définition, montrons $f^\omega(X_{s\circ f}) \subseteq X_s$. Soit $x \in X_{s\circ f}$ une s-simulation modulo f ; on démontre tout d'abord la propriété suivante, par récurrence sur n :

$$\forall n,\ f^{(n)}(x) \sqsubseteq s(f^{(n+1)}(x))\ .$$

– Si $n = 0$, il s'agit de l'hypothèse sur x ;
– sinon, on a :

$$\begin{aligned} f^{(n-1)}(x) &\sqsubseteq s(f^{(n)}(x)) && \text{(hypothèse de récurrence)}\\ f^{(n)}(x) &\sqsubseteq f(s(f^{(n)}(x))) && \text{(croissance de } f\text{)}\\ &\sqsubseteq s(f(f^{(n)}(x))) = s(f^{(n+1)}(x))\ . && \text{(compatibilité de } f\text{)} \end{aligned}$$

Finalement,

$$\begin{aligned} \forall n,\ f^{(n)}(x) &\sqsubseteq s(f^\omega(x)) && \text{((2.4) et croissance de } s\text{)}\\ f^\omega(x) &\sqsubseteq s(f^\omega(x))\ . && \text{(par (2.5))} \end{aligned}$$

c'est-à-dire, $f^\omega(x) \in X_s$. ■

Bien que sous une forme assez différente, cette dernière proposition est essentiellement présente dans [San98] (cf. remarque 2.80) ; elle peut aussi être vue comme une instance de la « règle d'échange » que l'on trouve dans les calculs de point fixes (cette règle est donnée, entre autres, dans [DP90, page 191], où il suffit de prendre l'identité pour la connection de Galois puis de dualiser).

On verra au chapitre 4 qu'il existe des fonctions correctes qui ne sont pas compatibles : la notion de compatibilité n'est pas complète vis-à-vis de (ii). L'avantage majeur des fonctions compatibles sur les fonctions correctes est que les premières peuvent se composer de façon modulaire, grâce aux résultats suivants de fermeture de la famille des fonctions compatibles :

Proposition 2.50. *Les fonctions suivantes sont compatibles avec s :*

1. *l'identité* (id_X) *;*
2. *les fonctions constantes $\widehat{x}$, pour toute s-simulation x ;*
3. *$f \circ g$, pour toutes fonctions f et g compatibles avec s ;*

4. $\bigvee F$, *pour toute famille* F *de fonctions compatibles avec* s ;
5. $f^{(n)}$ *pour tout entier* n *et toute fonction* f *compatible avec* s ;
6. f^ω, *pour toute fonction* f *compatible avec* s.

Démonstration. 1. On a $\mathrm{id}_X \circ s = s \circ \mathrm{id}_X$.

2. Si $x \in X_s$: $x \sqsubseteq s(x)$, alors $\widehat{x} \circ s = \widehat{x} \sqsubseteq \widehat{s(x)} = s \circ \widehat{x}$.
3. On a $f \circ g \circ s \sqsubseteq f \circ s \circ g \sqsubseteq s \circ f \circ g$.
4. En utilisant le lemme 2.14, $(\bigvee F) \circ s = \bigvee_{f \in F} (f \circ s) \sqsubseteq \bigvee_{f \in F} (s \circ f) \sqsubseteq s \circ \bigvee F$.
5. Par récurrence sur n, en utilisant (1) et (3).
6. Par les deux points précédents. ■

De telles propriétés de fermeture ne tiennent pas pour les fonctions correctes (ou correctes via) : on donnera un contre-exemple au chapitre 4. Plus généralement, la fonction $t = \bigvee \{t \mid \boldsymbol{\nu} t \sqsubseteq \boldsymbol{\nu} s\}$ ne satisfait pas nécessairement $\boldsymbol{\nu} t \sqsubseteq \boldsymbol{\nu} s$.

Comme on le verra dans la section 2.3, où l'on donne une présentation alternative des résultats précédents, on a pour l'instant seulement défini une généralisation très abstraite de la théorie des techniques modulo pour la bisimilarité forte définie dans [San98]. Le premier résultat qui n'a pas de contrepartie dans [San98] est le théorème 2.52 ci-dessous, qui donne une condition suffisante pour que la composition d'une fonction correcte avec une fonction compatible reste correcte. Sa preuve passe par le lemme élémentaire suivant :

Lemme 2.51. *Soient* f, g *deux fonctions. Si* f *est compatible avec* s *et* g, *alors elle est compatible avec* $s \circ g$.

Démonstration. On a $f \circ s \circ g \sqsubseteq s \circ f \circ g \sqsubseteq s \circ g \circ f$. ■

Théorème 2.52 (Composition des fonctions correctes et compatibles)**.** *Soit* f *une fonction compatible avec* s, *et* g *une fonction correcte pour* s *via* g'. *Si* f *est compatible avec* g, *alors* $g \circ f$ *est correcte pour* s *via* $(g' \circ f^\omega)$.

Démonstration. Par le lemme 2.51, f est compatible avec $s \circ g$, d'où

$$f^\omega(X_{s \circ g \circ f}) \subseteq X_{s \circ g} \ . \qquad \text{(proposition 2.49)}$$

Comme g est correcte pour s via g' on peut conclure :

$$g'(f^\omega(X_{s \circ g \circ f})) \subseteq g'(X_{s \circ g}) \subseteq X_s \ . \qquad \blacksquare$$

En prenant pour g la fonction identité, qui est compatible avec toute fonction, et correcte pour toute fonction via elle-même, le théorème précédent peut se voir comme une généralisation de la proposition 2.49. Une autre lecture de ce théorème consiste à isoler la fonction $s_g = s \circ g$:

– la correction de g pour s nous donne $\boldsymbol{\nu} s_g \sqsubseteq \boldsymbol{\nu} s$;
– le lemme 2.51 permet ensuite d'obtenir qu'une partie des fonctions compatibles avec s sont compatibles avec s_g : celles qui sont aussi compatibles avec g.

Bien que ce théorème ne soit pas difficile, il s'avère fondamental pour l'étude des techniques modulo : comme nous l'illustrerons au chapitre 4, il permet de se concentrer sur le coeur d'une technique modulo potentiellement compliquée (la fonction correcte g), sans se soucier des améliorations possibles, qui pourront être apportées par la suite, sous la forme de fonctions compatibles (la fonction compatible f, qui peut en regrouper plusieurs, par modularité). Sans le théorème 2.52, le manque de modularité de la notion de fonction correcte nous obligerait à prouver à nouveau chaque extension possible de la technique compliquée.

2.2.2 Lien avec des propriétés de clôture

Même lorsque deux fonctions s et s' définissent la même similarité (i.e., lorsque $\boldsymbol{\nu} s = \boldsymbol{\nu} s'$), les fonctions compatibles ou correctes leur correspondant peuvent être complètement différentes. On s'en apercevra au chapitre 3.2, puis au chapitre 5, lorsque l'on considérera les techniques modulo contexte pour la bisimulation faible. Les notions de compatibilité et de correction dépendent donc bien de la génératrice (s), plutôt que de la s-similarité ($\boldsymbol{\nu} s$).

On peut néanmoins essayer d'obtenir des indices sur les fonctions compatibles et correctes potentiellement supportées par s, à partir de la propriété que cette génératrice caractérise ($\boldsymbol{\nu} s$) :

Proposition 2.53. *Soient f, f' deux fonctions.*

1. *Si f est compatible avec s, alors f préserve les s-simulations, et la s-similarité est close par f :*

$$f(X_s) \subseteq X_s \ , \qquad f(\boldsymbol{\nu} s) \sqsubseteq \boldsymbol{\nu} s \ .$$

2. *Si f est extensive et correcte pour s via f', alors :*

$$f'(X_s) \subseteq X_s \ , \qquad f'(\boldsymbol{\nu} s) \sqsubseteq \boldsymbol{\nu} s \ .$$

Démonstration. 1. Soit $x \in X_s$, on a $x \sqsubseteq s(x)$. Par compatibilité de f avec s, on a donc $f(x) \sqsubseteq f(s(x)) \sqsubseteq s(f(x))$, i.e., $f(x) \in X_s$. Le second point en découle puisque $\boldsymbol{\nu} s$ est la plus grande s-simulation.

2. Soit $x \in X_s$, on a $x \sqsubseteq s(x) \sqsubseteq s(f(x))$ (f est extensive et s croissante), d'où $x \in X_{s \circ f}$, puis $f'(x) \in X_s$ par la correction de f pour s via f'. Le second point en découle, f' étant nécessairement extensive. ■

Le premier point indique que les fonctions compatibles avec s correspondent nécessairement à des propriétés de fermeture satisfaites par la s-similarité. On verra au chapitre 5 que ce n'est pas une condition suffisante : il existe des fonctions telles que $f(\boldsymbol{\nu} s) \sqsubseteq \boldsymbol{\nu} s$ mais qui ne sont pas correctes pour s (et qui ne sont donc pas non plus compatibles avec s). Bien que nous n'ayons pas de contre-exemple, le fait de préserver l'ensemble des s-simulations n'est a priori pas non plus une condition suffisante.

Le second point est plus anecdotique, puisque la propriété de fermeture est requise pour la fonction témoin (f'). Le résultat ne tient pas pour les fonctions correctes elles-mêmes : si l'on prend pour s la fonction constante $\widehat{\bot}$, toute fonction f est correcte pour s : $\boldsymbol{\nu}\left(\widehat{\bot} \circ f\right) = \boldsymbol{\nu}\widehat{\bot} = \bot$, mais dès que le treillis possède deux éléments, la s-similarité ($\bot$) n'est pas close par la fonction $\widehat{\top}$.

2.2.3 Conjonctions

Les bornes inférieures du treillis complet initial fournissent naturellement des bornes inférieures dans l'espace des fonctions. On s'intéresse dans cette sous-section à la caractérisation des techniques modulo disponibles pour une fonction obtenue comme la borne inférieure de plusieurs fonctions élémentaires. Comme nous le montrerons dans le chapitre suivant, à la section 3.3, ce genre de fonctions correspond naturellement à des définitions co-inductives obtenues par la conjonction de plusieurs propriétés.

On fixe donc dans la suite un ensemble S de génératrices (de fonctions croissantes), et on étudie les techniques modulo pour la $\bigwedge S$-similarité. Par exemple, au chapitre suivant, la bisimilarité forte sera obtenue à l'aide de l'ensemble $S_{\mathbf{s}} = \{\mathbf{s}, \overline{\mathbf{s}}\}$, où $\mathbf{s}$ sera la génératrice des simulations fortes « de gauche à droite » et sa converse ($\overline{\mathbf{s}}$) celle des simulations fortes « de droite à gauche » ; ainsi, la bisimilarité forte est $\boldsymbol{\nu}(\mathbf{s} \wedge \overline{\mathbf{s}})$ (la plus grande relation qui soit à la fois une simulation de gauche à droite, et de droite à gauche), tandis que la 2-similarité forte correspond à la conjonction $\boldsymbol{\nu}\mathbf{s} \wedge \boldsymbol{\nu}\overline{\mathbf{s}}$. La proposition suivante donne dans ce cas l'inclusion de la bisimilarité dans la 2-similarité.

Proposition 2.54. *On a :*

$$X_{\bigwedge S} = \bigcap_{s \in S} X_s \ , \ \textit{et} \ \ \boldsymbol{\nu}\bigwedge S \sqsubseteq \bigwedge_{s \in S} \boldsymbol{\nu} s \ .$$

Démonstration. Pour le premier point, on a :

$$\begin{aligned} x \in X_{\bigwedge S} \Leftrightarrow x \sqsubseteq \bigwedge S(x) &\Leftrightarrow \forall s \in S,\ x \sqsubseteq s(x) \\ &\Leftrightarrow \forall s \in S,\ x \in X_s \Leftrightarrow x \in \bigcap_{s \in S} X_s \ . \end{aligned}$$

Ensuite, par (2.14), on a :

$$\boldsymbol{\nu}\bigwedge S = \bigvee X_{\bigwedge S} = \bigvee \bigcap_{s\in S} X_s \sqsubseteq \bigwedge_{s\in S} \bigvee X_s = \bigwedge_{s\in S} \boldsymbol{\nu} s \ . \qquad \blacksquare$$

Remarque 2.55. Comme nous l'avons rappelé en introduction, la réciproque n'est pas vraie : la 2-similarité n'est en général pas contenue dans la bi-similarité. Une conséquence importante en pratique est qu'il ne suffit pas d'étudier indépendamment les techniques modulo pour chacun des points fixes $(\boldsymbol{\nu} s)_{s\in S}$, pour obtenir automatiquement des techniques modulo pour $\boldsymbol{\nu}\bigwedge S$.

Proposition 2.56. *Toute fonction compatible avec chacune des génératrices de S est compatible avec $\bigwedge S$.*

Démonstration. Soit f une telle fonction.

$$\begin{aligned} f \circ S &\sqsubseteq \bigwedge_{s\in S} (f \circ s) && \text{(par (2.32))} \\ &\sqsubseteq \bigwedge_{s\in S} (s \circ f) && \text{(compatibilité et (2.11))} \\ &= S \circ f \ . && \text{(dual de (2.24))} \end{aligned}$$

$\blacksquare$

Remarquons que cette proposition est en fait la duale de la proposition 2.50(4) : « la borne supérieure d'un ensemble de fonctions compatibles avec s est compatible avec s ».

La proposition 2.56 traite le cas de fonctions compatibles, et impose que la même technique soit utilisée pour chacune des composantes de S. On peut s'affranchir de ces restrictions en travaillant avec des fonctions correctes, à condition qu'elles « s'accordent » sur une fonction témoin commune :

Théorème 2.57. *Soit $(f_s)_{s\in S}$ une famille de fonctions indexée par S ; soit f' une fonction extensive. Posons $s' \triangleq \bigwedge \{s \circ f_s \mid s \in S\}$.*

Si pour tout $s \in S$, f_s est correcte pour s via f', alors $\boldsymbol{\nu} s' \sqsubseteq \boldsymbol{\nu}\bigwedge S$.

Démonstration. Par le théorème 2.38, on a $\boldsymbol{\nu} s' = s'(\boldsymbol{\nu} s')$. Ensuite,

$$\begin{aligned} \forall s \in S,\ \boldsymbol{\nu} s' &\sqsubseteq (s \circ f_s)\,(\boldsymbol{\nu} s') \\ \forall s \in S,\ f'(\boldsymbol{\nu} s') &\in X_s && (f_s \text{ est correcte pour } s \text{ via } f') \\ f'(\boldsymbol{\nu} s') &\in X_{\bigwedge S} && \text{(proposition 2.54)} \end{aligned}$$

f' étant extensive, on peut conclure : $\boldsymbol{\nu} s' \sqsubseteq f'(\boldsymbol{\nu} s') \sqsubseteq \boldsymbol{\nu}\bigwedge S$. $\blacksquare$

Bien que le théorème précédent ne définisse pas une fonction correcte pour $\bigwedge S$, il nous donne bien une technique modulo pour $\bigwedge S$: a priori, $\bigwedge S \sqsubseteq s'$, de telle sorte que les s'-simulations sont plus faciles à construire que les $\bigwedge S$-simulations.

Ce théorème sera utilisé pour obtenir le théorème 2.62 (où l'accordage des fonctions témoin est remplacé par une condition équivalente de symétrie) ainsi qu'au chapitre suivant, pour définir obtenir des techniques modulo pour l'expansion. C'est pour établir ce résultat qu'il nous faut considérer la notion de correction via : il nous faut pouvoir parler des fonctions témoins afin de pouvoir les accorder. Le lemme suivant pourra être utilisé à cette fin : les simulation exhibées par une fonction témoin peuvent être « agrandies » à l'aide de toute fonction préservant les simulations :

Lemme 2.58. *Soit f, f', g' trois fonctions.*
Si $\begin{cases} f \text{ est correcte pour } s \text{ via } f', \\ g' \text{ est extensive et préserve les } s\text{-simulations } (g'(X_s) \subseteq X_s), \end{cases}$
alors f est correcte pour s via $(g' \circ f')$.

2.2.4 Symétrie

On étudie maintenant des propriétés de symétrie, qui nous permettront, avec les conjonctions précédemment étudiées, de ramener au chapitre suivant l'étude de la bisimulation à celle de la simulation.

On suppose pour cela une fonction croissante et involutive i, dite *de conversion*, Dans le treillis complet monoïdal des relations binaires, cette fonction sera celle qui à toute relation associe la relation qui en inverse tous les couples.

Définition 2.59 (Converses, fonction symétrisée)**.** Soient x un élément de X, Y une partie de X et f une fonction, on définit comme suit les *converses* de x, Y et f, et la *symétrisée* de f :

$$\overline{x} \triangleq i(x) \qquad \overline{Y} \triangleq i(Y) \qquad \overline{f} \triangleq i \circ f \circ i \qquad \overleftrightarrow{f} \triangleq f \wedge \overline{f} \ .$$

L'élément x (resp. la fonction f) est dit(e) *symétrique* si $x = \overline{x}$ (resp. $f = \overline{f}$).

La fonction i est en fait un auto-morphisme de treillis, ce qui lui confère de bonnes propriétés :

Lemme 2.60. *Soient x, y deux éléments de X, Y une partie de X, et f, g deux fonctions croissantes.*

1. $\overline{\overline{x}} = x$; $\overline{\overline{Y}} = Y$; $\overline{\overline{f}} = f$;
2. $\overline{f(x)} = \overline{f}(\overline{x})$; $\overline{f \circ g} = \overline{f} \circ \overline{g}$; $\overline{\mathrm{id}_X} = \mathrm{id}_X$; $\widehat{\overline{x}} = \widehat{\overline{x}}$.
3. $x \sqsubseteq y$ *si et seulement si* $\overline{x} \sqsubseteq \overline{y}$; $f \sqsubseteq g$ *si et seulement si* $\overline{f} \sqsubseteq \overline{g}$;

4. $\forall Y \subseteq X,\ \overline{\bigvee Y} = \bigvee \overline{Y}$;
5. $\forall Y \subseteq X,\ \overline{\bigwedge Y} = \bigwedge \overline{Y}$;
6. $\overline{f}$ *est croissante si et seulement si* f *l'est.*
7. $\overline{f^\omega} = \left(\overline{f}\right)^\omega$.
8. $\overleftrightarrow{f}$ *est symétrique.*

Démonstration. (1) et (2) sont immédiats puisque i est une involution.

3. On utilise la croissance de i pour les implications directes, et le point précédent pour le retour.
4. Soit $Y \subseteq X$; pour tout $y \in Y$, on a $y \sqsubseteq \bigvee Y$, puis $\overline{y} \sqsubseteq \overline{\bigvee Y}$ par le point précédent. On en déduit $\bigvee \overline{Y} \sqsubseteq \overline{\bigvee Y}$. L'inégalité réciproque s'obtient en appliquant le raisonnement précédent à $\overline{Y}$: on a $\bigvee Y \sqsubseteq \overline{\bigvee \overline{Y}}$, puis $\overline{\bigvee Y} \sqsubseteq \bigvee \overline{Y}$ par (1) et (3).
5. Par dualité avec le point précédent (les hypothèses faites sur i sont invariantes par dualisation du treillis complet).
6. Par composition de fonctions croissantes.
7. En utilisant (2), on montre par récurrence $\forall n \in \mathbb{N},\ \overline{f^{(n)}} = (\overline{f})^{(n)}$. On conclut avec (4)
8. De (5), on déduit $\overleftrightarrow{\overline{f}} \triangleq \overline{f \wedge \overline{f}} = \overline{f} \wedge \overline{\overline{f}} = \overline{f} \wedge f = \overleftrightarrow{f}$. ■

En utilisant les propriétés précédentes, on peut aisément mettre en relation les techniques modulo associées aux génératrices s et $\overline{s}$. Comme indiqué plus haut, dans le cadre du chapitre suivant, si s définit un jeu de simulation « de gauche à droite », $\overline{s}$ définit le même jeu de simulation mais « de droite à gauche », et la fonction $\overleftrightarrow{s}$ définira naturellement dans ce cas le jeu de bisimulation correspondant.

Proposition 2.61. *Soient* f, f' *deux fonctions croissantes.*

1. $\overline{X_s} = X_{\overline{s}}$;
2. $\overline{\boldsymbol{\nu} s} = \boldsymbol{\nu} \overline{s}$;
3. f *est correcte pour* s *si et seulement si* $\overline{f}$ *est correcte pour* $\overline{s}$;
4. f *est compatible avec* s *si et seulement si* $\overline{f}$ *est compatible avec* $\overline{s}$;
5. *Si* f *est correcte pour* s *via* f', *alors* $\overline{f}$ *est correcte pour* $\overline{s}$ *via* $\overline{f'}$;
6. *Si* f *préserve les* s*-simulations, alors* $\overline{f}$ *préserve les* $\overline{s}$*-simulations.*

Démonstration.
1. $x \in X_s \Leftrightarrow x \sqsubseteq s(x) \Leftrightarrow \overline{x} \sqsubseteq \overline{s(x)} \Leftrightarrow \overline{x} \sqsubseteq \overline{s}(\overline{x}) \Leftrightarrow \overline{x} \in X_{\overline{s}}$.
2. $\overline{\boldsymbol{\nu} s} \triangleq \overline{\bigvee X_s} = \bigvee \overline{X_s} = \bigvee X_{\overline{s}} \triangleq \boldsymbol{\nu} \overline{s}$.

Par le lemme 2.60(1), il suffit de montrer l'implication directe dans les cas restants.

3. Si $\boldsymbol{\nu}(s \circ f) \sqsubseteq \boldsymbol{\nu} s$ alors $\boldsymbol{\nu}\left(\overline{s} \circ \overline{f}\right) = \boldsymbol{\nu}\left(\overline{s \circ f}\right) = \overline{\boldsymbol{\nu}(s \circ f)} \sqsubseteq \overline{\boldsymbol{\nu} s} = \boldsymbol{\nu}\overline{s}$.
4. Si $f \circ s \sqsubseteq s \circ f$ alors $\overline{f} \circ \overline{s} = \overline{f \circ s} \sqsubseteq \overline{s \circ f} = \overline{s} \circ \overline{f}$.
5. Si $f'(X_{s \circ f}) \subseteq X_s$ alors $\overline{f'}(X_{\overline{s} \circ \overline{f}}) = \overline{f'}(\overline{X_{s \circ f}}) = \overline{f'(X_{s \circ f})} \subseteq \overline{X_s} = X_{\overline{s}}$.
6. Si $f(X_s) \subseteq X_s$ alors $\overline{f}(X_{\overline{s}}) = \overline{f}(\overline{X_s}) = \overline{f(X_s)} \subseteq \overline{X_s} = X_{\overline{s}}$. ■

On peut finalement combiner les résultats précédents avec le théorème 2.57 et réduire la recherche de techniques modulo pour $\overleftrightarrow{s}$ à celle de techniques modulo pour s. Comme annoncé, on utilisera le théorème ci-dessous au chapitre 3 (section 3.3), pour dériver des techniques modulo pour la bisimilarité à partir de celles obtenues pour la similarité.

Théorème 2.62. *Pour toute fonction f correcte pour s via une fonction symétrique, on a :*

$$\boldsymbol{\nu}\overleftrightarrow{s \circ f} \sqsubseteq \boldsymbol{\nu}\overleftrightarrow{s} \ .$$

Démonstration. Soit f' la fonction symétrique ; par la proposition 2.61, on vérifie que $\left\{f, \overline{f}\right\}$ et $f' = \overline{f'}$ satisfont les hypothèses du théorème 2.57, pour $S = \{s, \overline{s}\}$. ■

Corollaire 2.63. *Soit f une fonction correcte pour s via une fonction symétrique. Toute s-simulation modulo f qui est symétrique est majorée par la $\overleftrightarrow{s}$-similarité.*

Le théorème précédent et son corollaire indiquent qu'il sera utile de pouvoir prouver facilement la symétrie de certains éléments ou fonctions. Ce sera le cas, car la propriété de symétrie est très modulaire ; une façon élégante de le prouver consiste à remarquer que les éléments symétriques sont les i-simulations et que les fonctions croissantes et symétriques sont les fonctions compatibles avec i. On obtient alors immédiatement les résultats suivants :

Lemme 2.64. *L'identité et les fonctions constantes $\widehat{x}$ pour x symétrique sont symétriques ; les bornes supérieures et inférieures, la composition fonctionnelle et la fermeture par itération préservent la symétrie.*

L'image d'un élément symétrique par une fonction symétrique est symétrique.

Démonstration. Mise à part la préservation de la symétrie par les bornes inférieures, qui s'obtient par dualité, c'est un corollaire des propositions 2.50 et 2.53. ■

On peut finalement prouver la symétrie de la $\overleftrightarrow{s}$-similarité, dont la remarque précédente permet par ailleurs de donner une caractérisation alternative : les $(s \wedge i)$-simulations sont les s-simulations symétriques.

Proposition 2.65. *La $\overleftrightarrow{s}$-similarité est symétrique, on a de plus :*

$$\boldsymbol{\nu}^{\overleftrightarrow{s}} = \boldsymbol{\nu}\,(s \wedge i) \quad .$$

Démonstration. Par la proposition 2.61(2), $\overline{\boldsymbol{\nu}^{\overleftrightarrow{s}}} = \boldsymbol{\nu}^{\overleftrightarrow{\overline{s}}} = \boldsymbol{\nu}^{\overleftrightarrow{s}}$.

$\boldsymbol{\nu}^{\overleftrightarrow{s}}$ est donc une s-simulation symétrique, d'où $\boldsymbol{\nu}^{\overleftrightarrow{s}} \sqsubseteq \boldsymbol{\nu}\,(s \wedge i)$. Inversement, $\boldsymbol{\nu}\,(s \wedge i)$ est une $\overline{s}$-simulation par la proposition 2.61(1), ce qui en fait une $\overleftrightarrow{s}$-simulation : $\boldsymbol{\nu}\,(s \wedge i) \sqsubseteq \boldsymbol{\nu}^{\overleftrightarrow{s}}$. ■

2.2.5 Monoïde interne

On s'intéresse maintenant aux propriétés des fonctions compatibles vis-à-vis d'un monoïde quelconque, interne au treillis complet. D'une part, cela nous permettra de capturer de manière relativement uniforme un certain nombre de résultats du chapitre suivant, où le produit correspondra intuitivement à la composition relationnelle. D'autre part, en se plaçant dans le treillis complet des fonctions croissantes, et en prenant comme produit la composition fonctionnelle, cela nous mènera au chapitre 5 à la définition de techniques modulo pour prouver la compatibilité de fonctions « modulo contexte ».

On étend donc notre treillis complet en un treillis complet monoïdal $\langle X, \sqsubseteq, \bigvee, \cdot, 1\rangle$, non nécessairement continu, et on s'intéresse aux propriétés co-inductives relatives à une fonction croissante s. On définit pour cela trois notions de *respect* du monoïde par cette fonction :

Définition 2.66 (Respect, respect à gauche)**.** La fonction s *respecte le monoïde* si 1 est une s-simulation, et

$$\forall x, y \in X,\ s(x) \cdot s(y) \sqsubseteq s(x \cdot y) \ ; \tag{2.34}$$

elle *respecte le monoïde à gauche* si 1 est une s-simulation, et

$$\forall x \in X, \forall y \in X_s,\ s(x) \cdot y \sqsubseteq s(x \cdot y) \ ; \tag{2.35}$$

enfin, elle *respecte le monoïde à droite* si 1 est une s-simulation, et

$$\forall x \in X_s, \forall y \in X,\ x \cdot s(y) \sqsubseteq s(x \cdot y) \ . \tag{2.36}$$

Le respect implique le respect à gauche et à droite (on discute de la réciproque dans la remarque 2.70 ci-dessous), et le respect, d'un côté ou de l'autre, implique la réflexivité et la transitivité de la similarité correspondante. On obtiendra ainsi au chapitre suivant le fait que toutes les notions de (bi)similarité que l'on définira sont des préordres.

Proposition 2.67. *1. Si s respecte le monoïde, alors s respecte le monoïde à gauche comme à droite.*

2. *Si s respecte le monoïde à gauche ou à droite, alors le produit de deux s-simulations est une s-simulation, la s-similarité est réflexive et transitive.*
3. *Pour toute famille S de génératrices, dont chacune respecte le monoïde à gauche ou à droite, la $\bigwedge S$-similarité est réflexive et transitive.*

Démonstration. 1. Si $y \in X_s$, pour tout x, on a :

$$\begin{aligned} s(x) \cdot y &\sqsubseteq s(x) \cdot s(y) && \text{(par (2.17))} \\ &\sqsubseteq s(x \cdot y)\ . && \text{(respect)} \end{aligned}$$

Le cas du respect à droite est bien entendu similaire.

2. Soient $x, y \in X_s$; on a

$$\begin{aligned} x \cdot y &\sqsubseteq s(x) \cdot y && \text{((2.17) et hypothèse sur } x) \\ &\sqsubseteq s(x \cdot y)\ . && \text{(respect d'un côté)} \end{aligned}$$

$x \cdot y$ est donc bien une s-simulation. Ensuite, par hypothèse, $1 \in X_s$, d'où $1 \sqsubseteq \boldsymbol{\nu} s$; puis $\boldsymbol{\nu} s \in X_s$, d'où $\boldsymbol{\nu} s \cdot \boldsymbol{\nu} s \in X_s$ par le point précédent, et $\boldsymbol{\nu} s \cdot \boldsymbol{\nu} s \sqsubseteq \boldsymbol{\nu} s$.

3. Posons $x = \boldsymbol{\nu} \bigwedge S$.
   – pour tout $s \in S$, $1 \in X_s$, d'où $1 \in \bigcap_{s \in S} X_s$ et $1 \sqsubseteq x$;
   – pour tout $s \in S$, $x \in X_s$, d'où $x \cdot x \in X_s$ par le point précédent. On en déduit $x \cdot x \in \bigcap_{s \in S} X_s$, d'où $x \cdot x \sqsubseteq x$. ■

Le respect est une propriété relativement forte, qui se traduit par un très bon comportement des techniques modulo pour s : l'extension point à point du produit préserve la compatibilité avec s, et s supporte la technique « modulo transitivité ». On verra au chapitre suivant que la fonction définissant la simulation forte respecte le monoïde. La proposition suivante permettra donc de retrouver les bonnes propriétés des techniques modulo pour la (bi)simulation forte.

Proposition 2.68. *Si s respecte le monoïde, alors :*

1. *le produit de deux fonctions compatibles avec s est compatible avec s ;*
2. *la fonction de fermeture réflexive transitive $(\mathrm{id}_X^\star)$ est compatible avec s.*

Démonstration. 1. Soient f et g deux fonctions compatibles avec s.

$$\begin{aligned} (f \mathbin{\hat{\cdot}} g) \circ s &= (f \circ s) \mathbin{\hat{\cdot}} (g \circ s) && \text{(par définition)} \\ &\sqsubseteq (s \circ f) \mathbin{\hat{\cdot}} (s \circ g) && \text{((2.17) et compatibilité de } f \text{ et } g) \\ &\sqsubseteq s \circ (f \mathbin{\hat{\cdot}} g)\ . && \text{(respect)} \end{aligned}$$

2. Par la proposition 2.50(4), il suffit de démontrer que id_X^n est compatible avec $s.$ pour tout $n \in \mathbb{N}$. On procède par récurrence sur n :
   – le cas de base, $n = 0$, provient de la proposition 2.50(2) : $\mathrm{id}_X^0 \triangleq \widehat{1}$ est compatible avec s puisque 1 est une s-simulation par hypothèse.
   – sinon, par hypothèse de récurrence, id_X^{n-1} est compatible avec s, tout comme id_X, et il suffit d'appliquer le point précédent pour obtenir la compatibilité de $\mathrm{id}_X^n \triangleq \mathrm{id}_X \mathbin{\hat{\cdot}} \mathrm{id}_X^{n-1}$. ■

Par contre, la génératrice définissant la simulation faible, qui ne supporte pas la technique « modulo transitivité », ne pourra pas respecter entièrement le monoïde. C'est pour traiter ce cas que l'on a introduit la propriété de respect à gauche, qu'aura cette génératrice. Comme le montre la proposition suivante, le respect à gauche implique la validité de la technique « modulo post-composition », qui se traduira par la possibilité d'utiliser la similarité faible pour réécrire le processus de droite dans les jeux de simulation faible :

Proposition 2.69. *Soit x_s une s-simulation,*

1. *Si s respecte le monoïde à gauche, la fonction $x \mapsto x \cdot x_s$ est compatible avec s ;*
2. *si s respecte le monoïde à droite, la fonction $x \mapsto x_s \cdot x$ est compatible avec s.*

Démonstration. 1. Posons $f : x \mapsto x \cdot x_s$, puisque $x_s \in X_s$, pour tout $x \in X$, on a $f(s(x)) = s(x) \cdot x_s \sqsubseteq s(x \cdot x_s) = s(f(x))$.

2. Identique au point précédent. ■

Remarque 2.70. Le fait que le respect implique le respect à gauche et à droite se traduit au niveau des techniques modulo par le fait que « modulo transitivité » donne en particulier « modulo pré- et post-composition ». L'implication réciproque n'est pas vraie en général : dans le cas du treillis complet monoïdal continu de l'exemple 2.19, $\langle \overline{\mathbb{N}}, \leq, \sup, +, 0 \rangle$, la fonction successeur $(n \mapsto n+1)$ respecte le produit $(+)$ à droite comme à gauche, sans toutefois le respecter (on n'a pas $(m+1)+(n+1) \leq (m+n)+1$ pour tous $m, n \in \overline{\mathbb{N}}$) [2]. En particulier malgré le respect à droite et à gauche, le produit de deux fonctions compatibles ne l'est pas nécessairement : l'identité $(\mathrm{id}_{\overline{\mathbb{N}}})$ est compatible avec la fonction successeur, mais $\mathrm{id}_{\overline{\mathbb{N}}} \widehat{+} \mathrm{id}_{\overline{\mathbb{N}}} : n \mapsto n+n$ ne l'est pas.

Cependant, en termes de techniques modulo pour la bisimulation, une telle situation, où « modulo pré- et post-composition » est autorisée mais pas « modulo transitivité » ne semble pas naturelle, nous aimerions donc trouver un contre-exemple dans le treillis complet monoïdal des relations binaires, dans le cas d'une génératrice menant à une notion utile de bisimilarité (dans le contre-exemple précédent la similarité générée par la fonction successeur n'est autre que l'élément maximal...).

2. Merci à Vincent van Oostrom pour ce contre-exemple.

Considérons finalement la notion de symétrie de la section 2.2.4, et supposons que la fonction de converser (i) « renverse » le produit :

$$\forall x, y \in X,\ \overline{x \cdot y} = \overline{y} \cdot \overline{x}\ . \tag{2.37}$$

Cet axiome entraîne naturellement le lemme ci-dessous qui, couplé aux propositions 2.65 et 2.67(3), mène au corollaire qui le suit, et qui nous permettra de prouver très facilement que certaines génératrices caractérisent des relations d'équivalence.

Lemme 2.71. *s respecte le monoïde à gauche si et seulement si $\overline{s}$ le respecte à droite.*

Corollaire 2.72. *Si s respecte le monoïde à gauche, alors la $\overleftrightarrow{s}$-similarité est réflexive, transitive et symétrique.*

2.3 Formulation en termes de progressions

Le point de départ de la théorie de la section précédente est en fait la théorie des techniques modulo pour la bisimulation forte développée par Davide Sangiorgi [San98]. L'idée principale dans cet article est de considérer une relation ($\rightsquigarrow$) entre relations binaires (appelée *progression*), telle qu'une relation R (entre états) soit une bisimulation forte si et seulement si $R \rightsquigarrow R$. Une technique modulo s'exprime alors par une fonction f des relations : R est une « bisimulation modulo f » lorsque $R \rightsquigarrow f(R)$, la correction de la technique devant alors assurer que R est contenue dans une bisimulation forte. Une notion de fonction « sûre » (« respectful » dans [San98] et « safe » dans [SW01]) est ensuite définie : les fonctions sûres sont correctes, et satisfont de bonnes propriétés de clôture : on peut les assembler de façon modulaire.

On montre dans cette section comment généraliser la théorie de [San98], en abstrayant sur la nature des relations, et en définissant une notion générique de progression. On obtient ainsi des résultats équivalents à ceux de la section précédente, mais en caractérisant les simulations à l'aide d'une progression plutôt que par une génératrice. Il apparaît notamment que la notion de fonction compatible correspond à une notion de *monotonie*, correspondant plus ou moins à celle de sûreté [San98]. Ceci nous permet en premier lieu de situer nos travaux par rapport à [San98], mais cela nous permet surtout de caractériser les différentes notions de la section précédente de manière plus intuitive : si les fonctions (et leurs notations) sont mieux adaptées que les progressions pour énoncer la théorie de la section précédente, l'idée de progression est plus intuitive en pratique, notamment lorsque l'on considère des diagrammes de commutation, comme on le fera au prochain chapitre.

Comme précédemment, on travaille toujours dans un treillis complet monoïdal, non nécessairement continu : $\langle X, \sqsubseteq, \bigvee, \cdot, 1\rangle$. On commence par généraliser la notion de progression, puis l'on montre comment l'utiliser pour

reformuler les résultats principaux de la section précédente. On montre ensuite qu'il existe une bijection entre fonctions croissantes (génératrices) et progressions, qui préserve les notions co-inductives qui leurs sont associées : simulations, similarité, et techniques modulo.

Définition 2.73 (Progression)**.** Une *progression* est une relation binaire $\rightsquigarrow$ entre les éléments de X, telle que :

$$\forall x, x', y', y \in X,\ x \sqsubseteq x', x' \rightsquigarrow y',\ \text{et } y' \sqsubseteq y \text{ impliquent } x \rightsquigarrow y\ , \tag{2.38}$$

$$\forall Y \subseteq X,\ \forall z \in X,\ \text{si } \forall y \in Y,\ y \rightsquigarrow z,\ \text{alors } \bigvee Y \rightsquigarrow z\ . \tag{2.39}$$

On fixe dans la suite une progression $\rightsquigarrow$, et on écrit x *progresse vers* y lorsque x et y sont deux éléments de X tels que $x \rightsquigarrow y$.

Définition 2.74 (Simulation et similarité vis-à-vis d'une progression)**.** Un élément x de X est une $\rightsquigarrow$*-simulation* si $x \rightsquigarrow x$; la $\rightsquigarrow$*-similarité* est la borne supérieure de l'ensemble des $\rightsquigarrow$*-simulations* :

$$\boldsymbol{\nu}_{\rightsquigarrow} \triangleq \bigvee \{x \in X \mid x \rightsquigarrow x\}\ .$$

Exemple 2.75 (Progression pour la bisimilarité forte)**.** La relation suivante, entre relations binaires, est une progression :

$$R \rightsquigarrow_b S \ \text{ si } \begin{cases} \forall \alpha, p, p',\ p\ R\ q \text{ et } p \xrightarrow{\alpha} p' \text{ impliquent } \exists q', q \xrightarrow{\alpha} q' \text{ et } p'\ S\ q'\ , \\ \forall \alpha, q, q',\ p\ R\ q \text{ et } q \xrightarrow{\alpha} q' \text{ impliquent } \exists p', p \xrightarrow{\alpha} p' \text{ et } p'\ S\ q'\ . \end{cases}$$

On vérifie aisément que les $\rightsquigarrow_b$-simulations sont les b-simulations, où b est la génératrice donnée à l'exemple 2.40 : cette progression génère la bisimilarité forte.

Proposition 2.76. *La $\rightsquigarrow$-similarité est la plus grande $\rightsquigarrow$-simulation.*

Démonstration. Il suffit de montrer que $\boldsymbol{\nu}_{\rightsquigarrow}$ est une $\rightsquigarrow$-simulation. Pour tout x tel que $x \rightsquigarrow x$, on a : $x \sqsubseteq \bigvee \{y \mid y \rightsquigarrow y\} \triangleq \boldsymbol{\nu}_{\rightsquigarrow}$ par (2.4), puis $x \rightsquigarrow \boldsymbol{\nu}_{\rightsquigarrow}$ avec (2.38). On conclut avec (2.39) : $\bigvee \{x \mid x \rightsquigarrow x\} \rightsquigarrow \boldsymbol{\nu}_{\rightsquigarrow}$, c'est-à-dire, $\boldsymbol{\nu}_{\rightsquigarrow} \rightsquigarrow \boldsymbol{\nu}_{\rightsquigarrow}$. ■

Définition 2.77 (Fonctions correctes, monotones)**.** Une fonction croissante $f \in X^{\langle X \rangle}$, est dite :

- *correcte pour* $\rightsquigarrow$ si $\forall x \in X,\ x \rightsquigarrow f(x)$ implique $x \sqsubseteq \boldsymbol{\nu}_{\rightsquigarrow}$;
- *monotone pour* $\rightsquigarrow$ si $\forall x, y \in X,\ x \rightsquigarrow y$ implique $f(x) \rightsquigarrow f(y)$.

Comme dans la section précédente, la propriété de correction indique seulement ce qui est attendu d'une technique modulo valide ; au contraire, la propriété de monotonie est une notion indépendante de la $\rightsquigarrow$-similarité.

On vérifie aisément que la famille des fonctions monotones possède les mêmes bonnes propriétés de clôture que celle des fonctions compatibles (proposition 2.50) ; on obtient leur correction par itération :

Lemme 2.78. *Soit f une fonction monotone pour $\rightsquigarrow$.*
Pour tout x tel que $x \rightsquigarrow f(x)$, $f^\omega(x)$ est une $\rightsquigarrow$-simulation

Démonstration. On prouve $\forall n \in \mathbb{N},\ f^{(n)}(x) \rightsquigarrow f^{(n+1)}(x)$ par récurrence sur n. Avec (2.38), on en déduit $\forall n \in \mathbb{N},\ f^{(n)}(x) \rightsquigarrow f^\omega(x)$, d'où $f^\omega(x) \rightsquigarrow f^\omega(x)$, par (2.39). ■

Théorème 2.79. *Les fonctions monotones pour $\rightsquigarrow$ sont correctes pour $\rightsquigarrow$.*

Remarque 2.80 (Liens avec [San98]). Cette présentation des techniques modulo provient essentiellement de [San98], où elle est donnée dans le cas de la bisimulation forte. La relation de progression que l'on y trouve, ainsi que celle définie dans [SW01] pour la bisimulation faible, sont notamment des instances de notre notion de progression.

La notion correspondant à la monotonie (la sûreté) y est cependant légèrement différente :

> « *Une fonction f est* sûre *si $R \subseteq S$ et $R \rightsquigarrow S$ impliquent $f(R) \subseteq f(S)$ et $f(R) \rightsquigarrow f(S)$* ».

Cela permet d'obtenir la sûreté de certaines fonctions de fermeture par contexte (cf. [San98, remarque 2.6]), mais la preuve de correction des fonctions sûres est légèrement plus compliquée (il faut notamment itérer la fonction différemment). A l'inverse, nous conservons la notion simple de monotonie, mais nous choisissons « mieux » la relation de progression, en demandant explicitement que $R \rightsquigarrow S$ implique $R \subseteq S$ (cf. définition 3.10 et remarque 3.16 au chapitre suivant). On conserve ainsi une preuve élémentaire pour le théorème de correction, sans perdre la fiabilité des fonctions de fermeture par contexte (entre autres).

Le fait d'avoir une preuve simple de correction n'est pas seulement esthétique : c'est cette simplification qui nous a mené au théorème 2.52, qui permet de composer une fonction correcte et une fonction compatible. Cette généralisation du théorème de correction, qui n'a pas de correspondant dans [San98], peut aussi être obtenue en utilisant les progressions :

Théorème 2.81 (Composition d'une fonction monotone avec une fonction correcte)**.** *Soient f une fonction monotone pour $\rightsquigarrow$ et g une fonction correcte pour $\rightsquigarrow$. Si f est compatible avec g, alors $g \circ f$ est correcte pour $\rightsquigarrow$.*

Démonstration. On définit $\rightsquigarrow^g \triangleq \{\langle x, y\rangle \mid x \rightsquigarrow g(y)\}$, qui est une progression car g est croissante. La correction de g donne $\boldsymbol{\nu}_{\rightsquigarrow^g} \sqsubseteq \boldsymbol{\nu}_{\rightsquigarrow}$, on prouve ensuite que f est monotone pour $\rightsquigarrow^g$: supposons $x \rightsquigarrow^g y$, on a :

$$
\begin{aligned}
x &\rightsquigarrow g(y) && \text{(par définition)}\\
f(x) &\rightsquigarrow f(g(y)) && \text{(monotonie de } f\text{)}\\
f(x) &\rightsquigarrow g(f(y)) && \text{((2.38) et compatibilité de } f \text{ avec } g\text{)}\\
f(x) &\rightsquigarrow^g f(y)\ . && \text{(par définition)}
\end{aligned}
$$

Par le théorème 2.79, f est donc correcte pour $\rightsquigarrow^g$, ce qui implique que $g \circ f$ est correcte pour $\rightsquigarrow$. ■

Bien que les notations s'y prêtent moins, le reste de la théorie de la section précédente pourrait aussi être développé en termes de progressions : correction via, conjonctions, symétrie, respect du monoïde interne... Cela ne sera pas nécessaire : on va montrer qu'il y a équivalence entre les deux présentations.

2.3.1 Des fonctions aux progressions

On considère toujours une génératrice s, et on montre qu'on peut lui associer une progression ayant les mêmes propriétés co-inductives (simulations et techniques modulo). Bien que ce ne soit pas l'encodage dans cette direction qui nous intéresse en pratique, il permet de mettre en relief les liens entre les deux présentations et nous fournira de façon indirecte l'équivalence entre fonctions monotones et compatibles pour l'autre encodage.

Définition 2.82 (Progression associée à une génératrice)**.** On appelle *progression associée à* s la relation suivante :

$$\rightsquigarrow_s \triangleq \{\langle x, y\rangle \mid x \sqsubseteq s(y)\} \quad .$$

Lemme 2.83. *$\rightsquigarrow_s$ est une progression.*

Démonstration.

(2.38) Supposons $x \sqsubseteq x' \rightsquigarrow_s y' \sqsubseteq y$. Par la croissance de s, $s(y') \sqsubseteq s(y)$; d'où $x \sqsubseteq x' \sqsubseteq s(y') \sqsubseteq s(y)$, c'est-à-dire, $x \rightsquigarrow_s y$.

(2.39) Soient $Y \subseteq X$ et $z \in X$ tels que $\forall y \in Y,\ y \rightsquigarrow_s z$. Par définition, on a $\forall y \in Y,\ y \sqsubseteq s(z)$, d'où $\bigvee Y \sqsubseteq s(z)$, i.e., $\bigvee Y \rightsquigarrow_s z$. ■

Notons que la progression $\rightsquigarrow_b$ de l'example 2.75 n'est autre que la progression associée à la génératrice b, de l'example 2.40. La définition de la progression associée correspond exactement à la façon dont on utilise s dans la section précédente : ce sont principalement ses post-points fixes qui nous intéressent. On en déduit immédiatement les deux premiers points de la proposition suivante. Seul le troisième point peut sembler étonnant, puisqu'il relie deux propriétés relativement différentes : monotonie et compatibilité.

Proposition 2.84. *1. Les $\rightsquigarrow_s$-simulations sont les s-simulations.*

2. La $\rightsquigarrow_s$-similarité est la s-similarité.

3. Les fonctions monotones pour $\rightsquigarrow_s$ sont celles compatibles avec s.

4. Les fonctions correctes pour $\rightsquigarrow_s$ sont celles correctes pour s.

Démonstration. 1. Immédiat par définition de $\rightsquigarrow_s$.

2. Immédiat par le point précédent.

3. Soit f une fonction monotone pour $\rightsquigarrow_s$. Soit $x \in X$, on a :

$$\begin{aligned} s(x) &\sqsubseteq s(x) && \text{(réflexivité)} \\ s(x) &\rightsquigarrow_s x && \text{(par définition)} \\ f(s(x)) &\rightsquigarrow_s f(x) && \text{(monotonie de } f\text{)} \\ f(s(x)) &\sqsubseteq s(f(x)) \ . && \text{(par définition)} \end{aligned}$$

On a donc bien $f \circ s \sqsubseteq s \circ f$. Inversement, soit f une fonction compatible avec s, et supposons $x \rightsquigarrow_s y$:

$$\begin{aligned} x &\sqsubseteq s(y) && \text{(par définition)} \\ f(x) &\sqsubseteq f(s(y)) && \text{(croissance de } f\text{)} \\ f(x) &\sqsubseteq s(f(y)) && \text{(compatibilité de } f\text{)} \\ f(x) &\rightsquigarrow_s f(y) \ . && \text{(par définition)} \end{aligned}$$

4. On a $x \rightsquigarrow_s f(x)$ si et seulement si $x \in X_{s \circ f}$, et on conclut avec (2). ■

2.3.2 Des progressions aux fonctions

On définit maintenant l'encodage des progressions dans les fonctions. C'est cet encodage qui va nous intéresser dans la suite : cela nous permettra au chapitre suivant de raisonner en termes de progressions, tout en s'exprimant dans les termes plus adaptés de la théorie précédente.

Définition 2.85 (Génératrice associée à une progression)**.** Pour tout élément x de X, on note $\lfloor x \rfloor$ l'ensemble des éléments qui progressent vers x :

$$\lfloor x \rfloor \triangleq \{y \in X \mid y \rightsquigarrow x\}$$

On définit alors la *génératrice associée à* $\rightsquigarrow$ comme suit :

$$\begin{aligned} s_{\rightsquigarrow} : X &\to X \\ x &\mapsto \bigvee \lfloor x \rfloor \ . \end{aligned}$$

Lemme 2.86. *$s_{\rightsquigarrow}$ est croissante ;*

Démonstration. Si $x \sqsubseteq y$ alors (2.38) donne $\lfloor x \rfloor \subseteq \lfloor y \rfloor$, d'où $\bigvee \lfloor x \rfloor \sqsubseteq \bigvee \lfloor y \rfloor$ par le lemme 2.60. ■

Bien entendu, la génératrice b de l'example 2.40 est la génératrice associée à la progression $\rightsquigarrow_b$, de l'example 2.75.

Lemme 2.87. *Pour tout élément $x \in X$, on a $\bigvee \lfloor x \rfloor \rightsquigarrow x$.*

La proposition suivante montre que l'on a en fait défini une bijection entre les progressions et les fonctions croissantes. Le second point de cette proposition nous permettra de nous appuyer sur la proposition 2.84 dans la preuve du théorème 2.89 ci dessous.

Proposition 2.88. *On a* $s = s_{\rightsquigarrow_s}$ *et* $\rightsquigarrow = \rightsquigarrow_{s_{\rightsquigarrow}}$.

Démonstration.
– Pour tout x, $s_{\rightsquigarrow_s}(x) = \bigvee \{y \in X \mid y \sqsubseteq s(x)\} = s(x)$.
– Si $x \rightsquigarrow y$ alors $x \in \lfloor y \rfloor$, d'où $x \sqsubseteq \bigvee \lfloor y \rfloor$, i.e., $x \rightsquigarrow_{s_{\rightsquigarrow}} y$. Réciproquement, supposons $x \rightsquigarrow_{s_{\rightsquigarrow}} y$, i.e., $x \sqsubseteq \bigvee \lfloor y \rfloor$. Comme $\bigvee \lfloor y \rfloor \rightsquigarrow y$ par le lemme 2.87, on obtient $x \rightsquigarrow y$ avec (2.38). ■

Nous pouvons maintenant caractériser l'ensemble des notions de la section précédente, lorsque la génératrice s que l'on considère se trouve être la fonction associée à une progression. Nous énonçons le théorème sans utiliser le vocabulaire introduit pour les progressions, de façon à ce qu'il puisse se lire comme une série de définitions :

Théorème 2.89. *1. Pour tous* $x, y \in X$, $x \sqsubseteq s_{\rightsquigarrow}(y) \Leftrightarrow x \rightsquigarrow y$.

2. Un élément $x \in X$ *est une* $s_{\rightsquigarrow}$*-simulation si et seulement si* $x \rightsquigarrow x$.

3. Une fonction f *est compatible avec* $s_{\rightsquigarrow}$ *si et seulement si*

$$\forall x, y \in X,\ x \rightsquigarrow y \Rightarrow f(x) \rightsquigarrow f(y) \ .$$

4. Une fonction f *est correcte pour* $s_{\rightsquigarrow}$ *via* f' *si et seulement si* f' *est extensive et :*

$$\forall x \in X,\ x \rightsquigarrow f(x) \Rightarrow f'(x) \rightsquigarrow f'(x) \ .$$

5. $s_{\rightsquigarrow}$ *respecte le monoïde si et seulement si on a* $1 \rightsquigarrow 1$ *et*

$$\forall x, x', y, y' \in X,\ x \rightsquigarrow x',\ y \rightsquigarrow y' \Rightarrow x \cdot y \rightsquigarrow x' \cdot y' \ .$$

6. $s_{\rightsquigarrow}$ *respecte le monoïde à gauche si et seulement si on a* $1 \rightsquigarrow 1$ *et*

$$\forall x, x', y \in X,\ x \rightsquigarrow x',\ y \rightsquigarrow y \Rightarrow x \cdot y \rightsquigarrow x' \cdot y \ .$$

Démonstration.
1. On a $x \rightsquigarrow y \Leftrightarrow x \rightsquigarrow_{s_{\rightsquigarrow}} y \Leftrightarrow x \sqsubseteq s_{\rightsquigarrow}(y)$, par la proposition 2.88.
2. Immédiat avec le point précédent.
3. On utilise les propositions 2.84 et 2.88 : les fonctions compatibles avec $s_{\rightsquigarrow}$ sont celles monotones pour $\rightsquigarrow_{s_{\rightsquigarrow}}$ qui sont celles monotones pour $\rightsquigarrow$.
4. Par le premier point, $x \sqsubseteq s_{\rightsquigarrow}(f(x)) \Leftrightarrow x \rightsquigarrow f(x)$; et par le second $f'(x)$ est une $s_{\rightsquigarrow}$-simulation si et seulement si c'est une $\rightsquigarrow$-simulation.
5. Le second point indique en particulier que 1 est une $s_{\rightsquigarrow}$-simulation si et seulement si $1 \rightsquigarrow 1$. Ensuite, pour l'implication directe, soient $x \rightsquigarrow x'$ et $y \rightsquigarrow y'$, et montrons $x \cdot y \rightsquigarrow x' \cdot y'$. Par le premier point, on a $x \sqsubseteq s_{\rightsquigarrow}(x')$ et $y \sqsubseteq s_{\rightsquigarrow}(y')$, d'où $x \cdot y \sqsubseteq s_{\rightsquigarrow}(x') \cdot s_{\rightsquigarrow}(y') \sqsubseteq s_{\rightsquigarrow}(x' \cdot y')$,

par respect de l'ordre partiel par le produit, puis respect du produit par $s_{\rightsquigarrow}$. Le premier point nous donne finalement bien $x \cdot y \rightsquigarrow x' \cdot y'$. Pour la réciproque, soient $x, y \in X$, on a $s_{\rightsquigarrow}(x) \rightsquigarrow x$ et $s_{\rightsquigarrow}(y) \rightsquigarrow y$, d'où, par hypothèse, $s_{\rightsquigarrow}(x) \cdot s_{\rightsquigarrow}(y) \rightsquigarrow x \cdot y$, c'est-à-dire $s_{\rightsquigarrow}(x) \cdot s_{\rightsquigarrow}(y) \sqsubseteq s_{\rightsquigarrow}(x \cdot y)$ par le premier point.

6. Similaire à la preuve du point précédent. ■

2.4 Pour aller plus loin

La section 5.1 complétera ce chapitre, en étudiant les techniques modulo pour la compatibilité ; nous concluons maintenant par quelques remarques.

Treillis complets monoïdaux « exotiques »

Une bonne partie de la théorie précédente est établie dans un treillis complet monoïdal quelconque. C'est a priori un treillis de relations binaires qui sera utilisé dans le cas de la bisimulation, et nous considérerons au chapitre 5 le treillis complet monoïdal des fonctions croissantes.

La question se pose naturellement de savoir s'il existe d'autre sortes de treillis complets monoïdaux, pour lesquels la théorie précédente pourrait être utile. Le fait de ne demander ni les complémentaires ni la distributivité nous laisse un peu de marge de manœuvre quant au choix du treillis : tout treillis distributif est isomorphe à un treillis d'ensembles (i.e., à un sous-treillis non nécessairement complet de $\mathscr{P}(\mathcal{Q})$, pour un ensemble quelconque $\mathcal{Q}$) ; toute algèbre de Boole complète et complètement distributive est isomorphe au treillis des parties d'un ensemble [DP90, théorèmes 10.21 et 10.24]. Il reste cependant difficile de se rendre compte des contraintes qu'imposerait la structure de monoïde sur un treillis qui ne serait pas un treillis de parties.

Contraintes

On souhaite parfois imposer des contraintes sur les objets mis en jeu : pour la « bisimilarité barbelée » [MS92a], les relations ne doivent mettre en correspondance que des processus possédant les mêmes « barbes » ; pour les « bisimulations typées » [SW01, HR04] les processus mis en relation doivent obéir à la même politique de typage. De telles contraintes peuvent être représentées par le plus grand objet les satisfaisant, noté c dans la suite.

Deux approches sont alors possibles : lorsque le monoïde préserve cette contrainte, le treillis initial (X), restreint aux relations majorées par c, est un treillis complet monoïdal : $X^c \triangleq X \cap \{c\}^l$, qui peut être utilisé à condition que toutes les fonctions manipulées respectent la contrainte.

L'autre solution consiste à contraindre les génératrices considérées à l'aide de la fonction constante $\widehat{c}$: pour toute génératrice s, on peut travailler avec la génératrice $(s \wedge \widehat{c})$, qui nous force elle aussi à respecter la contrainte (on

a en fait $X^c = X_{\widehat{c}}$). L'avantage de cette seconde solution est qu'elle reste valide même lorsque le monoïde ne respecte pas la contrainte.

Étages

La remarque précédente fait apparaître un lien entre le choix du treillis et celui de la génératrice, dans le cas d'une contrainte « constante ». Cette idée peut en fait être généralisée : lorsque l'on souhaite considérer le plus grand point fixe commun à deux génératrices s et t, on peut considérer la génératrice composée $s \wedge t$, comme on l'a fait la section 2.2.3, mais on peut aussi remarquer que X_s est un treillis complet, dans lequel il suffit d'étudier la t-similarité (ou l'inverse). Plus formellement, on a les égalités suivantes entre ensembles :

$$X_{s\wedge t} = (X_s)_t = (X_t)_s = X_s \cap X_t \ .$$

On pourrait ainsi imaginer des constructions étagées, où l'on raffinerait le treillis initial au fur et à mesure.

L'avantage d'une telle présentation serait d'obtenir des résultats comme la proposition suivante, où l'hypothèse de compatibilité sur la deuxième composante (ici, g avec t) est allégée : elle ne doit être vérifiée que sur les s-simulations :

Proposition 2.90. *Soient $s, t, f, g \in X^{\langle X\rangle}$ quatre fonctions croissantes telles que :*

1. *f est compatible avec s, t et g,*
2. *$g(X_s) \subseteq X_s$, et*
3. *g restreinte à X_s est compatible avec t.*

Alors pour tout $x \in X$ tel que $x \sqsubseteq s(f(x))$ et $x \sqsubseteq t(g(x))$, on a $x \sqsubseteq \boldsymbol{\nu}\,(s \wedge t)$.

Démonstration. On a tout d'abord $f^\omega(x) \in X_s$ par compatibilité de f avec s. Ensuite, on a $f^\omega(x) \sqsubseteq f^\omega(t(g(x))) \sqsubseteq t(f^\omega(g(x))) \sqsubseteq t(g(f^\omega(x)))$, par compatibilité de f (et donc de f^ω) avec t puis g. On en déduit $g^\omega(f^\omega(x)) \in (X_s)_t$ par compatibilité de g avec t sur X_s, ce qui permet de conclure. ■

Nous décrirons à la remarque 3.25 comment une telle proposition pourrait être utilisée dans le cas de la bisimulation faible.

Chapitre 3

Bisimulations étiquetées

Nous avons développé au chapitre précédent une théorie des techniques permettant de raffiner la méthode de preuve par co-induction. Dans ce chapitre, nous nous concentrons sur le cas particulier des diverses notions de bisimulation que l'on peut définir sur des systèmes de transitions étiquetées.

Nous commençons par préciser la structure abstraite dans laquelle nous travaillons (les algèbres relationnelles), puis nous montrons comment y retrouver les diverses notions usuelles de simulation et similarité. En appliquant les résultats précédents, on retrouve l'ensemble des techniques modulo standard pour ces différents préordres, que l'on symétrise ensuite, afin de retomber sur les équivalences comportementales standard.

Ce chapitre nous permet de poser les bases du chapitre suivant, et de faire un inventaire méthodique et uniforme de l'ensemble des préordres ou équivalences comportementaux ainsi que des techniques modulo qui leurs sont associés. Outre l'idée de se placer au niveau d'abstraction fourni par les algèbres relationnelles, la contribution la plus novatrice de ce chapitre est la notion de *préordre contrôlé*, qui nous permettra de généraliser la technique « modulo expansion ».

3.1 Algèbres relationnelles et systèmes de transitions étiquetées

3.1.1 Algèbres relationnelles

Afin de mieux définir le domaine d'application de notre théorie des techniques modulo, nous sommes partis au chapitre précédent d'une structure de treillis complet, que nous avons ensuite enrichie avec une opération de symétrie, puis une structure de monoïde.

La notion de bisimulation, traditionnellement définie dans le cadre des relations binaires, utilise pleinement ces trois ingrédients, qui seront donc au centre des développements à suivre.

Définition 3.1 (Algèbre relationelle)**.** Une *algèbre relationnelle* est un sextuplet $\langle X, \sqsubseteq, \bigvee, \cdot, 1, \overline{\cdot} \rangle$, où $\langle X, \sqsubseteq, \bigvee, \cdot, 1 \rangle$ est un treillis complet monoïdal continu, et $\overline{\cdot} : X \to X$ est une fonction de *conversion* satisfaisant les quatre propriétés suivantes :

$$\forall x \in X, \quad \overline{\overline{x}} = x \tag{3.1}$$

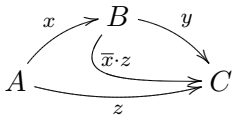
$$\forall x, y \in X, \quad x \sqsubseteq y \Rightarrow \overline{x} \sqsubseteq \overline{y} \tag{3.2}$$

$$\forall x, y \in X, \quad \overline{x \cdot y} = \overline{y} \cdot \overline{x} \tag{3.3}$$

$$\forall x, y, z \in X, \quad (x \cdot y) \wedge z \sqsubseteq x \cdot (y \wedge (\overline{x} \cdot z)) \tag{3.4}$$

Les trois premières propriétés indiquent que la fonction de conversion est une involution croissante qui renverse le produit, elles correspondent aux hypothèses que l'on faisait au chapitre 2, de telle sorte que les résultats précédents s'appliquent à toute algèbre relationnelle.

Le dernier axiome (3.4), nommé « loi de modularité » dans [FS90], permet de relier les trois « ingrédients » de cette structure (treillis, monoïde, conversion). Il ne sera principalement utilisé qu'au prochain chapitre, et nous le mentionnerons explicitement chaque fois que nous l'utiliserons. Un moyen mnémotechnique de se rappeler cet axiome consiste à en « typer » les éléments :



Notons que cette dernière loi est la seule qui fasse intervenir les bornes inférieures, que nous avons choisi de ne pas intégrer à la structure (elles sont seulement dérivées, définition 2.5). Cela n'est pas problématique dans notre cadre, puisque nous ne parlerons pas de morphismes d'algèbre relationelle ; si tel devait être le cas, il nous faudrait prendre garde à ce que ces morphismes préservent la loi de modularité. Nous laissons pour de futures recherches la question d'obtenir cette loi à partir d'axiomes ne mentionant que les bornes supérieures (finies).

Rappelons que nous donnons la priorité au produit sur les bornes inférieures et supérieures, la loi de modularité sera par exemple écrite sous la forme $x \cdot y \wedge z \sqsubseteq x \cdot (y \wedge \overline{x} \cdot z)$. On a les propriétés suivantes :

Lemme 3.2. *Dans toute algèbre relationnelle X, on a :*

$$\overline{1} = 1 \tag{3.5}$$

$$\forall x, y \in X, \quad x \sqsubseteq y \Leftrightarrow \overline{x} \sqsubseteq \overline{y} \tag{3.6}$$

$$\forall Y \subseteq X, \quad \overline{\bigvee Y} = \bigvee \overline{Y} \text{ et } \overline{\bigwedge Y} = \bigwedge \overline{Y} \tag{3.7}$$

$$\forall x, y, z \in X, \quad x \cdot y \wedge z \sqsubseteq (x \wedge z \cdot \overline{y}) \cdot y \tag{3.8}$$

Démonstration. Pour (3.5), on a $\overline{1} = 1 \cdot \overline{1} = \overline{\overline{1}} \cdot \overline{1} = \overline{\overline{1} \cdot \overline{1}} = \overline{\overline{1}} = 1$ (par complétion du système de réécriture naturel pour $\bar{\cdot}$, $\cdot$ et 1). Les propriétés (3.6) et (3.7) ont déjà été prouvées (lemme 2.60, elles découlent de (3.1) et (3.2)) ; (3.8) s'obtient en « inversant » (3.4) à l'aide de (3.3), (3.6) et (3.7). ■

Les algèbres relationnelles étant destinées à représenter de façon abstraite les relations binaires, il nous faut vérifier que ces dernières en forment bien un modèle, et ce quel que soit la nature des « individus » mis en relation. Fixons pour cela un ensemble $\mathcal{P}$ quelconque.

Définition 3.3 (Relations binaires)**.** On note $\mathcal{R} \triangleq \mathscr{P}(\mathcal{P} \times \mathcal{P})$ l'ensemble des *relations binaires* sur $\mathcal{P}$. Les éléments de $\mathcal{R}$ seront dénotés par R, S, et l'appartenance d'un couple $\langle p, q\rangle$ à une relation binaire R sera notée $p\ R\ q$.

Soient $R, S \in \mathcal{R}$ deux relations binaires, et $\mathcal{S} \subseteq \mathcal{R}$ une famille de relations binaires. On définit comme suit l'*inclusion* de S dans R, l'*union* et l'*intersection* de $\mathcal{S}$, la *composition* de R et S, la relation *identité*, et la *converse* de R :

$$
\begin{aligned}
S \subseteq R \text{ si } & \forall p, q \in \mathcal{P},\ p\ S\ q \Rightarrow p\ R\ q \\
\bigcup_{R \in \mathcal{S}} R &\triangleq \{\langle p, q\rangle \mid \exists R \in \mathcal{S}, p\ R\ q\} \\
\bigcap_{R \in \mathcal{S}} R &\triangleq \{\langle p, q\rangle \mid \forall R \in \mathcal{S}, p\ R\ q\} \\
R \cdot S &\triangleq \{\langle p, q\rangle \mid \exists r \in \mathcal{P}, p\ R\ r \text{ et } r\ S\ q\} \\
I &\triangleq \{\langle p, p\rangle \mid p \in \mathcal{P}\} \\
\overline{R} &\triangleq \{\langle q, p\rangle \mid p\ R\ q\}
\end{aligned}
$$

On dira que R *contient* une relation S (ou encore, que S *est contenue dans* R), lorsque $S \subseteq R$.

L'inclusion, l'union et l'intersection correspondent aux opérations ensemblistes dans le treillis des parties sur $\mathcal{P} \times \mathcal{P}$, d'où leurs notations ; les autres opérations étendent ce treillis complet en une algèbre relationnelle, dite *propre*, et on retrouve les notions usuelles de réflexivité, transitivité, symétrie, ainsi que les fermetures associées :

Proposition 3.4. $\langle \mathcal{R}, \subseteq, \bigcup, \cdot, I, \bar{\cdot}\rangle$ *est une algèbre relationnelle appelée* algèbre relationnelle propre de $\mathcal{P}$*, dans laquelle toute relation binaire* $R \in \mathcal{R}$ *satisfait :*

- R *est réflexive si et seulement si* $\forall p \in \mathcal{P},\ p\ R\ p$ *;*
- R *est transitive si et seulement si* $\forall p, q, r \in \mathcal{P},\ p\ R\ q$ *et* $q\ R\ r$ *impliquent* $p\ R\ r$ *;*
- R *est symétrique si et seulement si* $\forall p, q \in \mathcal{P},\ p\ R\ q$ *si et seulement si* $q\ R\ p$ *;*

– $R^{=} = \{\langle p, q\rangle \mid p = q \ \textit{ou}\ p\ R\ q\}$;
– $R^{+} = \{\langle p_0, p_n\rangle \mid \exists n > 0,\ \exists p_1, \ldots, p_{n-1} \in \mathcal{P},\ \forall i < n, p_i\ R\ p_{i+1}\}$;
– $R^{\star} = \{\langle p_0, p_n\rangle \mid \exists n \in \mathbb{N},\ \exists p_1, \ldots, p_{n-1} \in \mathcal{P},\ \forall i < n, p_i\ R\ p_{i+1}\}$.

Démonstration. Nous montrons seulement la loi de modularité (3.4), qui est la moins standard : soient R, S, T trois relations binaires, et supposons que $p\ (R \cdot S) \cap T\ q$: on a $p\ T\ q$ et il existe r tel que $p\ R\ r\ S\ q$. Il nous faut montrer $p\ R \cdot (S \cap (\overline{R} \cdot T))\ q$; ayant $p\ R\ r$ et $r\ S\ q$, il suffit de montrer $r\ \overline{R} \cdot T\ q$, c'est le cas, via p : on a $r\ \overline{R}\ p\ T\ q$. ■

Remarque 3.5 (Calcul des relations, allégories). La notion d'algèbre relationnelle provient du calcul des relations, d'Alfred Tarski [Tar41, CT51, JT52, TG87] (qui indique s'appuyer sur les travaux plus anciens d'Augustus de Morgan puis Charles Peirce et Ernst Schröder, entre 1860 et 1900 ; consulter [Pra92] pour un historique détaillé et une sélection de références – ou l'article récent de Steven Givant [Giv07]). En théorie des catégories, les « allégories » [FS90, Joh02] sont aussi très proches des algèbres relationnelles.

Notre objectif est cependant différent : nous ne cherchons pas à donner un ensemble d'axiomes (sans variables d'individus) qui caractérise le « comportement » des relations binaires, mais plutôt à définir un ensemble d'axiomes minimal, qui nous permette de définir les notions qui nous intéressent, puis de prouver à un niveau d'abstraction plus élevé les résultats obtenus dans le cadre spécifique des relations binaires.

La présentation que l'on a adoptée ici s'éloigne considérablement du calcul des relations tel qu'il est présenté dans [Tar41], et se rapproche bien plus des « relation algebras » de Doornbos et al. [DBvdW97] : c'est sur cet article que nous nous appuierons au chapitre suivant, pour caractériser au niveau abstrait des algèbres relationnelles la notion de terminaison dont nous aurons besoin. La seule différence avec les algèbres relationnelles telles qu'elles sont définies dans cet article est que nous n'imposons pas un treillis (complètement) distributif : la distributivité des bornes supérieures sur les bornes inférieures est demandée dans [DBvdW97], et vice-versa. La différence avec le calcul des relations est plus importante : nous considérons des treillis *complets*, mais non distributifs, et non nécessairement « complémentés » [1] (le calcul des relations prend comme point de départ les *algèbres de Boole*, i.e., les treillis distributifs et complémentés). Les allégories ne requièrent pas non plus un treillis complémenté (l'idée étant de rester en logique intuitioniste), et ne comportent initialement que le semi-treillis correspondant aux bornes inférieures. Les bornes supérieures (infinies) sont ajoutées afin d'obtenir les allégories *localement complètes*, dans lesquelles toute allégorie peut être fidèlement représentée [FS90], mais qui sont distributives.

1. Dans le cas des relations binaires, l'opération de complémentation est celle qui à toute relation associe la relation contenant exactement les couples que ne contient pas la première ; c'est la négation dans les algèbres de Boole.

Remarque 3.6 (Représentation). Dès son premier article sur le calcul des relations [Tar41], Tarski pose la question de la « représentation » :

> *« Est-ce que tout modèle du calcul des relations est isomorphe à une famille de relations binaires close par les opérations du calcul ? ».*

La réponse est négative [Lyn50] ; l'axiomatisation peut cependant être complétée de façon à ce que la réponse devienne affirmative [Lyn56] ; mais il n'existe pas d'axiomatisation finie [Mon64]. Notons aussi que la notion d'algèbre relationnelle *propre* utilisée dans [Lyn50, JT52] pour définir le problème de la représentation est un peu plus générale que la nôtre : elle permet de prendre en compte les sous-modèles issus des relations binaires (l'ensemble des relations binaires réflexives sur un ensemble donné forme par exemple une algèbre relationnelle, qui n'est pas déclarée propre selon notre définition).

Dans cette thèse, c'est la question « opposée » qui nous préoccupe : existe-t-il des algèbres relationnelles « intéressantes », qui ne soient pas un cas particulier d'algèbre relationnelle propre ? Si le fait de se placer dans une algèbre relationnelle quelconque pour étudier les techniques modulo nous semble très bénéfique en termes de méthodologie (notations, clarté des preuves, etc...), nous aimerions aussi pouvoir démontrer l'intérêt de cette approche en obtenant des résultats qui ne s'appliquent pas que dans le cas des relations binaires.

Nous avons discuté à la section 2.4 de l'existence probable de treillis complets monoïdaux continus « exotiques ». Demander de plus une involution croissante qui renverse le produit nous semble assez contraignant : si le monoïde est commutatif, on peut certes choisir l'identité ; par exemple on vérifie aisément que les deux treillis complets monoïdaux continus donnés dans l'example 2.19, étendus par l'identité, forment des algèbres relationnelles ($\langle \overline{\mathbb{N}}, \leq, \sup, +, 0, \mathrm{id}_{\overline{\mathbb{N}}} \rangle$ et $\langle \overline{\mathbb{N}}, \leq, \sup, \times, 1, \mathrm{id}_{\overline{\mathbb{N}}} \rangle$). Mais cette hypothèse de commutativité rend vaine toute la théorie que l'on va définir : lorsque tout commute, les diverses notions de bisimilarité se réduisent trivialement à l'élément maximal du treillis complet. Nous n'avons pour l'instant pas d'intuition quant à l'impact de la loi de modularité (3.4), qu'il nous est difficile d'appréhender. C'est finalement le fait que l'on ne demande ni complémentaires, ni distributivité qui nous semble laisser le plus de marge.

Avant de définir les systèmes de transitions étiquetées dans ce cadre abstrait, nous donnons un lemme qui nous permettra de calculer la converse de certains éléments ou fonctions ; il complète le lemme 2.60 du chapitre précédent :

Lemme 3.7. *Soit X une algèbre relationnelle ; pour tout élément $x \in X$, et toutes fonctions $f, g \in X^X$, on a :*

$$\overline{x^=} = (\overline{x})^= \quad , \quad \overline{x^+} = (\overline{x})^+ \quad , \quad \overline{x^\star} = (\overline{x})^\star \quad ; \tag{3.9}$$

$$\overline{f \mathbin{\hat{\cdot}} g} = \overline{g} \mathbin{\hat{\cdot}} \overline{f} \quad , \quad \overline{f^=} = (\overline{f})^= \quad , \quad \overline{f^+} = (\overline{f})^+ \quad , \quad \overline{f^\star} = (\overline{f})^\star \quad . \tag{3.10}$$

Démonstration. Le seul point délicat consiste à montrer $\forall n \in \mathbb{N},\ \overline{x^n} = (\overline{x})^n$, ce que l'on fait par récurrence, en utilisant le lemme 2.12 pour obtenir $x \cdot x^n = x^n \cdot x$, pour tout entier n. Pour passer aux fonctions, on utilise le lemme 2.29. ■

En particulier, les itérées $f^=$, f^+, et $f^\star$ d'une fonction f sont symétriques dès que f l'est (notons que la fonction de conversion ne respecte pas le monoïde, de sorte que cette propriété de préservation de la symétrie ne peut pas être obtenue en tant que corollaire de la proposition 2.68(2), comme nous l'avions fait pour les autres opérateurs (lemme 2.64)).

3.1.2 Systèmes de transitions étiquetées

On fixe à partir de maintenant une algèbre relationnelle dont on dénote les éléments du domaine (X) par $x, y \ldots$; malgré cette notation, ces éléments peuvent (doivent) être pensés comme des relations binaires sur un ensemble donné. Les objets correspondants dans [DBvdW97] y sont nommés *specs* (pour spécifications) ; comme cela dessert l'intuition lorsque l'on passe au français, on les appellera simplement *relations* dans la suite. La notion de *relation binaire* et les lettres R, S étant utilisées pour désigner des relations concrètes, cela ne devrait pas prêter à confusion.

On fixe ensuite un ensemble ($\mathcal{L}$) d'*étiquettes* (ou encore *actions*, *labels*), dont les éléments seront notés $\alpha, \beta \ldots$ Cela nous permet de donner une définition très simple d'un système de transitions étiquetées :

Définition 3.8 (Système (abstrait) de transitions étiquetées)**.** Un *système abstrait de transitions étiquetées par* $\mathcal{L}$ (LTS, pour « Labelled Transition System ») est une collection de relations indexée par $\mathcal{L}$: $(\xrightarrow{\alpha})_{\alpha\in\mathcal{L}} \in X^{\mathcal{L}}$.

Intuitivement, $\xrightarrow{\alpha}$ représente l'ensemble des transitions étiquetées par l'action α, il n'y a pas besoin de parler de domaine ou d'états, puisque ceux-ci sont implicitement cachés dans le support de l'algèbre relationnelle.

Notons aussi que cette définition, ainsi que la plupart de celles qui suivent, dépend de l'ensemble d'étiquettes que l'on s'est fixé ($\mathcal{L}$). Comme cet ensemble ne variera qu'au chapitre 6, nous n'introduirons des notations mentionnant explicitement cette dépendance que pour ce chapitre.

Afin de pouvoir considérer les équivalences dites « faibles », on distingue parmi les éléments de $\mathcal{L}$ une action dite *silencieuse* (ou *interne*), notée τ. Les éléments de $\mathcal{L}^{\mathrm{v}} \triangleq \mathcal{L} \setminus \{\tau\}$, dits *visibles* (et parfois appelés *interactions*) seront notés $a, b \ldots$

Définition 3.9 (Transitions faibles, LTS faible)**.** Soit $(\xrightarrow{\alpha})_{\alpha\in\mathcal{L}}$ un LTS. Pour toute étiquette $\alpha \in \mathcal{L}$, on définit les relations $\xrightarrow{\widehat{\alpha}}$, $\overset{\alpha}{\Rightarrow}$ et $\overset{\widehat{\alpha}}{\Rightarrow}$ par :

$$\xrightarrow{\widehat{\alpha}} \triangleq \begin{cases} \xrightarrow{\tau}{}^{=} & \text{si } \alpha = \tau \\ \xrightarrow{a} & \text{si } \alpha = a \in \mathcal{L}^{\mathrm{v}} \end{cases} \qquad \begin{aligned} \overset{\alpha}{\Rightarrow} &\triangleq \xrightarrow{\tau}{}^{\star} \cdot \xrightarrow{\alpha} \cdot \xrightarrow{\tau}{}^{\star} \\ \overset{\widehat{\alpha}}{\Rightarrow} &\triangleq \xrightarrow{\tau}{}^{\star} \cdot \xrightarrow{\widehat{\alpha}} \cdot \xrightarrow{\tau}{}^{\star} \end{aligned}$$

On appelle *LTS faible* le LTS $(\overset{\widehat{\alpha}}{\Rightarrow})_{\alpha\in\mathcal{L}}$.

Remarquons les propriétés suivantes :

$$\overset{\widehat{\tau}}{\Rightarrow} = \overset{\tau}{\rightarrow}^{\star}\ , \qquad \overset{\tau}{\Rightarrow} = \overset{\tau}{\rightarrow}^{+}, \qquad \overset{\widehat{a}}{\Rightarrow} = \overset{a}{\Rightarrow}\ ; \tag{3.11}$$

en particulier $\overset{\widehat{\tau}}{\Rightarrow}$ est réflexive, mais pas $\overset{\tau}{\Rightarrow}$ (à moins, par exemple, que $\overset{\tau}{\rightarrow}$ ne le soit). Les relations converses des relations dénotées par des flèches seront dénotées par les flèches renversées correspondantes ; par exemple, $\overset{\alpha}{\leftarrow} = \overset{\overline{\alpha}}{\rightarrow}$.

3.2 Les différentes simulations et leurs techniques modulo

Comme expliqué en introduction, les jeux de bisimulation sont définis comme la conjonction de deux jeux de simulation, l'un de gauche à droite, et l'autre de droite à gauche. Nous exploitons cette symétrie en nous ramenant à l'étude des simulations, ce qui nous permet d'une part de factoriser et de simplifier certaines preuves, et d'autre part de définir de manière systématique les préordres se trouvant entre la bisimilarité forte et la bisimilarité faible (dont l'expansion, notamment) : il suffit de choisir deux jeux, puis de les combiner, l'un de gauche à droite et l'autre de droite à gauche.

3.2.1 Génératrices des simulations

Pour appliquer la théorie du chapitre précédent, il nous faut définir une génératrice pour chacun des objets co-inductifs que l'on souhaite étudier. On passe pour cela par la notion de progression, développée à la section 2.3, qui nous semble plus intuitive lorsque l'on considère les jeux de simulation :

Définition 3.10 (Progressions et génératrices pour les jeux de simulation)**.** Les génératrices des jeux de *simulation forte* (**s**), *simulation faible* (**w**), *pré-expansion* (**e**) et *simulation progressive* (**p**), sont les génératrices associées aux relations suivantes :

$$\begin{array}{lll}
x \rightsquigarrow_{\mathbf{s}} y & \text{si} & x \sqsubseteq y \text{ et } \forall\alpha\in\mathcal{L},\ \overset{\alpha}{\leftarrow}\cdot\, x \sqsubseteq y \cdot \overset{\alpha}{\leftarrow}\ ; \\
x \rightsquigarrow_{\mathbf{w}} y & \text{si} & x \sqsubseteq y \text{ et } \forall\alpha\in\mathcal{L},\ \overset{\alpha}{\leftarrow}\cdot\, x \sqsubseteq y \cdot \overset{\widehat{\alpha}}{\Leftarrow}\ ; \\
x \rightsquigarrow_{\mathbf{e}} y & \text{si} & x \sqsubseteq y \text{ et } \forall\alpha\in\mathcal{L},\ \overset{\alpha}{\leftarrow}\cdot\, x \sqsubseteq y \cdot \overset{\widehat{\alpha}}{\leftarrow}\ ; \\
x \rightsquigarrow_{\mathbf{p}} y & \text{si} & x \sqsubseteq y \text{ et } \forall\alpha\in\mathcal{L},\ \overset{\alpha}{\leftarrow}\cdot\, x \sqsubseteq y \cdot \overset{\alpha}{\Leftarrow}\ .
\end{array}$$

Lemme 3.11. *Les quatre relations précédemment définies sont des progressions.*

Démonstration (cas de **w***).* Si $x \sqsubseteq x'$, $y' \sqsubseteq y$ et $x' \rightsquigarrow_{\mathbf{w}} y'$, on a $x \sqsubseteq y$ par transitivité, et pour tout $\alpha \in \mathcal{L}$, comme le produit préserve l'ordre partiel (2.17), on a :

$$\xleftarrow{\alpha} \cdot x \sqsubseteq \xleftarrow{\alpha} \cdot x' \sqsubseteq y' \cdot \Leftarrow^{\widehat{\alpha}} \sqsubseteq y \cdot \Leftarrow^{\widehat{\alpha}} .$$

Ensuite, soient $Y \sqsubseteq X$, $z \in X$ tels que $\forall y \in Y,\ y \rightsquigarrow_{\mathbf{w}} z$; soit $\alpha \in \mathcal{L}$. Pour tout $y \in Y$, on a $\xleftarrow{\alpha} \cdot y \sqsubseteq z \cdot \Leftarrow^{\widehat{\alpha}}$. Par la continuité du produit (2.18), on conclut :

$$\xleftarrow{\alpha} \cdot \bigvee Y = \bigvee_{y \in Y} \xleftarrow{\alpha} \cdot y \sqsubseteq z \cdot \Leftarrow^{\widehat{\alpha}} .$$

■

Par le lemme 2.86, les génératrices associées à ces progressions sont donc croissantes, et le théorème 2.89 nous permet de travailler indifféremment avec les progressions ou les génératrices. Dans le cas d'une algèbre relationnelle propre, la première progression peut-être reformulée comme suit : $R \rightsquigarrow_{\mathbf{s}} S$ si $R \subseteq S$, et $\forall \alpha, p, p',\ p\ R\ q$ et $p \xrightarrow{\alpha} p'$ impliquent $\exists q', q \xrightarrow{\alpha} q'$ et $p'\ S\ q'$; ce qui correspond à la « moitié » de la progression de l'exemple 2.75.

Nous commentons les quatre génératrices ci-dessous.

(**s**) Cette génératrice définit des jeux de *simulation forte* (la lettre **s** vient de l'anglais « strong ») : le côté droit simule exactement le côté gauche, en répondant par la même transition que celle de l'offre. De manière plus intuitive, une relation x est une **s**-simulation si et seulement si elle satisfait le diagramme suivant :

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \downarrow{\scriptstyle\alpha} \\ x & & \cdot \end{array}$$

(où la quantification universelle sur toutes les étiquettes α est implicite).

Une définition plus standard, dans le cas des relations binaires serait :

> *« Une relation R est une simulation forte si pour tous p, p', q et α tels que $p\ R\ q$ et $p \xrightarrow{\alpha} p'$, il existe q' tel que $p'\ R\ q'$ et $q \xrightarrow{\alpha} q'$ ».*

(**w**) Cette génératrice définit des jeux de *simulation faible* (« **w**eak ») : le côté droit simule le côté gauche, modulo des transitions silencieuses, en répondant par la transition faible ($\stackrel{\widehat{\alpha}}{\Rightarrow}$) correspondant à l'offre ($\xrightarrow{\alpha}$). Une relation x est une **w**-simulation si et seulement si elle satisfait le diagramme suivant :

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ x & & \cdot \end{array}$$

Comme précédemment la définition plus standard, dans le cas des relations binaires serait :

> *« Une relation R est une simulation faible si pour tous p, p', q, α tels que $p\ R\ q$ et $p \xrightarrow{\alpha} p'$, il existe q' tel que $p'\ R\ q'$ et $q \stackrel{\widehat{\alpha}}{\Rightarrow} q'$ ».*

L'idée principale de la similarité faible est d'abstraire vis-à-vis de la partie « interne » d'un processus, représentée par les actions silencieuses. Ces transitions correspondent par exemple à des calculs, que l'on ne souhaite pas observer (à l'inverse, la similarité forte est capable de compter les actions silencieuses, ce qui lui permet de distinguer de processus ne différant que par leur « efficacité » pour les calculs internes).

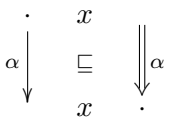
(**e**) Cette génératrice correspond à la partie « de gauche à droite » d'un jeu d'**e***xpansion* ($\succsim$) [AKH92, SM92]. L'idée est de contraindre le côté droit à être au moins aussi rapide que le côté gauche : le jeu est fort sur les actions visibles ($\xrightarrow{\widehat{a}} = \xrightarrow{a}$), mais le côté droit peut décider de ne pas bouger lors des offres silencieuses ($\xrightarrow{\widehat{\tau}} = \xrightarrow{\tau}^{=}$). Afin de faire la distinction avec l'expansion, où le jeu de droite à gauche est aussi présent, nous appellerons *pré-expansions* les **e**-simulations. Une relation x est une **e**-simulation si et seulement si elle satisfait le diagramme suivant :

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \downarrow{\scriptstyle\widehat{\alpha}} \\ x & & \cdot \end{array}$$

L'expansion a été introduite indépendamment par S. Arun Kumar et Matthew Hennessy [AKH92] et Milner et Sangiorgi [SM92]. Les premiers définissent ce préordre comportemental afin de caractériser une notion d'efficacité, et s'intéressent à ses propriétés de congruence, ainsi qu'à son axiomatisation dans la partie finie de CCS (notons aussi que ce préordre y est donné « à l'envers » : notre génératrice, **e** correspond en fait la partie « de droite à gauche » dans [AKH92]). Les seconds [SM92] s'intéressent à l'expansion en tant que technique de preuve pour la bisimilarité faible. Comme expliqué dans l'introduction, ce préordre permet d'une certaine manière de pallier au problème de « faible modulo faible » : le fait que le côté droit doive être plus rapide que le côté gauche empêche le cas pathologique où la bisimilarité faible annule une offre silencieuse (on n'a pas $a \succsim \tau.a$, ce qui élimine, entre autres, le contre-exemple (1.15) donné en introduction).

(**p**) Cette dernière génératrice correspond à la partie « de droite à gauche » d'un jeu d'*élaboration* ($\succsim$), préordre initialement étudié par Arun Kumar et Natarajan [AKN95]. C'est en quelque sorte la duale de **e** : le

côté droit doit être au moins aussi lent que le côté gauche. Une relation x est une **p**-simulation si et seulement si elle satisfait le diagramme suivant :



Lorsque l'on symétrise ce jeu (comme on le fera à la section 3.3), on obtient la *bisimulation progressive* d'Ugo Montanari et Vladimiro Sassone [MS92c]. La terminologie « progressive » provient du fait que les processus doivent progresser à chaque offre, c'est-à-dire répondre avec au moins une transition. On réservera donc le nom d'*élaboration* au préordre ($\gtrapprox$) [AKN95], que l'on définira à la section 3.3, et on appellera *simulations progressives* les **p**-simulations (ce qui justifie le nom donné à leur génératrice).

Les avantages de cette notion de simulation sont ses bonnes propriétés en termes de techniques modulo et de congruence, qui seront respectivement étudiées aux chapitres 4 et 5 ; ses propriétés élémentaires sont cependant données dans ce chapitre, par souci d'uniformité.

Afin de comparer ces différentes notions de simulation, il suffit de comparer leurs génératrices :

Proposition 3.12. *On a :*

$$\begin{array}{ccccccc} \mathbf{s} & \sqsubseteq & \mathbf{e} & \sqsubseteq & \mathbf{w}\ , & \qquad & \mathbf{s} \sqsubseteq \mathbf{p} \sqsubseteq \mathbf{w}\ , \\ X_{\mathbf{s}} & \subseteq & X_{\mathbf{e}} & \subseteq & X_{\mathbf{w}}\ , & & X_{\mathbf{s}} \subseteq X_{\mathbf{p}} \subseteq X_{\mathbf{w}}\ , \\ \boldsymbol{\nu}\mathbf{s} & \sqsubseteq & \boldsymbol{\nu}\mathbf{e} & \sqsubseteq & \boldsymbol{\nu}\mathbf{w}\ , & & \boldsymbol{\nu}\mathbf{s} \sqsubseteq \boldsymbol{\nu}\mathbf{p} \sqsubseteq \boldsymbol{\nu}\mathbf{w}\ . \end{array}$$

Démonstration. On a :

$$\begin{array}{rccccccccccc} \forall \alpha \in \mathcal{L}, & \xrightarrow{\alpha} & \sqsubseteq & \xrightarrow{\widehat{\alpha}} & \sqsubseteq & \overset{\widehat{\alpha}}{\Rightarrow} & \text{et} & \xrightarrow{\alpha} & \sqsubseteq & \overset{\alpha}{\Rightarrow} & \sqsubseteq & \overset{\widehat{\alpha}}{\Rightarrow}\ , \\ \text{ce qui entraîne} & \rightsquigarrow_{\mathbf{s}} & \subseteq & \rightsquigarrow_{\mathbf{e}} & \subseteq & \rightsquigarrow_{\mathbf{w}} & \text{et} & \rightsquigarrow_{\mathbf{s}} & \subseteq & \rightsquigarrow_{\mathbf{p}} & \subseteq & \rightsquigarrow_{\mathbf{w}}\ , \\ \text{puis} & \mathbf{s} & \sqsubseteq & \mathbf{e} & \sqsubseteq & \mathbf{w} & \text{et} & \mathbf{s} & \sqsubseteq & \mathbf{p} & \sqsubseteq & \mathbf{w}\ . \end{array}$$

On conclut par la proposition 2.43. ■

Nous ne nous étendons pas plus sur ces comparaisons, qui seront plus instructives dans la section suivante, où l'on aura défini les préordres et équivalences standard (on n'en a ici que les « moitiés ») ; notons simplement que **p** et **e** sont a priori incomparables.

Avant de passer à l'étude des techniques modulo pour les simulations, nous établissons une caractérisation alternative des simulations faibles et progressives. Cette caractérisation mène habituellement à la transitivité de la similarité faible ; elle nous sera utile pour le lemme 3.15. Sa preuve est élémentaire, et consiste à assembler verticalement des diagrammes, comme

nous l'avions annoncé dans l'introduction (1.14) ; nous la donnons pour illustrer la façon très intuitive dont se traduit ce genre de raisonnements dans des algèbres relationnelles.

Lemme 3.13. *Une relation x est une* **w***-simulation si et seulement si :*

$$\forall \alpha \in \mathcal{L},\ \overset{\widehat{\alpha}}{\Leftarrow} \cdot x \sqsubseteq x \cdot \overset{\widehat{\alpha}}{\Leftarrow} \ . \tag{3.12}$$

Une relation x est une **p***-simulation si et seulement si :*

$$\forall \alpha \in \mathcal{L},\ \overset{\alpha}{\Leftarrow} \cdot x \sqsubseteq x \cdot \overset{\alpha}{\Leftarrow} \ . \tag{3.13}$$

Démonstration. Dans les deux cas, la réciproque est immédiate.

Soit $x \in X_{\mathbf{w}}$ une **w**-simulation ; par hypothèse, on a $\overset{\tau}{\leftarrow} \cdot x \sqsubseteq x \cdot \overset{\widehat{\tau}}{\Leftarrow}$, d'où $\overset{\widehat{\tau}}{\Leftarrow} \cdot x \sqsubseteq x \cdot \overset{\widehat{\tau}}{\Leftarrow}$ par (2.19). Ensuite, soit $a \in \mathcal{L}^{\mathrm{v}}$ une action visible ; on a

$$\begin{aligned}
\overset{\widehat{a}}{\Leftarrow} \cdot x &\triangleq \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot x \\
&\sqsubseteq \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot x \cdot \overset{\widehat{\tau}}{\Leftarrow} && \text{(par le point précédent)} \\
&\sqsubseteq \overset{\widehat{\tau}}{\Leftarrow} \cdot x \cdot \overset{\widehat{a}}{\Leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} && \text{(par hypothèse)} \\
&\sqsubseteq x \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{\widehat{a}}{\Leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} \ = \ x \cdot \overset{\widehat{\alpha}}{\Leftarrow} \ . && \text{(par le point précédent)}
\end{aligned}$$

Enfin, si x est une **p**-simulation, c'est en particulier une **w**-simulation : l'étude précédente s'applique. Il suffit donc de vérifier $\overset{\tau}{\Leftarrow} \cdot x \sqsubseteq x \cdot \overset{\tau}{\Leftarrow}$; on a pour cela

$$\begin{aligned}
\overset{\tau}{\Leftarrow} \cdot x &= \overset{\tau}{\leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot x \\
&\sqsubseteq \overset{\tau}{\leftarrow} \cdot x \cdot \overset{\widehat{\tau}}{\Leftarrow} && \text{(par le point précédent)} \\
&\sqsubseteq x \cdot \overset{\tau}{\Leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} \ = \ x \cdot \overset{\tau}{\Leftarrow} \ . && \text{(par hypothèse)}
\end{aligned}$$

■

Remarque 3.14 (LTS faible). On retrouve donc bien la première définition de bisimulation faible donnée en introduction (1.7) : les bisimulations faibles sont les bisimulations fortes, prises dans le LTS faible. Cependant, comme expliqué dans l'introduction (ou encore [Mil99, page 53], [SW01, page 93]), cette définition donne lieu à des jeux inutilement difficiles à vérifier. Notre objectif étant de définir des méthodes de preuve, nous ne considérerons pas cette définition par la suite.

3.2.2 Techniques standard : fonctions compatibles

On va maintenant montrer que la plupart des techniques modulo standard pour la simulation peuvent s'exprimer en termes de fonctions compatibles. Vérifions tout d'abord que la notion de simulation modulo, définie

au chapitre précédent, correspond bien à celle décrite informellement dans l'introduction. Considérons par exemple le cas faible (**w**) : une relation x est une **w**-simulation modulo une fonction f si et seulement si $x \sqsubseteq \mathbf{w}(f(x))$, i.e., si $x \rightsquigarrow_{\mathbf{w}} f(x)$, ce qui équivaut au diagramme suivant dès que f est extensive (où l'étiquette α est quantifiée universellement) :

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ & f(x) & \cdot \end{array}$$

On retrouve bien la partie de gauche à droite des diagrammes dessinés dans l'introduction, et la validité de la technique modulo s'obtient en prouvant la correction de la fonction f pour la génératrice considérée (ici, **w**).

On recherche maintenant des fonctions compatibles avec nos différentes génératrices. Le lemme suivant, couplé aux résultats de la section 2.2.5, va nous en fournir un certain nombre.

Lemme 3.15. *Les génératrices* **s** *et* **e** *respectent le monoïde ; les génératrices* **w** *et* **p** *ne le respectent qu'à gauche.*

Démonstration. On applique la caractérisation donnée par le théorème 2.89 : dans les quatre cas, on vérifie bien que 1 est une simulation ; ensuite on distingue les cas :

(**s**) Soient $x, x', y, y' \in X$ telles que $x \rightsquigarrow_{\mathbf{s}} x'$ et $y \rightsquigarrow_{\mathbf{s}} y'$. Pour toute étiquette $\alpha \in \mathcal{L}$, on a :

$$\xleftarrow{\alpha} \cdot x \cdot y \sqsubseteq x' \cdot \xleftarrow{\alpha} \cdot y \sqsubseteq x' \cdot y' \cdot \xleftarrow{\alpha} \ ;$$

on a donc bien $x \cdot y \rightsquigarrow_{\mathbf{s}} x' \cdot y'$ (comme le produit préserve l'ordre partiel, on a immédiatement $x \cdot y \sqsubseteq x' \cdot y'$).

(**e**) Presque identique, il faut simplement faire attention au cas d'une offre silencieuse, où l'on utilise le fait que $y \rightsquigarrow_{\mathbf{e}} y'$ entraîne $y \sqsubseteq y'$:

$$\begin{aligned} \xleftarrow{\tau} \cdot x \cdot y \sqsubseteq x' \cdot \xleftarrow{\widehat{\tau}} \cdot y = x' \cdot \xleftarrow{\tau} \cdot y \vee x' \cdot y \\ \sqsubseteq x' \cdot y' \cdot \xleftarrow{\widehat{\tau}} \vee x' \cdot y' = x' \cdot y' \cdot \xleftarrow{\widehat{\tau}} \ . \end{aligned}$$

(**w**) Comme on ne montre que le respect à gauche, on prend $x, x' \in X$ telles que $x \rightsquigarrow_{\mathbf{w}} x'$ et $y \in X_{\mathbf{w}}$. Pour toute étiquette $\alpha \in \mathcal{L}$, on a :

$$\xleftarrow{\alpha} \cdot x \cdot y \sqsubseteq x' \cdot \xLeftarrow{\widehat{\alpha}} \cdot y \sqsubseteq x' \cdot y \cdot \xLeftarrow{\widehat{\alpha}}$$

(où l'on utilise (3.12) pour la seconde inégalité), ce qui nous donne bien $x \cdot y \rightsquigarrow_{\mathbf{w}} x' \cdot y$.

(**p**) Identique au cas précédent, en utilisant (3.13) au lieu de (3.12). ■

Remarque 3.16. La condition $x \sqsubseteq y$ dans chacune des progressions peut sembler superflue ; elle est cependant nécessaire dans le lemme précédent, afin que **e** respecte le monoïde. On verra au chapitre 5 qu'elle est aussi nécessaire pour les autres progressions, afin d'obtenir la compatibilité de certaines techniques modulo contexte.

Par la proposition 2.67, le lemme 3.15 entraîne la réflexivité et la transitivité des quatre notions de similarité ($\boldsymbol{\nu}\mathbf{s}$, $\boldsymbol{\nu}\mathbf{e}$, $\boldsymbol{\nu}\mathbf{p}$ et $\boldsymbol{\nu}\mathbf{w}$), et admet par les propositions 2.68 et 2.69 le corollaire suivant, qui nous fournit les premières techniques modulo pour les simulations correspondantes :

Corollaire 3.17. *Soient x_w une* **w***-simulation, et x_p une* **p** *simulation.*

1. *La fonction de fermeture réflexive et transitive* ($\mathrm{id}_X^\star : x \mapsto x^\star$)*, est compatible avec* **s** *et* **e**.
2. *La post-composition par x_w ($x \mapsto x \cdot x_w$) est compatible avec* **w**.
3. *La post-composition par x_p ($x \mapsto x \cdot x_p$) est compatible avec* **p**.

Le corollaire suivant n'est qu'une application directe du précédent, reformulée dans des termes plus concrets. Sa preuve illustre la façon dont la proposition 2.50 permet d'assembler plusieurs fonctions compatibles, pour en former de nouvelles.

Corollaire 3.18. *Soit $x \in X$ une relation.*

1. *si x satisfait le diagramme (1) ci dessous, alors $x \sqsubseteq \boldsymbol{\nu}\mathbf{s}$;*
2. *si x satisfait le diagramme (2), alors $x \sqsubseteq \boldsymbol{\nu}\mathbf{w}$.*

(1)
$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \downarrow{\scriptstyle\alpha} \\ & (x \vee \boldsymbol{\nu}\mathbf{s})^\star & \cdot \end{array}$$

(2)
$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ & x^= \cdot \boldsymbol{\nu}\mathbf{w} & \cdot \end{array}$$

Démonstration. 1. La fonction $\mathrm{id}_X^\star \circ (\mathrm{id}_X \vee \widehat{\boldsymbol{\nu}\mathbf{s}})$ est compatible avec **s**, par le corollaire 3.17(1) et la proposition 2.50.

2. De la même façon, mais en utilisant le corollaire 3.17(2), la fonction $(\mathrm{id}_X \vee \widehat{1}) \mathbin{\hat{\cdot}} \widehat{\boldsymbol{\nu}\mathbf{s}}$, égale à $(x \mapsto x \cdot \boldsymbol{\nu}\mathbf{w}) \circ (\mathrm{id}_X \vee \widehat{1})$, est compatible avec **w**. ■

La simulation forte supporte donc la technique modulo transitivité. Notons que la validité de cette technique implique celle de « fort modulo fort », qui correspond à la fonction $f_\sim$ utilisée dans l'introduction : si une relation x satisfait le diagramme suivant, elle satisfait en particulier le diagramme (1) du corollaire 3.18 :

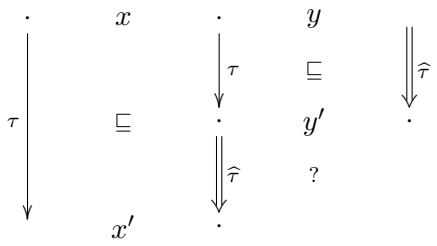
(1')
$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \downarrow{\scriptstyle\alpha} \\ & \boldsymbol{\nu}\mathbf{s} \cdot x \cdot \boldsymbol{\nu}\mathbf{s} & \cdot \end{array}$$

On peut donc utiliser la similarité forte elle même, pour simplifier les processus apparaissant dans les jeux de simulation forte, et ce des deux côtés.

Ce n'est par contre pas le cas de la simulation faible, sur laquelle on se concentre désormais. Nous revenons sur le contre-exemple de l'introduction ci-dessous ; afin de mieux cerner le problème, essayons tout d'abord de prouver que $\mathbf{w}$ respecte le monoïde : supposons $x \rightsquigarrow_{\mathbf{w}} x'$ et $y \rightsquigarrow_{\mathbf{w}} y'$, et tentons de montrer $x \cdot y \rightsquigarrow_{\mathbf{w}} x' \cdot y'$. Prenons le cas d'une offre silencieuse, où nous devons montrer :

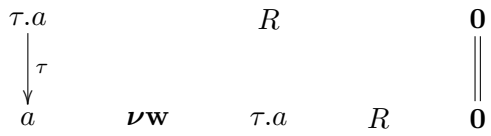
$$\overset{\tau}{\leftarrow} \cdot x \cdot y \sqsubseteq x' \cdot y' \cdot \overset{\widehat{\tau}}{\Leftarrow} .$$

Comme $x \rightsquigarrow_{\mathbf{w}} x'$, on a $\overset{\tau}{\leftarrow} \cdot x \cdot y \sqsubseteq x' \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot y$, mais l'hypothèse $y \rightsquigarrow_{\mathbf{w}} y'$ ne nous permet de fermer qu'un seul pas de réduction silencieuse :



Au contraire, dans la preuve du lemme 3.15, lorsque l'on démontre le respect à gauche du monoïde, l'hypothèse sur y est renforcée (on a $y \rightsquigarrow_{\mathbf{w}} y$), ce qui nous permet de fermer le diagramme par itération (lemme 3.13).

Considérons maintenant le contre-exemple donné en introduction : prenons pour R le singleton $\{\langle \tau.a, \mathbf{0}\rangle\}$, on prouve aisément que $a \ \boldsymbol{\nu}\mathbf{w} \ \tau.a$, de telle sorte que R est une simulation modulo simulation faible à gauche ($R \rightsquigarrow_{\mathbf{w}} \boldsymbol{\nu}\mathbf{w} \cdot R$) :



(cf. 1.15)

R n'est pourtant pas contenue dans la similarité faible : $\tau.a$ peut faire une transition faible visible ($\overset{a}{\Rightarrow}$) à laquelle $\mathbf{0}$ ne peut pas répondre.

Comme expliqué dans l'introduction, on s'aperçoit en fait que la similarité faible permet d'annuler des transitions silencieuses, ce qui permet, si on l'autorise pour réécrire le processus de gauche dans les jeux de simulations, d'« esquiver » des offres et de cacher ainsi le contenu réel d'un processus. L'idée générale que nous allons suivre par la suite va consister à mieux contrôler les relations utilisées pour réécrire le processus de gauche dans les jeux de simulation, pour qu'elles ne permettent pas de « remonter » le long des transitions.

L'usage de l'expansion [SM92] suit justement cette approche : comme ce préordre, contenu dans la bisimulation faible, impose que le processus de droite soit « plus rapide » que celui de gauche, il ne permet pas d'annuler les transitions silencieuses (on n'a pas $a \gtrsim \tau.a$). Dans notre cadre, la validité de la technique correspondante s'exprime par le lemme suivant :

Lemme 3.19. *Pour toute* $\mathbf{e}$*-simulation* x_e*, la fonction* $x \mapsto x_e \cdot x$ *est compatible avec* $\mathbf{w}$*.*

Démonstration. Supposons $x \rightsquigarrow_{\mathbf{w}} y$ et montrons $x_e \cdot x \rightsquigarrow_{\mathbf{w}} x_e \cdot y$. Soit $\alpha \in \mathcal{L}$, on a $\overset{\alpha}{\leftarrow} \cdot x_e \cdot x \sqsubseteq x_e \cdot \overset{\widehat{\alpha}}{\leftarrow} \cdot x$ puisque $x_e \in X_{\mathbf{e}}$.

- Si $\alpha = a \in \mathcal{L}^{\mathrm{v}}$, on en déduit $\overset{a}{\leftarrow} \cdot x_e \cdot x \sqsubseteq x_e \cdot \overset{a}{\leftarrow} \cdot x \sqsubseteq x_e \cdot y \cdot \overset{a}{\Leftarrow}$;
- sinon, on a $\overset{\tau}{\leftarrow} \cdot x \sqsubseteq y \cdot \overset{\widehat{\tau}}{\Leftarrow}$ et $1 \cdot x = x \sqsubseteq y \cdot \overset{\widehat{\tau}}{\Leftarrow}$ (rappelons que $x \rightsquigarrow_{\mathbf{w}} y$ implique $x \sqsubseteq y$), d'où $\overset{\tau}{\leftarrow} \cdot x_e \cdot x \sqsubseteq x_e \cdot \overset{\tau\,=}{\leftarrow} \cdot x \sqsubseteq x_e \cdot y \cdot \overset{\widehat{\tau}}{\Leftarrow}$. ■

On peut donc raffiner la seconde partie du corollaire 3.18 comme suit : pour qu'une relation x soit majorée par la similarité faible, il suffit qu'elle satisfasse le diagramme suivant :

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ \boldsymbol{\nu}\mathbf{e} \cdot x^{=} \cdot \boldsymbol{\nu}\mathbf{w} & & \cdot \end{array} \tag{2'}$$

La restriction à la pré-expansion pour réécrire le processus de gauche peut sembler très contraignante : la définition de la pré-expansion la rapproche plus de la similarité forte que de la similarité faible. Ce n'est toutefois dû qu'à notre présentation en deux temps de l'expansion : les jeux que ce préordre imposera de droite à gauche, qui ne sont pour l'instant pas mentionnés, seront des jeux de simulation faible, ce qui permettra à l'expansion d'être bien moins rigide que la bisimulation forte.

3.2.3 Transitivité lors des offres visibles

Une première façon d'obtenir la technique modulo transitivité consiste à considérer la simulation forte dans le LTS faible (cf. introduction et remarque 3.14) ; c'est l'approche que suit Milner lorsqu'il corrige le problème de faible modulo faible dans la seconde édition de son livre [Mil89].

Si l'on souhaite garder des offres simples, on peut remarquer que les actions visibles sont jouées de façon relativement stricte, même dans la simulation faible : on doit répondre à toute action visible a une unique action a. La simulation faible est donc moralement capable de compter ces actions, et ne permet pas de les annuler : on est dans une situation similaire au cas fort. Cette intuition mène au lemme suivant, qui permet de répondre aux offres visibles modulo transitivité :

Lemme 3.20. *Soit x une relation.*

$$\textit{Si } \begin{cases} \overset{\tau}{\leftarrow} \cdot x \sqsubseteq x \cdot \overset{\widehat{\tau}}{\Leftarrow} \textit{ et} \\ \overset{a}{\leftarrow} \cdot x \sqsubseteq x^\star \cdot \overset{a}{\Leftarrow} \ , \forall a \in \mathcal{L}^{\mathrm{v}}, \end{cases} \qquad \textit{alors } x^\star \textit{ est une } \mathbf{w}\textit{-simulation.}$$

Démonstration. On a tout d'abord $\overset{\widehat{\tau}}{\Leftarrow} \cdot x \sqsubseteq x \cdot \overset{\widehat{\tau}}{\Leftarrow}$, par (2.19) ; ce qui mène à $\overset{\widehat{\tau}}{\Leftarrow} \cdot x^\star \sqsubseteq x^\star \cdot \overset{\widehat{\tau}}{\Leftarrow}$ ($\star$), par (2.20).

Par une récurrence, on démontre ensuite $\forall n \in \mathbb{N}$, $\overset{a}{\leftarrow} \cdot x^n \sqsubseteq x^\star \cdot \overset{a}{\Leftarrow}$. Le cas de base est trivial ; pour le cas inductif, on a

$$\begin{aligned}
\overset{a}{\leftarrow} \cdot x^{n+1} \triangleq \overset{a}{\leftarrow} \cdot x \cdot x^n &\sqsubseteq x^\star \cdot \overset{a}{\Leftarrow} \cdot x^n && \text{(par hypothèse sur } x) \\
&\triangleq x^\star \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot x^n \\
&\sqsubseteq x^\star \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot x^n \cdot \overset{\widehat{\tau}}{\Leftarrow} && \text{(par } (*)) \\
&\sqsubseteq x^\star \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot x^\star \cdot \overset{a}{\Leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} && \text{(par récurrence)} \\
&\sqsubseteq x^\star \cdot x^\star \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\Leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} = x^\star \cdot \overset{a}{\Leftarrow} . && \text{(par } (\star))
\end{aligned}$$

La continuité du produit permet de conclure. ■

Ce résultat est une technique modulo, au sens où il facilite la tâche consistant à trouver des simulations faibles, mais il ne peut pas être exprimé sous la forme d'une fonction correcte pour $\mathbf{w}$: la distinction que l'on fait selon le type d'offre nous en empêche. On peut cependant définir une autre génératrice, dont on montre par le lemme précédent qu'elle définit la même similarité :

Définition 3.21 (Génératrice adaptée des simulations faibles)**.** On définit la génératrice $\mathbf{w}_{\mathrm{t}}$ comme étant la fonction associée à la progression suivante :

$$x \rightsquigarrow_{\mathbf{w}_{\mathrm{t}}} y \quad \text{si} \quad \begin{cases} x \sqsubseteq y \ , \\ \overset{\tau}{\leftarrow} \cdot x \sqsubseteq y \cdot \overset{\widehat{\tau}}{\Leftarrow} \ , \\ \forall a \in \mathcal{L}^{\mathrm{v}}, \ \overset{a}{\leftarrow} \cdot x \sqsubseteq y^+ \cdot \overset{a}{\Leftarrow} \ . \end{cases}$$

Le choix d'une fermeture strictement transitive (id_X^+) plutôt que réflexive et transitive (comme le permet le lemme 3.20) est purement technique : cela permet de simplifier certaines preuves. On ne perd pas en généralité : la fonction de fermeture réflexive reste compatible avec $\mathbf{w}_{\mathrm{t}}$ (lemme 3.23 ci-dessous) et fournit ainsi une technique qui compense cette restriction.

Proposition 3.22. *Pour toute $\mathbf{w}_t$-simulation x, $x^\star$ est une $\mathbf{w}$-simulation. La $\mathbf{w}_t$-similarité et la $\mathbf{w}$-similarité coïncident :*

$$\boldsymbol{\nu}\mathbf{w}_t = \boldsymbol{\nu}\mathbf{w} \ .$$

Cette nouvelle génératrice $\mathbf{w}_\mathrm{t}$ ne respecte pas plus le monoïde que $\mathbf{w}$; mais les fonctions compatibles que l'on avait pour $\mathbf{w}$ sont globalement conservées (les hypothèses de réflexivité ne sont pas restrictives) :

Lemme 3.23. *Les fonctions suivantes sont compatibles avec* $\mathbf{w}_t$ *:*

1. *la fonction de fermeture réflexive* $(\mathrm{id}^{=}_X)$ *;*
2. *la pré-composition par toute* $\mathbf{e}$*-simulation réflexive* x_e $(x \mapsto x_e \cdot x)$ *;*
3. *la post-composition par toute* $\mathbf{w}$*-simulation réflexive* x_w $(x \mapsto x \cdot x_w)$*.*

Démonstration. Le fait que 1 soit une $\mathbf{w}_\mathrm{t}$-simulation mène au premier point par la proposition 2.50 ; la preuve du second point suit celle du lemme 3.19, la réflexivité de x_e est nécessaire dans le cas des offres visibles, afin d'obtenir l'inégalité $x_e \cdot y^+ \sqsubseteq (x_e \cdot y)^+$ (c'est pour cette obtenir inégalité qu'il est préférable de considérer la transitivité stricte dans la définition de $\mathbf{w}_\mathrm{t}$: elle n'est plus vraie si l'on rajoute dès le départ la fermeture réflexive). Le troisième point est similaire au précédent. ■

Pour montrer qu'une relation x est majorée par la similarité faible, il suffit finalement de montrer qu'elle satisfait les deux diagrammes suivants, où l'étiquette a est quantifiée universellement sur toutes les actions visibles dans le second diagramme (pour justifier ce diagramme, il suffit de constater que l'on a $(\boldsymbol{\nu}\mathbf{e} \cdot x^{=} \cdot \boldsymbol{\nu}\mathbf{w})^+ = (x \vee \boldsymbol{\nu}\mathbf{w})^\star$) :

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\tau}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle\widehat{\tau}} \\ \cdot & \boldsymbol{\nu}\mathbf{e} \cdot x^{=} \cdot \boldsymbol{\nu}\mathbf{w} & \cdot \end{array} \qquad \begin{array}{ccc} \cdot & x & \\ {\scriptstyle a}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle a} \\ \cdot & (x \vee \boldsymbol{\nu}\mathbf{w})^\star & \cdot \end{array} \tag{2"}$$

La technique ainsi obtenue est finalement strictement plus puissante que celle obtenue avec la première génératrice (2'). On pourrait donc être tenté d'abandonner la première génératrice, au profit de la seconde. On verra cependant au chapitre 5 que les techniques modulo contexte ne sont valides que pour la première. Nous considérons donc dans la suite les deux génératrices, afin de garder la possibilité de choisir, à la fin, entre les techniques modulo contexte et la transitivité lors des offres visibles.

Le lemme suivant nous permettra cependant de nous concentrer quand même principalement sur la génératrice $\mathbf{w}_\mathrm{t}$; il n'admet pas de réciproque, en particulier du fait des techniques modulo contexte :

Lemme 3.24. *Toute fonction* f *correcte pour* $\mathbf{w}_t$ *via une fonction* f' *est correcte pour* $\mathbf{w}$ *via* $f'^\star$*.*

Démonstration. Soit x une $\mathbf{w}$-simulation modulo f : $x \rightsquigarrow_{\mathbf{w}} f(x)$. On a en particulier $x \rightsquigarrow_{\mathbf{w}_\mathrm{t}} f(x)$, et $f'(x)$ est donc une $\mathbf{w}_\mathrm{t}$-simulation. Le lemme 3.20 permet de conclure : $f'(x)^\star$ est une $\mathbf{w}$-simulation. ■

Remarque 3.25 (Construction étagée). La transitivité lors des offres visibles pourrait être obtenue en utilisant la construction étagée évoquée à la section 2.4, en séparant la génératrice des simulations faibles ($\mathbf{w}$) en une première génératrice $\mathbf{w}^\tau$ correspondant à la partie silencieuse des jeux, et une seconde, $\mathbf{w}^v$, correspondant aux jeux visibles. La proposition 2.90 pourrait alors utilisée : la fonction de fermeture réflexive transitive est compatible avec $\mathbf{w}^v$, sur l'ensemble $X_{\mathbf{w}^\tau}$ des « simulations silencieuses » (notons bien qu'elle ne l'est pas sur X) ; et les techniques telles que modulo expansion satisfont les hypothèses que cette proposition demande pour la fonction f.

Remarque 3.26 (Simulation progressive). L'ensemble des techniques précédemment développées peuvent s'adapter à la simulation progressive. Par contre, le contre-exemple invalidant la technique « modulo faible » ne s'applique pas : le processus $\mathbf{0}$ ne peut pas répondre avec au moins une transition silencieuse. Comme on le verra au chapitre suivant (section 4.5.1), dans des systèmes où toute séquence de transitions silencieuses est finie, $\mathbf{p}$ supporte la technique modulo transitivité. On peut néanmoins facilement trouver un contre-exemple lorsque l'on a des séquences infinies de transitions silencieuses ($!\tau$ dans CCS), qui peuvent être utilisées pour gommer la différence entre simulations faibles et progressives (cf. remarque 3.29) : il suffit par exemple de prendre le singleton $\{\langle \tau.a, !\tau \rangle\}$.

3.3 Extension aux jeux à deux côtés

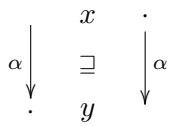
Maintenant que l'on a étudié les différents jeux de simulation, où l'on demande au côté droit de répondre au côté gauche, on regarde comment symétriser ces jeux, afin d'obtenir les équivalence et préordres couramment étudiés dans la littérature. On commence par les définir, puis on étudie les techniques modulo qui leur sont associées.

3.3.1 Définition des équivalences comportementales standard

Il suffit pour cela de considérer les converses des génératrices de la section précédente : par exemple, pour la bisimulation forte, la converse de la génératrice $\mathbf{s}$, $\overline{\mathbf{s}}$, correspond à des jeux de simulation forte de droite à gauche : pour tous x, y, on a :

$$\begin{aligned} x \sqsubseteq \overline{\mathbf{s}}(y) &\Leftrightarrow \overline{x} \sqsubseteq \mathbf{s}(\overline{y}) \\ &\Leftrightarrow \overline{x} \rightsquigarrow_{\mathbf{s}} \overline{y} \\ &\Leftrightarrow \forall \alpha \in \mathcal{L},\ \xleftarrow{\alpha} \cdot \overline{x} \sqsubseteq \overline{y} \cdot \xleftarrow{\alpha} \\ &\Leftrightarrow \forall \alpha \in \mathcal{L},\ x \cdot \xrightarrow{\alpha} \sqsubseteq \xrightarrow{\alpha} \cdot y \end{aligned}$$

ce qui est équivalent au diagramme suivant :



En utilisant les résultats de la section 2.2.5, on peut alors combiner les différentes génératrices « gauche-droite » avec les différentes génératrices « droite-gauche », et obtenir des point fixes correspondant à des jeux à deux côtés.

Définition 3.27 (Les différents préordres et équivalences)**.** La *bisimilarité forte* ($\sim$), la *bisimilarité faible* ($\approx$), la *bi-expansion* ($\asymp$), la *bisimilarité progressive* ($\simeq$), l'*expansion* ($\precsim$) et l'*élaboration* ($\precapprox$) sont définies comme suit :

$$\sim \triangleq \boldsymbol{\nu}^{\overleftrightarrow{\mathbf{s}}} \qquad \approx \triangleq \boldsymbol{\nu}^{\overleftrightarrow{\mathbf{w}}} \qquad \asymp \triangleq \boldsymbol{\nu}^{\overleftrightarrow{\mathbf{e}}} \qquad \simeq \triangleq \boldsymbol{\nu}^{\overleftrightarrow{\mathbf{p}}}$$

$$\precsim \triangleq \boldsymbol{\nu}(\mathbf{e} \wedge \overline{\mathbf{w}}) \qquad \precapprox \triangleq \boldsymbol{\nu}(\mathbf{w} \wedge \overline{\mathbf{p}})$$

On appellera de plus *bisimulations fortes*, *bisimulations faibles*, *bi-expansions*, *bisimulations progressives*, *expansions* et *élaborations* les post-points fixes de ces génératrices [2]. Les bisimilarités fortes et faibles sont bien connues, nous détaillons les autres relations ci-dessous.

($\precsim$) L'expansion [AKH92, SM92] combine un jeu de pré-expansion de gauche à droite à un jeu de simulation faible, de droite à gauche. Cela lui permet d'être utilisée comme technique modulo pour la bisimulation faible, tout en restant relativement proche de celle-ci : seul le côté droit est contraint.

($\asymp$) La bi-expansion est la version symétrique de l'expansion ; elle apparaît dans [Fou98, chapitre 4] en tant que technique modulo pour l'expansion.

($\precapprox$) L'élaboration [AKN95] combine un jeu de simulation faible de gauche à droite avec un jeu de simulation progressive de droite à gauche, imposant au processus de gauche d'être au moins aussi lent que celui de droite. Elle est donc en quelque sorte duale à l'expansion, qui demande que le processus de droite soit au moins aussi rapide que celui de gauche (notons bien que dans les deux cas, c'est le processus de gauche qui est le moins efficace).

($\simeq$) La bisimilarité progressive [MS92c] est la version symétrique de l'élaboration : chaque côté s'efforce d'être aussi lent que l'autre. Il est montré dans [MS92c] que cette équivalence est la plus grande bisimulation faible qui soit en même temps une congruence (dans CCS) ; nous retrouverons ce résultat au chapitre 5.

2. Lorsqu'il y a conflit, l'article lèvera l'ambiguïté : l'élaboration est la relation $\precapprox$, une élaboration est une $(\mathbf{w} \wedge \overline{\mathbf{p}})$-simulation.

Les techniques modulo pour l'élaboration et la bisimulation progressive étant étudiées au chapitre suivant (section 4.5.1), nous nous contentons ici de comparer ces relations aux autres équivalences et préordres comportementaux.

Proposition 3.28. *Les relations précédemment définies sont réflexives et transitives ; $\sim$, $\approx$, $\asymp$ et $\simeq$ sont symétriques. On a les inégalités suivantes :*

$$\sim \sqsubseteq \asymp \sqsubseteq \gtrsim \sqsubseteq \approx \ ,$$
$$\sim \sqsubseteq \simeq \sqsubseteq \gtrapprox \sqsubseteq \approx \ .$$

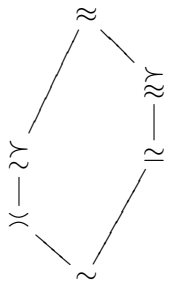
Démonstration. Le premier point provient de la proposition 2.67 et du corollaire 2.72. Ensuite, par la proposition 3.12, on a :

$$\begin{array}{ccccccc} \mathbf{s}\wedge\overline{\mathbf{s}} & \sqsubseteq & \mathbf{e}\wedge\overline{\mathbf{e}} & \sqsubseteq & \mathbf{e}\wedge\overline{\mathbf{w}} & \sqsubseteq & \mathbf{w}\wedge\overline{\mathbf{w}} \\ \mathbf{s}\wedge\overline{\mathbf{s}} & \sqsubseteq & \mathbf{p}\wedge\overline{\mathbf{p}} & \sqsubseteq & \mathbf{w}\wedge\overline{\mathbf{p}} & \sqsubseteq & \mathbf{w}\wedge\overline{\mathbf{w}} \end{array}$$

ce qui conduit par la proposition 2.43 aux inégalités annoncées. ■

Ces inégalités, représentées par le graphe ci-dessous, sont en général strictes, et les relations $\asymp$ et $\gtrsim$ sont de plus incomparables à $\simeq$ et $\gtrapprox$. En utilisant des processus CCS, on a par exemple :

$a \approx \tau.a$	mais	$a \not\gtrsim \tau.a$ et $a \not\gtrapprox \tau.a$,
$\tau.a \gtrsim a$ et $\tau.a \gtrapprox a$	mais	$\tau.a \not\asymp a$ et $\tau.a \not\simeq a$,
$\tau + \tau.\tau \asymp \tau.\tau$ et $\tau + \tau.\tau \simeq \tau.\tau$	mais	$\tau + \tau.\tau \not\sim \tau.\tau$,
$a \asymp a + \tau.a$	mais	$a \not\gtrapprox a + \tau.a$,
$\tau.a + \tau.\tau.a \simeq \tau.\tau.a$	mais	$\tau.a + \tau.\tau.a \not\gtrsim \tau.\tau.a$.



Remarque 3.29. Nous avons dessiné le graphe précédent de façon asymétrique, car la bisimulation progressive ($\simeq$) (et donc l'élaboration ($\gtrapprox$)) est intuitivement plus proche de la bisimilarité faible que la bi-expansion (ou même que l'expansion) : dans CCS, la bisimulation progressive est la plus grande bisimulation faible qui soit une congruence [MS92c] (nous retrouvons ce résultat au chapitre 5 : corollaire 5.29). De plus les séquences infinies de transitions permettent de gommer la différence entre les deux équivalences :

$$\text{si } p \approx q \ , \text{ alors } p \mid !\tau \simeq q \mid !\tau$$

(comme $!\tau \xrightarrow{\tau} !\tau$, la contrainte imposée par la bisimulation progressive peut être artificiellement évitée).

Notons que ce genre de résultat ne tient pas pour l'expansion : par exemple, on n'a pas $a \mid !\tau \gtrsim \tau.a \mid !\tau$: rajouter artificiellement des étapes silencieuses des deux côtés ne permet pas au processus de droite de devenir aussi rapide que celui de gauche.

Remarque 3.30. Une autre définition standard des bisimulations passe par la notion de symétrie : « une bisimulation est une simulation symétrique ». Cela correspond en fait à la caractérisation alternative donnée par la proposition 2.65. Ce choix n'a pas d'impact en termes de techniques modulo (tant que l'on autorise des fonctions non-symétriques, mais correctes via des fonctions symétriques, on peut toujours considérer la fermeture symétrique des candidats de bisimulation). Nous préférons travailler avec notre première définition, qui nous permet de considérer de manière uniforme les équivalences comme les préordres, où les jeux utilisés pour chacun des deux cotés ne sont pas identiques.

3.3.2 Traces, 2-similarité et bisimilarité

Avant de s'attaquer au techniques modulo pour les jeux symétriques, nous faisons un détour par une autre notion d'équivalence déjà évoquée dans l'introduction : la 2-similarité. Cette section est relativement annexe, et ne nous servira pas dans la suite ; elle nous permet principalement à illustrer la façon dont diverses notions standard peuvent être étudiées au niveau des algèbres relationnelles.

Comme expliqué au chapitre précédent (section 2.2.3), le fait de s'intéresser à la bisimilarité ($\sim \triangleq \boldsymbol{\nu}\mathbf{s} \wedge \overline{\mathbf{s}}$) plutôt qu'à la *2-similarité* ($\boldsymbol{\nu}\mathbf{s} \wedge \boldsymbol{\nu}\overline{\mathbf{s}}$) nous oblige en effet à considérer la notion de fonction témoin lorsque l'on étudie les techniques modulo. Par exemple, par le théorème 2.62, c'est lorsque la correction d'une fonction peut être prouvée via une fonction *symétrique* que la technique correspondante pour le jeu de simulation (s) peut être utilisée dans les jeux de bisimulation ($\overleftrightarrow{s}$). Cette contrainte nous forcera à effectuer dans les sections suivantes des calculs parfois pénibles, afin de prouver la symétrie de certaines fonctions. Au contraire, lorsque l'on considère la 2-similarité, cette contrainte disparaît et on peut jouer les deux jeux de simulation indépendamment :

Proposition 3.31. *Soient f, g deux fonctions correctes pour $\mathbf{s}$. Si x est une $\mathbf{s}$-simulation modulo f, et une $\overline{\mathbf{s}}$-simulation modulo $\overline{g}$, alors x est majorée par la 2-similarité.*

Démonstration. Par la correction de f, on a $x \sqsubseteq \boldsymbol{\nu}\mathbf{s}$, et la correction de g nous donne $x \sqsubseteq \boldsymbol{\nu}\overline{\mathbf{s}}$ (par la proposition 2.61(3)). Le résultat en découle. ■

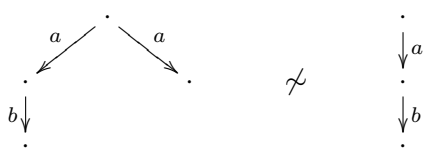
En particulier, toute relation symétrique qui est une **s**-simulation modulo une fonction correcte pour **s** est majorée par la 2-similarité. Si c'est effectivement la 2-similarité qui nous intéresse, les techniques modulo pour les différentes notions de simulation de gauche à droite sont donc suffisantes, et une bonne partie de ce chapitre devient inutile.

Coïncidence des deux équivalences : déterminisme

Comme expliqué dans l'introduction, la 2-similarité est cependant souvent considérée comme une équivalence trop faible : elle ne prend pas en compte les « blocages ». Nous pouvons toutefois montrer que la 2-similarité et la bisimilarité coïncident sous certaines conditions de déterminisme. Lorsque c'est le cas, la 2-similarité devient une équivalence raisonnable, dont les techniques modulo s'obtiennent plus facilement.

Nous allons ainsi retrouver la notion de *processus déterministe*, décrite par Milner [Mil89, chapitre 11]. Très grossièrement, un processus est déterministe si aucun de ses descendants ne peut faire deux transitions différentes selon la même étiquette. De tels processus sont considérés comme plus « propres » que les autres, au sens où leur comportement peut plus facilement être appréhendé. Milner étudie donc les propriétés de cette famille de « bons » processus (préservation par réduction, par bisimilarité), et caractérise les opérateurs syntaxiques qui permettent de conserver cette propriété (il passe pour cela par une notion un peu plus forte de « confluence »). Ce genre de notions ont ensuite été étudiées par Jan Friso Groote et Alex Sellink [GS96], Uwe Nestmann [Nes96] ; puis généralisées au π-calcul par Anna Philippou et Walker [PW97]. Roberto Amadio et Mehdi Dogguy on récemment défini un système de types garantissant le déterminisme dans une extension « synchrone » du π-calcul [AD07] (synchrone au sens des langages SL [BdS96] ou ESTEREL [BG92]).

Un résultat important de Milner est que deux processus déterministes ayant les mêmes traces sont bisimilaires ; ainsi, lorsque tous les processus sont déterministes, même l'équivalence des traces devient suffisamment fine : elle coïncide avec la bisimilarité… Revenant à la 2-similarité, le contre-exemple (1.9) de l'introduction (rappelé ci-dessous : les processus $a.b + a$ et $a.b$ sont 2-similaires, mais pas bisimilaires) utilise typiquement un processus non déterministe : $a.b + a$ peut effectuer deux transitions différentes selon la même étiquette [3].



3. C'est le même phénomène qui entre en jeu avec le distributeur de boissons (1.4).

Dans notre cadre abstrait, qui ne permet pas de mentionner les individus (ici les processus), nous devons rejeter la faute sur la relation de transition ($\xrightarrow{a}$) dans son ensemble, qui n'est pas déterministe au sens de la définition suivante :

Définition 3.32 (Relation déterministe)**.** Soient $\equiv, x \in X$ deux relations ; *x est déterministe modulo $\equiv$* si $\overline{x} \cdot x \sqsubseteq \equiv$.

Une relation déterministe modulo 1 sera simplement dite *déterministe* (de telles relations sont parfois dites *fonctionnelles* : dans l'algèbre relationnelle propre d'un ensemble $\mathcal{P}$, elles peuvent être considérées comme des applications partielles de $\mathcal{P}$ dans lui-même – une relation binaire ($\rightarrow$) y est déterministe si et seulement si pour tous $p, p_1, p_2 \in \mathcal{P}$, $p \rightarrow p_1$ et $p \rightarrow p_2$ impliquent $p_1 = p_2$).

Cette définition nous permet d'obtenir la proposition suivante (la bisimilarité forte étant réflexive, notons que le déterminisme implique le déterminisme modulo $\sim$) :

Proposition 3.33. *Si pour toute étiquette $\alpha \in \mathcal{L}$, $\xrightarrow{\alpha}$ est déterministe modulo $\sim$, alors la 2-similarité forte et la bisimilarité forte coïncident :*

$$\sim \; = \boldsymbol{\nu}\mathbf{s} \wedge \overline{\boldsymbol{\nu}\mathbf{s}} \quad .$$

Démonstration. L'inclusion directe a déjà été établie par la proposition 2.54. Pour sa converse, on démontre que pour toutes **s**-simulations $x, y \in X_s$, $x \wedge (\sim \cdot \overline{y})$ est aussi une **s**-simulation : pour tout $\alpha \in \mathcal{L}$, on a

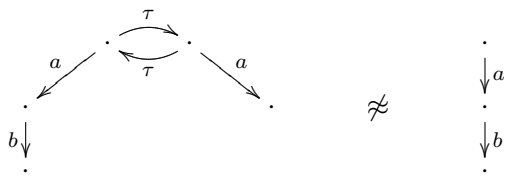
$$\begin{aligned}
\xleftarrow{\alpha} \cdot (x \wedge \sim \cdot \overline{y}) &\sqsubseteq \xleftarrow{\alpha} \cdot x \wedge \xleftarrow{\alpha} \cdot \sim \cdot \overline{y} \\
&\sqsubseteq x \cdot \xleftarrow{\alpha} \wedge \xleftarrow{\alpha} \cdot \sim \cdot \overline{y} && (x \in X_{\mathbf{s}}) \\
&\sqsubseteq (x \wedge \xleftarrow{\alpha} \cdot \sim \cdot \overline{y} \cdot \xrightarrow{\alpha}) \cdot \xleftarrow{\alpha} && \text{(loi de modularité (3.8))} \\
&\sqsubseteq (x \wedge \xleftarrow{\alpha} \cdot \sim \cdot \xrightarrow{\alpha} \cdot \overline{y}) \cdot \xleftarrow{\alpha} && (y \in X_{\mathbf{s}}) \\
&\sqsubseteq (x \wedge \sim \cdot \xleftarrow{\alpha} \cdot \xrightarrow{\alpha} \cdot \overline{y}) \cdot \xleftarrow{\alpha} && (\overline{\sim} = \sim \in X_{\mathbf{s}}) \\
&\sqsubseteq (x \wedge \sim \cdot \sim \cdot \overline{y}) \cdot \xleftarrow{\alpha} && (\xrightarrow{\alpha} \text{ est déterministe modulo } \sim) \\
&\sqsubseteq (x \wedge \sim \cdot \overline{y}) \cdot \xleftarrow{\alpha} && (\sim \text{ est transitive})
\end{aligned}$$

En prenant $x = \boldsymbol{\nu}\mathbf{s} \cdot \sim$ et $y = \boldsymbol{\nu}\mathbf{s}$, on obtient que $\boldsymbol{\nu}\mathbf{s} \cdot \sim \wedge \sim \cdot \overline{\boldsymbol{\nu}\mathbf{s}}$ est une **s**-simulation, ce qui permet de conclure : cette relation est symétrique, et majore $\boldsymbol{\nu}\mathbf{s} \wedge \overline{\boldsymbol{\nu}\mathbf{s}}$. ■

Notons qu'il s'agit du seul résultat de ce chapitre dont la preuve nécessite la loi de modularité.

Le cas faible

Ce résultat ne s'étend pas directement au cas faible : $a + \tau$ et a ont beau être déterministes (modulo I), ils sont faiblement 2-similaires ($\boldsymbol{\nu}\mathbf{w} \wedge \overline{\boldsymbol{\nu}\mathbf{w}}$), sans être faiblement bisimilaires. Comme le montre la variante suivante du contre-exemple dans le cas fort, demander en plus que les transitions silencieuses soient contenues dans la bisimilarité faible n'aide pas :



On peut cependant appliquer le résultat précédent sur le LTS faible, ce qui a pour effet de renforcer l'hypothèse de déterminisme : ce ne sont plus les transitions fortes qui doivent être déterministes, mais les transitions faibles.

Avant de donner ce corollaire, nous reformulons dans notre cadre deux notions : *déterminisme faible* et τ*-inertie*, introduites par Milner [Mil89] ; la seconde fait l'objet d'une étude détaillée par Groote et Sellink [GS96] :

Définition 3.34 (LTS faiblement déterministe, LTS τ-inerte)**.** Un LTS est dit *faiblement déterministe* lorsque $\stackrel{\widehat{\alpha}}{\Rightarrow}$ est déterministe modulo $\approx$ pour toute étiquette $\alpha \in \mathcal{L}$; il est dit τ*-inerte* lorsque $\stackrel{\tau}{\rightarrow} \sqsubseteq \approx$.

(pour le déterminisme faible, par une simple récurrence, notre définition est bien équivalente à celle de Milner, qui considère le déterminisme des séquences arbitraires de transitions faibles). Notons qu'un LTS faiblement déterministe est nécessairement τ-inerte.

Corollaire 3.35. *Si le LTS est faiblement déterministe, alors la 2-similarité faible et la bisimilarité faible coïncident :*

$$\approx \; = \boldsymbol{\nu}\mathbf{w} \wedge \overline{\boldsymbol{\nu}\mathbf{w}} \quad .$$

Démonstration. On applique la proposition 3.33 sur le LTS faible, $(\stackrel{\widehat{\alpha}}{\Rightarrow})_{\alpha \in \mathcal{L}}$. ■

Vérifier le déterminisme faible n'est cependant pas forcément facile : c'est une propriété globale, puisque l'on a deux quantifications universelles sur des séquences potentiellement infinies de transitions silencieuses. On aimerait donc bien pouvoir localiser cette propriété, en se ramenant au déterminisme des transitions fortes. Le lemme suivant permet de retirer l'une des deux quantifications ; la proposition qui suit donne une condition locale relativement simple.

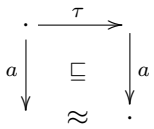
Lemme 3.36. *Soit $a \in \mathcal{L}^{\mathrm{v}}$ une étiquette visible d'un LTS τ-inerte.*
Si $\overset{a}{\Leftarrow} \cdot \overset{a}{\Rightarrow} \sqsubseteq \approx$, alors $\overset{a}{\Rightarrow}$ est déterministe modulo $\approx$.

Démonstration. On a

$$\begin{aligned}
\overset{a}{\Leftarrow} \cdot \overset{a}{\Rightarrow} &\triangleq \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\Rightarrow} \\
&\sqsubseteq \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot \overset{a}{\Rightarrow} \cdot \approx && (\tau\text{-inertie et (3.12)}) \\
&\sqsubseteq \overset{\widehat{\tau}}{\Leftarrow} \cdot \approx \cdot \approx && (\text{par hypothèse}) \\
&\sqsubseteq \approx \quad . && (\tau\text{-inertie et transitivité})
\end{aligned}$$

■

Proposition 3.37. *Dans tout LTS τ-inerte dont les transitions silencieuses satisfont de plus le diagramme suivant, le déterminisme de $\overset{a}{\rightarrow}$ modulo $\approx$ entraîne le déterminisme de $\overset{a}{\Rightarrow}$ modulo $\approx$.*



Démonstration. On montre d'abord $\overset{a}{\leftarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} \sqsubseteq \approx \cdot \overset{a}{\leftarrow}$ $(\star)$, par une simple récurrence. On a ensuite :

$$\begin{aligned}
\overset{a}{\leftarrow} \cdot \overset{a}{\Rightarrow} &\triangleq \overset{a}{\leftarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} \cdot \overset{a}{\rightarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} \\
&\sqsubseteq \approx \cdot \overset{a}{\leftarrow} \cdot \overset{a}{\rightarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} && (\star) \\
&\sqsubseteq \approx \cdot \approx \cdot \overset{\widehat{\tau}}{\Rightarrow} && (\text{déterminisme de } \overset{a}{\rightarrow}) \\
&\sqsubseteq \approx && (\tau\text{-inertie et transitivité})
\end{aligned}$$

On conclut par le lemme 3.36. ■

Cette condition renforce en fait l'hypothèse d'inertie : dans le jeu de bisimulation imposé à $(\overset{\tau}{\rightarrow})$, les transitions visibles doivent être jouées immédiatement, et non pas après quelques transitions silencieuses (remarquons que cette condition est satisfaite lorsque $\overset{\tau}{\rightarrow}$ est en fait contenue dans l'expansion $(\precsim)$). Le LTS que l'on définit sur le λ-calcul pour obtenir la *bisimilarité applicative* [Abr90] satisfait en particulier cette première condition ; il suffit donc dans ce cadre d'étudier la similarité faible : la bisimilarité n'apporte rien de plus.

Nous verrons au chapitre 4 (section 4.5.2) que le déterminisme des transitions fortes modulo $\approx$ suffit lorsque le LTS considéré n'admet pas de séquences infinies de transitions silencieuses (ces LTS seront dits *réactifs* – le contre-exemple précédent est typiquement non-réactif puisqu'un cycle de transitions silencieuses y est exhibé). Nous donnerons aussi des techniques

permettant de simplifier les preuves de τ-inertie lorsque cette hypothèse de terminaison est satisfaite.

Notre vision abstraite des relations nous a forcé à définir les notions précédentes de façon globale : ce sont des propriétés du LTS dans son ensemble. Pour obtenir les propriétés correspondantes sur des processus concrets (un processus est déterministe si...), il suffit de considérer le LTS restreint aux descendants du processus considéré.

Équivalence des traces

Dans le cas des relations binaires, Milner démontre un résultat plus fort que le corollaire 3.35 : sous les mêmes conditions, l'équivalence des traces (faibles) coïncide avec la bisimilarité faible [Mil89, chapitre 11, proposition 5].

A notre niveau d'abstraction, il est cependant délicat de définir directement l'équivalence des traces. Une façon indirecte de l'obtenir consiste à définir un LTS où la bisimilarité correspond à cette équivalence. Nous esquissons brièvement cette idée, reportant à plus tard une étude plus détaillée ; nous ne considérons que le cas fort, le cas faible pouvant s'obtenir en se plaçant dès le départ dans le LTS faible.

Dénotons par s, t les *traces*, c'est-à-dire les séquences finies d'étiquettes. Notons ϵ la trace vide, et $\mathcal{L}^\star$ l'ensemble de ces traces. Les relations de transition selon $\mathcal{L}$ s'étendent naturellement en des relations de transition selon $\mathcal{L}^\star$:

$$\xrightarrow{\epsilon} \triangleq 1 \qquad\qquad \xrightarrow{\alpha t} \triangleq \xrightarrow{\alpha} \cdot \xrightarrow{t} \ ,$$

de telle sorte que $(\xrightarrow{t})_{t\in\mathcal{L}^\star}$ forme un nouveau LTS. Comme la bisimilarité ainsi obtenue coïncide avec celle du LTS initial, on a seulement gagné la possibilité de parler plus facilement des traces.

Pour toute trace t, définissons maintenant la relation suivante, qui caractérise le « domaine » de la relation de transition $\xrightarrow{t}$: dans une algèbre relationnelle propre, c'est la relation identité, restreinte aux processus capables d'exhiber la trace t [4] :

$$1_t \triangleq (\xrightarrow{t} \cdot \top) \wedge 1 \ .$$

(l'élément maximal du treillis ($\top$) représente la relation binaire universelle). L'ensemble de ces relations forme un LTS : $(1_t)_{t\in\mathcal{L}^\star}$, dont la bisimilarité coïncide cette fois-ci avec l'équivalence des traces, du moins dans les algèbres relationnelles propres : dans ce LTS, vérifier qu'une relation binaire et symétrique R est une bisimulation signifie vérifier que pour tous processus p, p', q, et toute trace t, si $p\ R\ q$ et $p\ 1_t\ p'$, alors il existe q' tel que $p'\ R\ q'$

4. Cette construction est générique, on la retrouvera à la section 4.2.1 du chapitre suivant.

et $q\ 1_t\ q'$; or, comme on a nécessairement $p = p'$ et comme le seul candidat pour q' est q, cela revient à vérifier que si p exhibe la trace t, alors q aussi.

Cette caractérisation de l'équivalence des traces nous permet finalement d'énoncer le résultat de Milner au niveau abstrait d'une algèbre relationnelle quelconque :

> « *Si pour toute étiquette* $\alpha \in \mathcal{L}$, $\xrightarrow{\alpha}$ *est déterministe modulo* $\sim$, *alors les bisimilarités fortes sur* $(\xrightarrow{\alpha})_{\alpha\in\mathcal{L}}$ *et* $(1_t)_{t\in\mathcal{L}^\star}$ *coïncident* ».

Nous projetons d'étudier si ce résultat peut de plus être *prouvé* dans une algèbre relationnelle quelconque : bien que la preuve concrète soit relativement simple, elle repose fortement sur la possibilité de manipuler les processus. Notons qu'il suffit d'étendre la proposition 3.33 : dans le cas faible, les conditions exprimées ici et à la section 4.5.2 pour simplifier l'étude du déterminisme modulo $\approx$ s'appliqueront encore.

3.3.3 Clôtures

Avant de passer à l'étude des techniques modulo pour les jeux à deux côtés, nous définissons une notion de *clôture*, qui nous permettra de représenter au niveau abstrait les techniques modulo contexte (ces dernières ne seront définies qu'au chapitre 5). Nous montrerons au chapitre 5 que les fonctions de fermeture par contexte, qui associent à toute relation la plus petite congruence la contenant, sont des clôtures (théorème 5.11) ; et qu'elles sont généralement compatibles avec les quatre génératrices étudiées ici ($\mathbf{s}, \mathbf{e}, \mathbf{p}, \mathbf{w}$). Dans la suite, les clôtures que nous manipulerons pourront donc toujours être pensées comme des fonctions de fermeture par contexte ; elles pourront aussi être ignorées : l'identité est une clôture compatible avec toute fonction.

Définition 3.38 (Clôture, congruence)**.** On appelle *clôture* toute fermeture symétrique $\mathcal{C}$ telle que :

$$\forall x, y \in X,\ \mathcal{C}(x \cdot y) \sqsubseteq \mathcal{C}(x) \cdot \mathcal{C}(y)\ . \tag{3.14}$$

Une *congruence vis-à-vis d'une clôture* $\mathcal{C}$ est une relation x close par $\mathcal{C}$, i.e., telle que $\mathcal{C}(x) = x$.

La notion de clôture renforce donc celle de fermeture, en prenant en compte le monoïde et l'opération de conversion. L'extensivité, l'idempotence et (3.14) nous serviront principalement à simplifier les calculs, lorsque l'on aura à prouver la symétrie de certaines fonctions témoins, ou à accorder de telles fonctions. Elles sont ainsi fortement exploitées dans les deux lemmes suivants, qui tentent de capturer une fois pour toutes une partie de ces calculs.

Lemme 3.39. *Pour toute relation $y \in X$ réflexive et transitive, et pour toute fonction $f \in X^{\langle X \rangle}$ croissante et extensive, on a*

$$(\forall x \in X,\ f(x \cdot y) \sqsubseteq f(x) \cdot y) \Rightarrow (f \mathbin{\hat{\cdot}} \widehat{y})^{\omega} = f^{\omega} \mathbin{\hat{\cdot}} \widehat{y} \ , \tag{3.15}$$

$$(\forall x \in X,\ f(y \cdot x) \sqsubseteq y \cdot f(x)) \Rightarrow (\widehat{y} \mathbin{\hat{\cdot}} f)^{\omega} = \widehat{y} \mathbin{\hat{\cdot}} f^{\omega} \ . \tag{3.16}$$

Démonstration. On ne démontre que (3.15), (3.16) étant similaire. Par une récurrence, on démontre tout d'abord $\forall n \in \mathbb{N},\ (f \mathbin{\hat{\cdot}} \widehat{y})^{(n)} \sqsubseteq f^{(n)} \mathbin{\hat{\cdot}} \widehat{y}$, d'où l'on déduit $(f \mathbin{\hat{\cdot}} \widehat{y})^{\omega} \sqsubseteq f^{\omega} \mathbin{\hat{\cdot}} \widehat{y}$. Pour l'inégalité réciproque, on a pour tout entier $n \in \mathbb{N}$, $f^{(n)} \mathbin{\hat{\cdot}} \widehat{y} \sqsubseteq (f \mathbin{\hat{\cdot}} \widehat{y})^{(n)} \mathbin{\hat{\cdot}} \widehat{y} \sqsubseteq (f \circ (f \mathbin{\hat{\cdot}} \widehat{y}))^{(n)} \mathbin{\hat{\cdot}} \widehat{y} = (f \mathbin{\hat{\cdot}} \widehat{y})^{(n+1)}$, en utilisant la réflexivité de y puis l'extensivité de f. Le résultat en découle. ■

Lemme 3.40. *Soient $\mathcal{C}$ une clôture, et $y \sqsubseteq z$ deux relations réflexives et transitives. Si $\mathcal{C}(z) \sqsubseteq z$, alors on a*

$$(\mathcal{C} \mathbin{\hat{\cdot}} \widehat{z})^{\omega} = \mathcal{C} \mathbin{\hat{\cdot}} \widehat{z} \tag{3.17}$$

$$\widehat{z} \mathbin{\hat{\cdot}} (\widehat{y} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{z})^{\omega} = \widehat{z} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{z} \tag{3.18}$$

Démonstration. (3.17) est un cas particulier de (3.15), puisque pour tout $x \in X$, on a : $\mathcal{C}(x \cdot z) \sqsubseteq \mathcal{C}(x) \cdot (z) \sqsubseteq \mathcal{C}(x) \cdot z$.

(3.18) Soit $f \triangleq \widehat{y} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{z}$. On applique tout d'abord (3.16), afin d'obtenir $\widehat{z} \mathbin{\hat{\cdot}} f^{\omega} = (\widehat{z} \mathbin{\hat{\cdot}} f)^{\omega}$: pour tout $x \in X$, on a

$$\begin{aligned} f(z \cdot x) = y \cdot \mathcal{C}(z \cdot x) \cdot z &\sqsubseteq y \cdot \mathcal{C}(z) \cdot \mathcal{C}(x) \cdot z \\ &\sqsubseteq y \cdot z \cdot \mathcal{C}(x) \cdot z = z \cdot \mathcal{C}(x) \cdot z = z \cdot f(x) \ . \end{aligned}$$

Comme $\widehat{z} \mathbin{\hat{\cdot}} f = \widehat{z} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{z}$, on peut appliquer la proposition 2.35(2), après avoir montré l'extensivité et l'idempotence de $\widehat{z} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{z}$. L'extensivité provient de la réflexivité de z et de l'extensivité de $\mathcal{C}$; pour l'idempotence, on utilise celle de $\mathcal{C}$ et la transitivité de z. ■

Nous pouvons finalement passer à l'étude des techniques modulo pour les différents jeux à deux côtés.

3.4 Techniques modulo pour les jeux de bisimulation standard

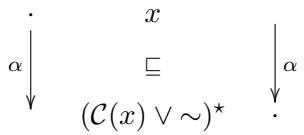
3.4.1 Pour la bisimulation forte

Les très bonnes propriétés de la simulation forte en termes de techniques modulo se retrouvent naturellement pour la bisimulation forte :

Théorème 3.41. *Soit $\mathcal{C}$ une clôture compatible avec* **s**. *La fonction $x \mapsto (\mathcal{C}(x) \vee \sim)^{\star}$ est compatible avec $\overleftrightarrow{\mathbf{s}}$.*

Démonstration. Par la proposition 2.50 et le corollaire 3.17(1), la fonction $\mathrm{id}^\star_X \circ (\mathcal{C} \vee \widehat{\sim})$ est compatible avec $\mathbf{s}$. Etant symétrique, elle est aussi compatible avec $\overline{\mathbf{s}}$, par la proposition 2.61(4). On conclut avec la proposition 2.56. ■

Par le corollaire 2.63, on peut énoncer ce résultat plus concrètement : pour montrer qu'une relation symétrique est contenue dans la bisimilarité forte, il suffit de vérifier qu'elle satisfait le diagramme suivant, où $\mathcal{C}$ est une clôture compatible avec $\mathbf{s}$.



A priori, on ne peut pas espérer mieux que cette technique ; le besoin ne s'en est toujours pas fait ressentir, en tout cas.

3.4.2 Pour la bisimulation faible, préordres contrôlés

La bisimilarité faible demande par contre plus de soin : modulo transitivité n'y étant pas supportée, il nous faut utiliser l'expansion, et éventuellement rompre la symétrie. On est ainsi contraint de faire certains calculs, pour vérifier que l'on peut toujours symétriser les fonctions témoins.

La proposition 3.42 ci-dessous récapitule les techniques plus ou moins standard [SM92, SW01] disponibles à ce stade :

- la première, « modulo expansion », s'exprime par une fonction compatible et symétrique ($\precsim$ dénote bien entendu la converse de $\succsim$), elle peut donc être utilisée directement.
- La restriction à l'expansion du côté droit, imposée par la technique précédente, n'est cependant pas nécessaire, comme l'indique la seconde technique. La fonction correspondant à cette dernière est compatible avec $\mathbf{w}$, mais pas avec $\overline{\mathbf{w}}$; pour que cette fonction corresponde bien à une technique modulo pour la bisimulation faible, il nous faut montrer qu'elle est correcte pour $\mathbf{w}$ via une fonction symétrique, afin de pouvoir utiliser le corollaire 2.63.
- La troisième technique s'exprime en changeant de génératrice, elle donne la possibilité d'utiliser la transitivité lors des offres visibles.

Proposition 3.42. *Soit $\mathcal{C}$ est une clôture compatible avec $\mathbf{w}$.*

1. *La fonction $x \mapsto \succsim \cdot \mathcal{C}(x) \cdot \precsim$ est compatible avec $\overleftrightarrow{\mathbf{w}}$.*
2. *La fonction $x \mapsto \succsim \cdot \mathcal{C}(x) \cdot \approx$ est correcte pour $\mathbf{w}$ via une fonction symétrique.*
3. *On a $\boldsymbol{\nu}\overleftrightarrow{\mathbf{w}_t} = \approx$, et la fonction $x \mapsto \succsim \cdot x^{=} \cdot \approx$ est correcte pour $\mathbf{w}_t$ via une fonction symétrique.*

Démonstration. 1. Posons $f_1 \triangleq \widehat{\succsim} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{\precsim}$; par les propositions 2.50, le lemme 3.19 et le corollaire 3.17(2),f_1 est compatible avec $\mathbf{w}$ (on a bien $\succsim \in X_{\mathbf{e}}$ et $\precsim \in X_{\mathbf{w}}$). Etant symétrique, f_1 est aussi compatible avec $\overline{\mathbf{w}}$, d'où le résultat, par la proposition 2.56.

2. Posons $f_2 \triangleq \widehat{\succsim} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{\approx}$; par les propositions 2.50, le lemme 3.19 et le corollaire 3.17(2),f_2 est compatible avec $\mathbf{w}$, puis correcte pour $\mathbf{w}$ via f_2^ω. Par le lemme 2.58, elle est aussi correcte via $\widehat{\approx} \mathbin{\hat{\cdot}} f_2^\omega$, dont il nous faut montrer la symétrie.

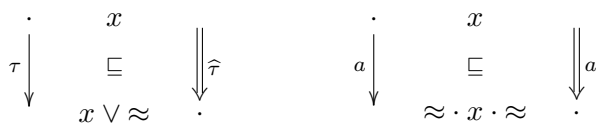
   Par la proposition 2.53, on a $\mathcal{C}(\approx) \sqsubseteq \approx$. On peut donc appliquer (3.18) pour simplifier la fonction témoin en une fonction explicitement symétrique : $\widehat{\approx} \mathbin{\hat{\cdot}} f_2^\omega = \widehat{\approx} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{\approx}$.

3. Par la proposition 3.22, si x est une $\overleftrightarrow{\mathbf{w}_\mathrm{t}}$-simulation, alors $x^\star$ est une $\overleftrightarrow{\mathbf{w}}$-simulation ; inversement, toute $\overleftrightarrow{\mathbf{w}}$-simulation est une $\overleftrightarrow{\mathbf{w}_\mathrm{t}}$-simulation. Pour le second point, on raisonne comme précédemment, en utilisant le lemme 3.23. ■

D'un point de vue technique, remarquons qu'il n'est pas nécessaire que $\mathcal{C}$ soit une clôture dans le premier point (il suffit que ce soit une fonction symétrique) : la modularité de la notion de compatibilité permet de composer très simplement l'ensemble des techniques dont nous disposons. Au contraire, dans le second point, le fait que l'on sorte du cadre des fonctions compatibles nous force à prouver la symétrie d'une fonction relativement complexe, ce qui nécessite les hypothèses que l'on fait sur les clôtures.

Notons aussi que la troisième technique établie par cette proposition peut naturellement être enrichie par l'utilisation d'une clôture compatible avec $\mathbf{w}_\mathrm{t}$. Nous verrons cependant au chapitre 5 que si les fonctions standard de fermeture par contexte sont généralement compatibles avec $\mathbf{w}$, elles ne le sont pas avec $\mathbf{w}_\mathrm{t}$. Il serait donc vain de donner ici l'illusion de cette possibilité… Cette troisième technique admet pour corollaire la technique « bisimulation faible modulo $\approx$ lors des offres visibles », que l'on utilisera au chapitre 6 (lemme 6.47) :

Corollaire 3.43. *Toute relation symétrique satisfaisant les deux diagrammes ci-dessous est contenue dans la bisimilarité faible :*



Nous ne donnons pas ici les diagrammes auxquels correspondent les autres techniques : il sont donnés plus loin dans un cadre plus général.

Dépasser l'expansion : préordres contrôlés

L'expansion est la technique standard pour simplifier les preuves de bisimulation faible. Elle ne pouvait cependant pas être utilisée dans le cadre de la GcPan [HPS05], ce qui nous a forcés à donner une preuve fastidieuse, puis qui nous a incités à rechercher de nouvelles techniques : on verra au chapitre suivant d'autres façons de contraindre une relation, afin d'autoriser son utilisation à la place de l'expansion lorsque cette dernière est trop restrictive. On prépare maintenant le terrain pour le chapitre suivant, en introduisant la notion de *préordre contrôlé*, dont le but est de remplacer l'expansion. Cela nous permet de factoriser pour le chapitre suivant toute la partie consistant à combiner les techniques que l'on obtiendra aux techniques plus standard (modulo contexte, réflexivité, et bisimilarité faible à droite)

Définition 3.44 (Relation, préordre contrôlé)**.** On appelle *relation contrôlée* toute relation $\triangleright \in X$ telle que la pré-composition par $\triangleright^\star$ ($\widehat{\triangleright^\star} \cdot \mathrm{id}_X : x \mapsto \triangleright^\star \cdot x$) soit correcte pour $\mathbf{w}_\mathrm{t}$ via $(\mathrm{id}_X \vee \widehat{\approx})^\star$.

Un *préordre contrôlé* est une relation réflexive, transitive et contrôlée.

Cette définition est relativement ad hoc : on y demande ce qui est nécessaire pour que la pré-composition par un préordre contrôlé soit correcte pour la bisimulation faible (la fonction témoin est notamment symétrique, et suffisamment « expressive » pour que des techniques impliquant des témoins complexes puissent être prises en compte). Notons aussi que c'est la correction pour $\mathbf{w}_\mathrm{t}$ qui est exigée : elle entraîne la correction pour $\mathbf{w}$ par le lemme 3.24.

Le lemme suivant nous permettra d'utiliser principalement la notion de préordre contrôlé ; sa preuve est triviale. La notion intermédiaire de relation contrôlée ne sera utile que pour pouvoir distinguer un peu plus tard deux problèmes orthogonaux liés à l'union de telles relations.

Lemme 3.45. *Une relation* ($\triangleright$) *est contrôlée si et seulement si sa fermeture réflexive transitive* ($\triangleright^\star$) *est un préordre contrôlé.*

La proposition suivante montre que cette notion capture l'expansion ; on retrouve donc en particulier dans le théorème qui suit les deux derniers points de la proposition 3.42.

Proposition 3.46. *L'expansion est un préordre contrôlé.*

Démonstration. Par le lemme 3.23, la fonction $x \mapsto \succsim \cdot x$ est compatible avec $\mathbf{w}_\mathrm{t}$, puis correcte pour $\mathbf{w}_\mathrm{t}$ via elle-même par la proposition 2.49 (c'est en fait une fermeture, et son itération ne la modifie pas, par la proposition 2.35(2)). On en déduit sa correction pour $\mathbf{w}_\mathrm{t}$ via $(\mathrm{id}_X \vee \widehat{\approx})^\star$ en la composant par cette dernière fonction, à l'aide du lemme 2.58, puis en utilisant (2.21) : $(\widehat{\succsim} \mathbin{\hat{\cdot}} \mathrm{id}_X \vee \widehat{\approx})^\star = (\mathrm{id}_X \vee \widehat{\approx})^\star$. ■

Théorème 3.47. *Soit $\triangleright$ un préordre contrôlé.*

1. *Pour toute clôture $\mathcal{C}$ compatible avec $\mathbf{w}$ telle que $\mathcal{C}(\triangleright) \sqsubseteq \triangleright$, la fonction $x \mapsto \triangleright \cdot \mathcal{C}(x) \cdot \approx$ est correcte pour $\mathbf{w}$ via $(\mathcal{C} \vee \widehat{\approx})^\star$.*
2. *La fonction $x \mapsto \triangleright \cdot x^= \cdot \approx$ est correcte pour $\mathbf{w}_t$ via $(\mathrm{id}_X \vee \widehat{\approx})^\star$.*

Démonstration. Nous ne démontrons que le premier point, le second étant similaire, et légèrement plus simple.

Soit $f : x \mapsto \mathcal{C}(x) \cdot \approx$, et $g : x \mapsto \triangleright \cdot x$; f est compatible avec $\mathbf{w}$, et g est correcte pour $\mathbf{w}$ via $(\mathrm{id}_X \vee \widehat{\approx})^\star$ (en utilisant le lemme 3.24). On peut donc appliquer théorème 2.52, après avoir vérifié la compatibilité de f avec g : pour tout $x \in X$, on a

$$\begin{aligned}(f \circ g)(x) &\triangleq \mathcal{C}(\triangleright \cdot x) \cdot \approx \\ &\sqsubseteq \mathcal{C}(\triangleright) \cdot \mathcal{C}(x) \cdot \approx && \text{(par (3.14))} \\ &\sqsubseteq \triangleright \cdot \mathcal{C}(x) \cdot \approx \triangleq (g \circ f)(x) \ . && \text{(hypothèse entre } \mathcal{C} \text{ et } \triangleright)\end{aligned}$$

Il reste alors à simplifier la fonction témoin obtenue, $(f^\omega \vee \widehat{\approx})^\star$. Par la proposition 2.53, on a $\mathcal{C}(\approx) \sqsubseteq \approx$, de telle sorte que (3.17) nous donne $f^\omega = f$. On en déduit $(f^\omega \vee \widehat{\approx})^\star = (\mathcal{C} \mathbin{\hat{:}} \widehat{\approx} \vee \widehat{\approx})^\star$, et on conclut par (2.21). ■

Remarque 3.48. Notons que le premier point de ce théorème ne généralise pas tout à fait la proposition 3.42(2) : l'hypothèse de congruence $(\mathcal{C}(\succsim) \sqsubseteq \succsim)$ n'est pas requise dans cette proposition. Cet artifice technique est dû au fait que l'on oublie en passant par la notion de préordre contrôlé les bonnes propriétés de l'expansion : la pré-composition par l'expansion n'est pas seulement correcte pour $\mathbf{w}$, elle est compatible avec $\mathbf{w}$.

Les deux corollaires ci-dessous donnent de façon plus concrète les conséquences directes du théorème 3.47.

Corollaire 3.49. *Soient $\triangleright$ un préordre contrôlé, et $\mathcal{C}$ une clôture compatible avec $\mathbf{w}$ telle que $\mathcal{C}(\triangleright) \sqsubseteq \triangleright$. Toute relation symétrique $x \in X$ satisfaisant le diagramme ci-dessous est majorée par la bisimilarité faible.*

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ & \triangleright \cdot \mathcal{C}(x) \cdot \approx & \cdot \end{array}$$

Corollaire 3.50. *Soit $\triangleright$ un préordre contrôlé. Toute relation $x \in X$ symétrique et satisfaisant les deux diagrammes ci-dessous est majorée par la bisimilarité faible.*

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\tau}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle\widehat{\tau}} \\ & \triangleright \cdot x^= \cdot \approx & \cdot \end{array} \qquad \begin{array}{ccc} \cdot & x & \\ {\scriptstyle a}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle a} \\ & (x \vee \approx)^\star & \cdot \end{array}$$

Nous avons ici présenté les relations contrôlées comme un cas particulier de fonctions correctes ; la condition de « congruence » imposée par le théorème 3.47 ($\mathcal{C}(\triangleright) \sqsubseteq \triangleright$) apparaît ainsi comme une conséquence de la condition de compatibilité imposée par le théorème 2.52 pour la composition d'une fonction compatible avec une fonction correcte. « Historiquement », nous avons cependant introduit cette idée de relation contrôlée dès notre premier article sur les techniques modulo [Pou05b], de manière directe (et très ad hoc), sans passer par la notion de correction via, que nous n'avions pas encore. De même, nous donnions déjà la condition de congruence permettant de combiner un préordre contrôlé à des techniques modulo contexte. C'est en partant d'une idée de Tom Hirschowitz que nous avons tenté de généraliser ces notions en termes de fonctions, et que nous avons abouti au théorème 2.52.

Unions de relations contrôlées

On verra au chapitre suivant (contre-exemple (4.21), section 4.3.3) que l'union de deux relations contrôlées n'est pas nécessairement contrôlée [5]. Ce problème provient naturellement du fait que cette notion s'appuie sur celle de fonction correcte plutôt que celle de fonction compatible. Bien que le théorème 3.47 permette de combiner l'utilisation d'un préordre contrôlé à d'autres techniques modulo pour la bisimulation faible, une fois le préordre contrôlé fixé, on ne peut donc pour l'instant rien rajouter « à gauche », en position de transitivité.

On peut cependant vouloir rajouter à cette place des équivalences relativement fortes (de la congruence structurelle par exemple), qui sont toujours « très bien supportées » en tant que technique modulo. La proposition suivante permet justement cela (bien que la relation $\equiv$ soit supposée être une expansion, elle peut être pensée comme de la congruence structurelle, formant par exemple une bisimulation forte) :

Théorème 3.51. *Soit $\triangleright$ une relation contrôlée, et $\equiv$ une expansion ($\equiv \in X_{\mathbf{e}\wedge\overline{\mathbf{w}}}$). Si $\equiv \cdot \triangleright \sqsubseteq \triangleright \cdot \equiv$, alors $(\triangleright \vee \equiv)$ est une relation contrôlée.*

Démonstration. Par une simple récurrence, on démontre tout d'abord que l'hypothèse de commutation entraîne

$$(\triangleright \vee \equiv)^\star \sqsubseteq \triangleright^\star \cdot \equiv^\star. \qquad (*)$$

Par le lemme 2.47, il suffit donc de montrer que la fonction $x \mapsto \triangleright^\star \cdot \equiv^\star \cdot x$ est correcte pour $\mathbf{w}_\mathrm{t}$ via $(\mathrm{id}_X \vee \widehat{\approx})^\star$.

5. L'union de deux préordres x, y n'est pas nécessairement un préordre : il faut en prendre la fermeture transitive, $(x \vee y)^+$. Il n'est donc pas surprenant que l'union de deux **préordres** contrôlés n'en soit pas un. Ce n'est cependant pas ce point de détail qui est problématique, puisque le lemme 3.45 nous autorise justement à considérer la fermeture réflexive transitive de cette union.

On applique pour cela le théorème 2.52 à la fonction $f : x \mapsto \equiv^\star \cdot x$ (qui est compatible avec $\mathbf{w}_\mathrm{t}$ par le lemme 3.23) et à la fonction $g : x \mapsto \triangleright^\star \cdot x$ (qui est correcte pour $\mathbf{w}_\mathrm{t}$ via $(\mathrm{id}_X \vee \widehat{\approx})^\star$ par hypothèse). Il nous faut pour cela vérifier la compatibilité de la première avec la seconde : pour tout x, on a

$$\begin{aligned}(f \circ g)(x) &= \equiv^\star \cdot \triangleright^\star \cdot x \\ &\sqsubseteq \triangleright^\star \cdot \equiv^\star \cdot x = (g \circ f)(x) && \text{(par } (*)\text{)}\end{aligned}$$

On conclut en simplifiant la fonction témoin obtenue :

$$\begin{aligned}(\mathrm{id}_X \vee \widehat{\approx})^\star \circ f^\omega &= (\mathrm{id}_X \vee \widehat{\approx})^\star \circ f && \text{(proposition 2.35(2))} \\ &= (\widehat{\equiv^\star} \mathbin{\hat{\cdot}} \mathrm{id}_X \vee \widehat{\approx})^\star = (\mathrm{id}_X \vee \widehat{\approx})^\star && \text{(par (2.21))}\end{aligned}$$

■

L'hypothèse de commutation que l'on fait entre $\triangleright$ et $\equiv$ est cruciale, et provient naturellement de l'hypothèse de compatibilité du théorème 2.52, sur laquelle repose la preuve de cette proposition : les relations $\equiv$ et $\triangleright$ correspondent respectivement à une fonction compatible et à une fonction correcte. Intuitivement cette hypothèse permet de s'assurer que l'argument permettant d'obtenir le contrôle de $\triangleright$ est « robuste » vis-à-vis de la relation de congruence structurelle utilisée (cf. contre-exemple (4.21) et remarque 4.38 au chapitre suivant).

Enfin, la fonction témoin exhibée par le théorème 3.47 étant indépendante du préordre contrôlé considéré, le théorème 2.57 nous permet d'utiliser deux préordres contrôlés : l'un pour la partie de gauche à droite du jeu de bisimulation, et l'autre pour la partie de droite à gauche. On obtient ainsi l'adaptation suivante du corollaire 3.49 (le corollaire 3.50 peut naturellement s'adapter de la même façon).

Corollaire 3.52. *Soient $\triangleright$, $\blacktriangleright$ deux préordres contrôlés, et $\mathcal{C}$ une clôture compatible avec $\mathbf{w}$. Si $\triangleright$ et $\blacktriangleright$ sont clos par $\mathcal{C}$, alors toute relation $x \in X$ satisfaisant les deux diagrammes ci-dessous est majorée par la bisimilarité faible.*

Notons que ce corollaire ne serait pas nécessaire si l'union de deux relations contrôlées restait contrôlée.

3.4.3 Pour l'expansion

Nous reviendrons sur les préordres contrôlés au chapitre suivant ; nous finissons maintenant notre exploration des techniques standard. Les techniques modulo pour la bisimulation faible faisant appel à l'expansion, il peut être utile d'avoir des techniques modulo pour ce préordre. Le théorème 2.57 nous permet d'en obtenir facilement, à partir des techniques modulo disponibles pour la simulation faible et pour la pré-expansion.

L'expansion étant un jeu asymétrique, il est moins aisé d'en exprimer les techniques modulo : on doit spécifier une fonction pour chaque côté du jeu. Le théorème ci-dessous est donc donné directement sous sa forme concrète. De gauche à droite, la transitivité est supportée, mais pas de droite à gauche, puisque l'on retrouve un jeu de simulation faible. De ce côté du jeu on utilise la bi-expansion, comme le propose Cédric Fournet [Fou98, chapitre 4] ; en utilisant notre théorie, on peut combiner automatiquement ce résultat aux techniques modulo contexte (ce que Fournet ne fait pas) :

Théorème 3.53. *Soient $\mathcal{C}$ une clôture compatible avec* $\mathbf{e}$ *et* $\mathbf{w}$*, et $x \in X$ une relation. Si x satisfait les deux diagrammes suivants, alors $x \sqsubseteq \gtrsim$.*

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \downarrow{\scriptstyle\widehat{\alpha}} \\ & (\mathcal{C}(x) \vee \gtrsim)^\star & \cdot \end{array} \qquad \begin{array}{ccc} & x & \cdot \\ {\scriptstyle\widehat{\alpha}}\Downarrow & \sqsupseteq & \downarrow{\scriptstyle\alpha} \\ \cdot & \gtrsim \cdot \mathcal{C}(x) \cdot \asymp & \end{array}$$

Démonstration. Soient $f_e = (\mathcal{C} \vee \widehat{\gtrsim})^\star$, et $f_w = \widehat{\asymp} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{\precsim}$. f_e est compatible avec $\mathbf{e}$, et donc correcte pour $\mathbf{e}$ via f_e^ω. f_w est compatible avec $\mathbf{w}$ (on a $\asymp \in X_{\mathbf{e}}$ et $\precsim \in X_{\mathbf{w}}$), et donc correcte pour $\mathbf{w}$ via f_w^ω. Par le lemme 2.58, on obtient ensuite la correction de $\overline{f_w}$ pour $\overline{\mathbf{w}}$ via $f' = (\mathrm{id}_X \vee \widehat{\gtrsim})^\star \circ \overline{f_w}^\omega$.

On vérifie alors que $f' = f_e^\omega$:

$$\begin{aligned} f' = (\overline{f_w}^\omega \vee \widehat{\gtrsim})^\star &= (\overline{f_w}^\omega \mathbin{\hat{\cdot}} \widehat{\gtrsim} \vee \widehat{\gtrsim})^\star && \text{(par (2.21))} \\ &= (\widehat{\gtrsim} \mathbin{\hat{\cdot}} \mathcal{C} \mathbin{\hat{\cdot}} \widehat{\gtrsim} \vee \widehat{\gtrsim})^\star && \text{(par (3.18))} \\ &= (\mathcal{C} \vee \widehat{\gtrsim})^\star = f_e && \text{(par (2.21))} \\ &= f_e^\omega && \text{(proposition 2.35(2))} \end{aligned}$$

(pour la seconde étape, on a $\mathcal{C}(\gtrsim) \sqsubseteq \gtrsim$ car $\mathcal{C}$ est compatible avec $\mathbf{e}$ et $\mathbf{w}$: proposition 2.53 ; cette inégalité est aussi requise pour prouver l'idempotence de f_e dans la dernière étape). On peut finalement appliquer le théorème 2.57, et obtenir ainsi $\boldsymbol{\nu}\left(\mathbf{e} \circ f_e \wedge \overline{\mathbf{w}} \circ \overline{f_w}\right) = \gtrsim$, dont découle le résultat énoncé. ■

Cette technique sera utilisée deux fois au chapitre 6 (proposition 6.19 et lemme 6.21) ; notons que les deux hypothèses faites sur la clôture $\mathcal{C}$ sont indépendantes : la compatibilité avec $\mathbf{e}$ n'entraîne pas nécessairement la compatibilité avec $\mathbf{w}$; on verra au chapitre 5 que ces hypothèses sont réalistes.

Remarquons aussi que l'on pourrait autoriser la transitivité lors des offres visibles du côté droit, en utilisant la génératrice $\overline{\mathbf{w}_t}$ plutôt que $\overline{\mathbf{w}}$.

Techniques modulo pour la bi-expansion

Les techniques modulo pour l'expansion font donc apparaître la nécessité d'étudier celles pour la bi-expansion… Cette descente infernale s'arrête cependant ici, la bi-expansion héritant naturellement des bonnes propriétés de la pré-expansion (corollaire 3.17(1)) : pour toute clôture $\mathcal{C}$ compatible avec $\mathbf{e}$, la fonction $x \mapsto (\mathcal{C}(x) \vee \asymp)^\star$ est compatible avec $\overleftrightarrow{\mathbf{e}}$:

Théorème 3.54. *Soit $\mathcal{C}$ une clôture compatible avec $\mathbf{e}$. Toute relation symétrique $x \in X$ satisfaisant le diagramme ci-dessous est majorée par la bi-expansion $(\asymp)$.*

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\alpha}\downarrow & \sqsubseteq & \downarrow{\scriptstyle\widehat{\alpha}} \\ & (\mathcal{C}(x) \vee \asymp)^\star & \cdot \end{array}$$

On utilisera cette dernière technique au chapitre 6 (lemme 6.33), mais en tant que technique de preuve pour l'expansion (rappelons que $\asymp \sqsubseteq \gtrsim$) : en renforçant le résultat que l'on souhaite prouver de façon co-inductive, on accède à des techniques modulo plus puissantes. Cela n'est pas sans rappeler le compromis que l'on a à faire lors de la définition du prédicat à utiliser pour une preuve par induction : si l'on renforce ce prédicat, on doit travailler davantage pour prouver l'étape inductive, mais l'hypothèse d'induction est plus informative.

Remarque 3.55 (Symétrie et accordage des fonctions témoins). L'un des points clef pour passer aux techniques modulo pour les jeux à deux côtés consiste à obtenir une fonction témoin symétrique lorsque l'on considère des équivalences (théorème 2.62), ou à « accorder » les fonctions témoins lorsque l'on considère des préordres comme l'expansion (théorème 2.57, notons que prouver la symétrie d'une fonction f revient à accorder f et $\overline{f}$).

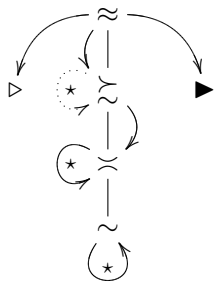
Sans cela, on pourrait oublier totalement les fonctions témoins, et ne parler que de fonctions correctes ; les calculs parfois pénibles que l'on a dû faire dans cette section ne seraient ainsi plus nécessaires (cf. section 3.3.2, sur la 2-similarité).

Ce point est pourtant essentiel, et peut être mis à profit pour obtenir des techniques modulo de manière déductive : partons de la fonction $f : x \mapsto x_e \cdot x \cdot x_w$, qui est compatible avec $\mathbf{w}$ dès que $x_e \in X_{\mathbf{e}}$ et $x_w \in X_{\mathbf{w}}$; pour rendre la fonction témoin (f^ω) symétrique, il va falloir égaliser x_e et $\overline{x_w}$, c'est-à-dire considérer un élément dans $X_{\mathbf{e}} \cap X_{\overline{\mathbf{w}}}$; on retombe ainsi naturellement sur l'expansion. De même, lorsque l'on cherche de cette manière des techniques modulo pour l'expansion, on tombe naturellement sur la bi-expansion.

3.4.4 Récapitulatif

Nous avons finalement retrouvé de façon uniforme les techniques plus ou moins standard associées à la bisimulation forte, la bisimulation faible, l'expansion et la bi-expansion, et nous avons introduit la notion de préordre contrôlé, qui nous permettra d'aller plus loin que l'expansion au chapitre suivant.

Le graphe ci-dessous tente de récapituler de manière intuitive l'ensemble de ces techniques : la bisimilarité forte et la bi-expansion supportent la technique modulo transitivité ; l'expansion ne la supporte que d'un coté, pour l'autre, il faut utiliser modulo bi-expansion ; et la bisimilarité faible supporte les techniques « modulo préordre contrôlé », dont entre autres, modulo expansion.



Aux techniques mentionnées sur ce graphe peuvent toujours s'ajouter, au choix, soit la transitivité lors des offres visibles, soit les techniques modulo contextes que nous obtiendrons au chapitre 5. Nous compléterons ce graphe au chapitre suivant (figure 4.1), après avoir étudié les propriétés de l'élaboration en termes de techniques modulo.

Chapitre 4

Terminaison et commutation

Lorsque l'on ne considère que les actions silencieuses, on s'aperçoit que l'invalidité de la technique « modulo transitivité » dans le cas de la simulation faible correspond exactement à la différence qu'il existe, dans le domaine de la réécriture [TeR03], entre les notions de *confluence locale* et de *confluence* (propriété de Church–Rosser). Dans ce domaine, une réponse standard à ce problème est le « lemme de Newman », qui fait appel à une hypothèse de terminaison, et à la technique de preuve par induction bien fondée qui va de pair. Nous montrons dans ce chapitre comment utiliser de telles méthodes afin de définir de nouvelles techniques modulo pour la bisimulation.

Dans un premier temps, nous nous plaçons au niveau des relations binaires, pour mettre en évidence les liens entre ces notions de réécriture et la bisimilarité faible, puis nous prouvons une généralisation du lemme de Newman, qui affaiblit l'hypothèse de terminaison requise par ce lemme. Nous ne savons pas si cette généralisation peut être utile dans le cadre de la réécriture ; elle est par contre essentielle lorsque l'on passe à la bisimulation : une extension directe du lemme de Newman pour la bisimulation engendrerait une contrainte de terminaison trop forte pour être vraiment utile en pratique.

Avant de déduire des techniques modulo pour la bisimulation de ce résultat de réécriture, nous utilisons les résultats de Doornbos, Backhouse et van der Woude [DBvdW97] pour re-démontrer cette généralisation dans une algèbre relationnelle quelconque. Cela nous permet d'une part de gagner en généralité, mais surtout, suivant les motivations des auteurs de [DBvdW97], de donner des preuves, qui, une fois acquises les méthodes de raisonnement inhérentes à cette présentation algébrique, nous semblent beaucoup plus claires : fermer des diagrammes de commutation et faire des inductions bien fondées y revient à appliquer de simples règles de calcul, afin de simplifier des expressions.

L'inconvénient majeur des techniques reposant sur des hypothèses de terminaison est leur manque de modularité, qui provient de la non-modularité de la notion de terminaison elle-même : l'union de deux relations qui ter-

minent ne termine pas nécessairement [BD86, Ges90]. La combinaison de techniques à base de terminaison, ou l'extension d'une preuve utilisant une telle technique, peut donc nécessiter de prouver à nouveau les résultats de terminaison qui avaient été initialement impliqués. Nous montrons comment utiliser le théorème 2.81 du premier chapitre, pour pouvoir combiner ce genre de techniques avec des techniques plus standard.

De façon assez surprenante, l'élaboration ($\gtrapprox$) permet aussi de résoudre ce problème de modularité, lorsque les transitions silencieuses du système considéré terminent : ce préordre permet de déduire de cette hypothèse de terminaison globale (elle n'est prouvée qu'une fois pour toutes, pour chaque système à considérer) celle qui permet d'obtenir la technique modulo transitivité par les résultats précédents. Dans de tels LTS, dits *réactifs*, l'élaboration est finalement un préordre relativement proche de la bisimilarité faible, qui bénéficie de techniques modulo aussi développées que celles de la bisimilarité forte. Nous montrerons aussi que cette hypothèse de terminaison permet d'obtenir des méthodes puissantes pour obtenir des propriétés de τ-inertie ou de déterminisme faible.

4.1 Terminaison et commutation en réécriture

Roger Hindley [Hin64], Barry K. Rosen [Ros70], Gérard Huet [Hue80], Nachum Bachmair et Leo Dershowitz [BD86], Françoise Bellegarde et Pierre Lescanne [BL87, BL90], Alfons Geser [Ges90] et Vincent van Oostrom [vO94] (entre autres) ont largement étudié les liens étroits entre commutation et terminaison, ainsi que leurs diverses applications à la réécriture. Le livre édité par Marc Bezem, Jan Willem Klop et Roel de Vrijer [TeR03] est devenu la référence dans le domaine de la réécriture.

On entend généralement par *réécriture* la réécriture de *termes* concrets, formés à partir de symboles de fonctions : les « Term Rewriting Systems » (TRS) y sont l'objet d'étude. Les propriétés qui nous intéressent (confluence, commutation, terminaison) sont cependant généralement étudiées au niveau un peu plus abstrait des « Abstract Rewriting Systems » (ARS), où la nature des objets que l'on réécrit importe peu. Dans les sections suivantes, nous étudierons ces notions dans le cadre encore plus général des algèbres relationnelles, où la nature même des relations (et donc des relations de réécriture) est abstraite. Dans cette section introductive, nous en restons cependant au niveau d'abstraction intermédiaire, en se plaçant dans l'algèbre relationnelle propre d'un ensemble quelconque $\mathcal{P}$. Les éléments de $\mathcal{P}$ (appelés *processus*, ou *termes*, en réécriture) seront notés p, q ; comme précédemment, les relations binaires seront notées R, S.

4.1.1 Lemme de Newman

Le lemme de Newman (Max [New42]) concerne les graphes, et s'énonce généralement ainsi :

> *« Toute graphe localement confluent et sans chemin infini est confluent ».*

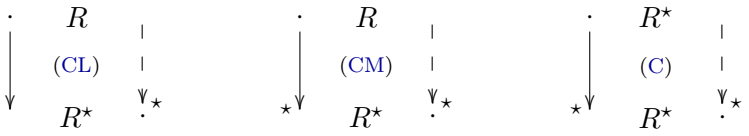
Ce sera cependant sa version extensionnelle (où l'on considère des relations plutôt que des graphes), étendue aux propriétés de commutation qui va nous intéresser. Nous rappelons ces diverses propriétés ci-dessous :

Définition 4.1 (Commutation, Confluence)**.** Soient $R, \to \in$ deux relations binaires sur $\mathcal{P}$.

R et $\to$ *commutent*,	si	$\leftarrow^\star \cdot R^\star \subseteq R^\star \cdot \leftarrow^\star$	(C)
R et $\to$ *commutent à moitié*,	si	$\leftarrow^\star \cdot R \subseteq R^\star \cdot \leftarrow^\star$	(CM)
R et $\to$ *commutent localement*,	si	$\leftarrow \cdot R \subseteq R^\star \cdot \leftarrow^\star$	(CL)

La relation $\to$ est *confluente* (resp. *à moitié confluente*, *localement confluente*) si $\leftarrow$ et $\to$ commutent (resp. commutent à moitié, localement).

Les diagrammes ci-dessous illustrent les trois propriétés de commutation ; on obtient naturellement une définition moins algébrique en déroulant la définition de la composition relationnelle ; par exemple, R et $\to$ commutent localement si pour tous $p, p', q \in \mathcal{P}$ tels que $p \to p'$ et $p\ R\ q$, il existe $q' \in \mathcal{P}$ tel que $q \to^\star q'$ et $p'\ R^\star\ q'$.



Parmi ces différentes notions, c'est généralement la commutation (ou la confluence) qui exprime la propriété utile en pratique ; c'est cependant la propriété la plus délicate à prouver, puisque l'on a affaire à deux quantifications universelles sur deux séquences quelconques de transitions. Au contraire, la commutation locale ne quantifie universellement que deux transitions ; il est donc a priori plus facile de considérer l'ensemble des cas possibles (par exemple, dans le cas fréquent où R est finie, et $\to$ à branchement fini, il n'y a qu'un nombre fini de cas à considérer). La commutation à moitié est intermédiaire : elle quantifie universellement sur une transition d'un côté, et sur une séquence de l'autre ; elle équivaut cependant à la commutation :

Lemme 4.2. *Les propriétés* (CM) *et* (C) *sont équivalentes, et impliquent* (CL).

Démonstration. Les inclusions (C) ⇒ (CM) ⇒ (CL) sont immédiates. L'implication restante, (CM) ⇒ (C), est une instance de (2.20). ■

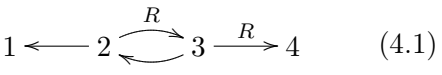
Remarque 4.3. R et $\to$ jouent des rôles symétriques dans les définitions de commutation et commutation locale : R et $\to$ commutent si et seulement si

$$\leftarrow^\star \cdot R^\star \subseteq R^\star \cdot \leftarrow^\star \quad \Leftrightarrow \quad \overline{\leftarrow^\star \cdot R^\star} \subseteq \overline{R^\star \cdot \leftarrow^\star} \quad \Leftrightarrow \quad \overline{R}^\star \cdot \to^\star \subseteq \overline{R}^\star \cdot \to^\star$$

i.e. si et seulement si $\to$ et R commutent. Bien que la définition de commutation à moitié soit au contraire asymétrique, l'équivalence donnée par le lemme précédent justifie a posteriori le rôle symétrique que joue les deux relations dans notre terminologie.

L'implication manquante ((CL) $\Rightarrow$ (C)) n'est pas toujours vraie, le contre-exemple standard étant rappelé ci-dessous :

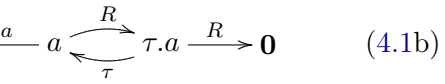
$$R \triangleq \{\langle 2,3\rangle,\ \langle 3,4\rangle\} \qquad \to \triangleq \{\langle 3,2\rangle,\ \langle 2,1\rangle\}$$



(4.1)

R et $\to$ commutent localement, mais ne commutent pas. Il suffit de considérer le contre-exemple (1.18) de l'introduction pour s'apercevoir que l'on a exactement le même problème : la relation R ci-dessous est une simulation faible modulo transitivité sans être contenue dans la similarité faible.

$$R \triangleq \{\langle a, \tau.a\rangle,\ \langle \tau.a, \mathbf{0}\rangle\}$$



(4.1b)

Le lemme de Newman utilise pour contourner cette irrégularité une hypothèse de terminaison ; plus précisément on y raisonne par induction bien fondée, en s'appuyant sur une hypothèse de terminaison. Nous commençons donc par introduire la notion suivante :

Définition 4.4 (Support de l'induction)**.** Soit $\prec$ une relation binaire sur $\mathcal{P}$. On définit inductivement l'ensemble des éléments *accessibles par* $\prec$, comme le plus petit ensemble $A_\prec$ tel que :

$$\forall p \in \mathcal{P},\ (\forall q \in \mathcal{P},\ q \prec p \Rightarrow q \in A_\prec) \Rightarrow p \in A_\prec\ . \tag{4.2}$$

La relation $\prec$ *supporte l'induction* si $A_\prec = \mathcal{P}$.

Bien que la notion soit standard, le terme *support de l'induction* l'est moins : la définition précédente correspond à la version constructive de la notion de *bonne fondation* (une relation est bien fondée si toute partie non vide admet un élément minimal), sur laquelle nous reviendrons page 130. Nous utilisons cette terminologie en prévision des sections suivantes, où cette notion sera généralisée au cadre abstrait des algèbres relationnelles. Le principe d'induction bien fondée n'est qu'une redite de la définition :

Théorème 4.5 (Principe d'induction bien fondée)**.** *Soit $\langle \mathcal{P}, \prec\rangle$ un ensemble muni d'une relation binaire. $\prec$ supporte l'induction si et seulement si pour tout prédicat $\varphi(\cdot)$ sur $\mathcal{P}$, on a*

$$\textit{Si } \forall p \in \mathcal{P},\ (\forall q \in \mathcal{P},\ q \prec p \Rightarrow \varphi(q)) \Rightarrow \varphi(p),\ \textit{alors } \forall p \in \mathcal{P},\ \varphi(p)\ .$$

Démonstration. Il s'agit exactement du principe d'induction associé à la définition 4.4, exprimé en utilisant des prédicats plutôt que des ensembles. ■

La façon la plus courante de prouver qu'une relation supporte l'induction consiste à passer par la notion suivante :

Définition 4.6 (Relation fortement normalisante)**.** Une relation binaire $\succ$ sur $\mathcal{P}$ est *fortement normalisante* s'il n'existe pas de suite $(p_i)_{i\in\mathbb{N}}$ d'éléments de $\mathcal{P}$ telle que $\forall i \in \mathbb{N},\ p_i \succ p_{i+1}$.

Dans le cadre de la sémantique des programmes, cette notion est très intuitive : lorsque R correspond aux étapes de réduction d'un programme, la forte normalisation assure l'absence de divergences : le programme finit toujours par s'arrêter – ainsi, nous parlerons parfois simplement de *terminaison*. Cette notion a été largement étudiée, et de nombreux outils existent, qui permettent de prouver de façon plus ou moins automatique des résultats de normalisation forte.

La forte normalisation et le support de l'induction sont en fait presque la même notion : il suffit de renverser la relation. On peut donc raisonner par induction bien fondée sur toute relation fortement normalisante, à condition de la considérer dans le bon sens. Dans la suite, nous noterons $\succ$ la relation converse de toute relation notée $\prec$ (qu'elle soit concrète, comme ici, ou abstraite, comme dans les sections suivantes).

Théorème 4.7. *Une relation binaire $\prec$ supporte l'induction si et seulement si sa converse $\succ$ est fortement normalisante.*

Démonstration. Supposons que $\prec$ supporte l'induction : $A_\prec = \mathcal{P}$, et raisonnons par l'absurde : s'il existe une séquence infinie $(p_i)_{i\in\mathbb{N}}$, telle que pour tout $i \in \mathbb{N}$, $p_i \succ p_{i+1}$, alors l'ensemble $A_\prec \setminus \{p_i \mid i \in \mathbb{N}\}$ satisfait (4.2), ce qui contredit l'hypothèse de minimalité de $A_\prec$.

Pour l'autre implication, supposons $\succ$ fortement normalisante, considérons un ensemble $\mathcal{Q}$ satisfaisant (4.2), et montrons par l'absurde que $\mathcal{Q} = \mathcal{P}$. Soit $\mathcal{Q}' = \mathcal{P} \setminus \mathcal{Q}$, $\mathcal{Q}'$ satisfait :

$$\forall p \in \mathcal{Q}',\ \exists q \in \mathcal{Q}',\ p \succ q \ . \tag{4.3}$$

Si $\mathcal{Q}'$ est non vide, alors il contient un élément p_0, qui nous permet de démarrer une séquence infinie selon $\succ$, contredisant l'hypothèse de forte normalisation (la construction formelle de cette séquence à partir de (4.3) correspond en fait à l'axiome du choix dépendant). ■

La preuve du théorème 4.7 n'étant cependant pas constructive, nous utiliserons principalement la notion de support de l'induction dans la suite, afin de conserver des preuves constructives. Notons toutefois qu'en pratique, lorsque l'on peut prouver la forte normalisation d'une relation de façon constructive, on peut aussi prouver constructivement et directement que

sa converse supporte l'induction. Nous laissons donc le lecteur convertir librement les hypothèses de support de l'induction en des hypothèses (plus standard) de forte normalisation.

On peut finalement donner la version « commutation » du lemme de Newman (cette version apparaît dans [TeR03, exercice 1.3.2]).

Théorème 4.8 (Lemme de Newman, version commutation)**.** *Soient R et $\to$ deux relations binaires sur $\mathcal{P}$.*

$$Si \begin{cases} \overline{R} \cup \leftarrow \textit{ supporte l'induction} \\ R \textit{ et } \to \textit{ commutent localement} \end{cases} \quad \textit{alors } R \textit{ et } \to \textit{ commutent.}$$

Démonstration. On procède par induction bien fondée, avec le prédicat suivant :

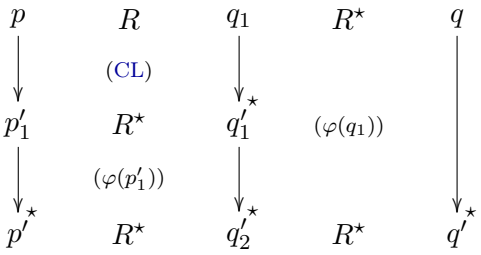
$$\varphi(p) : \ \forall p', q \in \mathcal{P},\ p \to^\star p' \text{ et } p\ R^\star\ q \text{ impliquent } \exists q',\ q \to^\star q' \text{ et } p'\ R^\star\ q' \ .$$

Le seul cas non trivial est celui où $p \to^+ q$ et $p \to^+ p'$, c'est-à-dire, où l'on a p'_1, q_1 tels que $p \to p'_1 \to^\star p'$ et $p\ R\ q_1\ R^\star\ q$. L'hypothèse de commutation locale nous donne alors q'_1 tel que $q_1 \to^\star q'_1$ et $p'_1\ R^\star\ q'_1$.

Comme $p \to p'_1$, on peut appliquer l'hypothèse d'induction en p'_1 ($\varphi(p'_1)$), ce qui nous donne q'_2 tel que $q'_1 \to^\star q'_2$ et $p'\ R^\star\ q'_2$.

Comme $p\ R\ q_1$, on peut conclure en appliquant l'hypothèse d'induction en q_1 ($\varphi(q_1)$), ce qui nous donne q' tel que $q \to^\star q'$ et $q'_2\ R^\star\ q'$. ■

Le cas détaillé dans la preuve est illustré par le diagramme ci-dessous :



On pourrait étendre directement ce lemme à la simulation faible, et obtenir le résultat suivant :

Proposition 4.9. *Si R est une simulation faible modulo transitivité telle que $(\overline{R} \cup \overset{\tau}{\leftarrow})$ supporte l'induction, alors $R^\star$ est une simulation faible.*

On s'aperçoit notamment que l'hypothèse de terminaison n'est pas satisfaite par le contre-exemple donné plus haut. Mais cette hypothèse reste assez contraignante :

1. elle implique la terminaison de $\xrightarrow{\tau}$: la technique ne s'applique que dans des LTS réactifs. De nombreux systèmes ont effectivement cette propriété, mais il existe aussi de nombreux cas où l'introduction de « boucles silencieuses » (τ-*loops*) est nécessaire : lorsque l'on modélise des systèmes tolérants aux fautes par exemple, certains algorithmes consistent à répéter une action jusqu'à ce qu'elle aboutisse (*data-link protocol* [GS96])).
2. Elle porte sur le candidat de bisimulation dans son ensemble, de telle sorte que la technique ne peut être utilisée que dans les cas où la bisimulation finale satisfait cette hypothèse de terminaison (notamment, cette bisimulation ne peut pas être symétrique, à moins d'être vide).

Nous montrons maintenant comment les contourner, en restant dans le cadre plus simple et plus intuitif de la commutation dans une algèbre relationnelle propre. Le résultat sera étendu au cas d'une algèbre relationnelle quelconque à la section 4.2 puis appliqué à la bisimulation à la section 4.3.

4.1.2 Un nouveau résultat de commutation

En analysant mieux le contre-exemple (4.1), on s'aperçoit que R et $\rightarrow$ sont fortement normalisantes : le cycle qui entraîne la divergence de $R \cup \rightarrow$ n'exhibe qu'une alternance infinie d'« étapes R » et « d'étapes $\rightarrow$ ». Partant de cette observation, on va démontrer que le lemme de Newman peut être renforcé en ne supposant que la terminaison de $R^+ \cdot \rightarrow^+$, c'est-à-dire l'absence de séquence alternant infiniment les étapes R et les étapes $\rightarrow$.

Définition 4.10 (Produit lexicographique)**.** Soient $\langle \mathcal{P}, \prec_\mathcal{P} \rangle$ et $\langle \mathcal{Q}, \prec_\mathcal{Q} \rangle$ deux ensembles munis d'une relation binaire chacun. On définit sur $\mathcal{P} \times \mathcal{Q}$ leur *produit lexicographique* comme la relation suivante :

$$\langle \prec_\mathcal{P}, \prec_\mathcal{Q} \rangle_{\text{lex}} \triangleq \left\{ \langle \langle p, q \rangle, \langle p', q' \rangle \rangle \mid p \prec_\mathcal{P} p', \text{ ou } \quad p = p' \text{ et } q \prec_\mathcal{Q} q' \right\} \quad .$$

La proposition suivante rappelle quelques propriétés standard, nous donnerons la preuve des trois derniers points dans le cadre plus général d'une algèbre relationnelle quelconque (proposition 4.25).

Proposition 4.11. *1. Le produit lexicographique de deux relations supportant l'induction supporte l'induction.*

2. L'ordre $<_\mathbb{N}$ sur les entiers naturels supporte l'induction.

3. Si $R \subseteq S$ et si S supporte l'induction, alors R aussi.

4. Une relation R supporte l'induction si et seulement si sa fermeture transitive (R^+) la supporte.

5. Le produit $R \cdot S$ de deux relations supporte l'induction si et seulement si le produit inversé ($S \cdot R$) la supporte aussi.

Notons que ces propriétés peuvent également être établies directement (et constructivement) pour la notion de forte normalisation.

Première généralisation

On a maintenant suffisamment d'outils pour s'embarquer dans la preuve du résultat annoncé. Nous le commentons ci-dessous.

Théorème 4.12. *Soient R et $\to$ deux relations binaires sur $\mathcal{P}$.*

$$\textit{Si } \begin{cases} \overline{R}^+ \cdot \leftarrow^+ \textit{ supporte l'induction} \\ R \textit{ et } \to \textit{ commutent localement} \end{cases} \quad \textit{alors } R \textit{ et } \to \textit{ commutent.}$$

Démonstration. Par le lemme 4.2 il suffit de montrer que $\to$ et R commutent à moitié. Par la proposition 4.11, on peut raisonner par induction bien fondée sur le produit $\langle \leftarrow^+ \cdot \overline{R}^+, <_{\mathbb{N}} \rangle_{\text{lex}}$; on choisit le prédicat suivant :

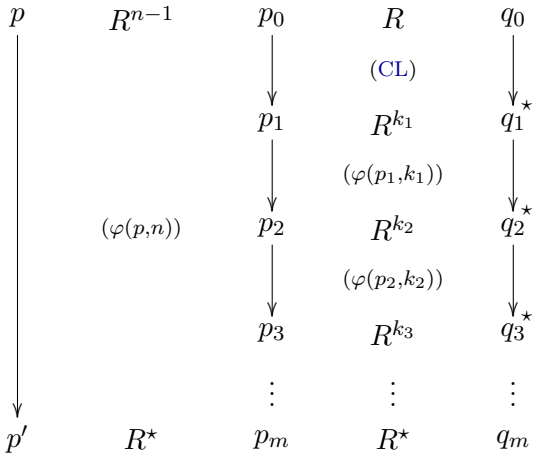
$$\varphi(p,n) : \ \forall p', q_0 \in \mathcal{P},\ p' \leftarrow p\ R^n\ q_0 \text{ implique } p'\ R^\star \cdot \leftarrow^\star q_0\ .$$

Soient p, p', q_0, n tels que $p' \leftarrow p\ R^n\ q_0$ et montrons $p'\ R^\star \cdot \leftarrow^\star q_0$:

- Le cas $n = 0$ est trivial, et le cas $n = 1$ correspond exactement à l'hypothèse de commutation locale.
- Supposons donc $n > 1$; soit p_0 tel que $p\ R^{n-1}\ p_0\ R\ q_0$. Par l'hypothèse d'induction, $\varphi(p, n-1)$, on obtient $m \in \mathbb{N}$ et $p_1, \ldots, p_m$ tels que $p'\ R^\star\ p_m$ et $\forall i < m,\ p_i \to p_{i+1}$. On construit ensuite par récurrence une séquence $q_1, \ldots, q_m$ telle que $\forall i < m,\ q_i \to^\star q_{i+1}$ et $p_{i+1}\ R^\star\ q_{i+1}$:
  - q_1 est donné par l'hypothèse de commutation locale ;
  - supposant la séquence construite jusqu'à un rang i tel que $0 < i < m$, q_{i+1} s'obtient en appliquant l'hypothèse d'induction à p_i, p_{i+1} et q_i : on vérifie bien que l'on a $p_i \leftarrow^+ \cdot \overline{R}^+ p$ (on a $n - 1 > 0$ et $i > 0$), de telle sorte que $\varphi(p_i, k_i)$ soit vrai, où k_i est le nombre arbitraire d'étapes entre p_i et q_i ; on obtient bien ainsi q_{i+1}, tel que $q_i \to^\star q_{i+1}$ et $p_{i+1}\ R^\star\ q_{i+1}$.

  Le dernier élément de cette séquence permet de conclure : on a $p'\ R^\star\ q_m$ et $q_0 \to^\star q_m$. ■

Le second cas de cette preuve est représenté ci dessous :



On a bien une généralisation du théorème 4.8 : par la proposition 4.11, la normalisation forte de $R \cup \rightarrow$ implique celle de $R^+ \cdot \rightarrow^+$, et il existe de nombreux cas où la réciproque n'est pas vraie (il suffit par exemple que R et $\rightarrow$ contiennent deux cycles indépendants ; on verra au chapitre 6 un exemple d'application concret et utile en pratique). Si par contre on choisit $R = \rightarrow$, on retombe exactement sur le lemme de Newman, dans sa formulation classique en termes de confluence (par la proposition 4.11(4), la normalisation forte de $\rightarrow^+ \cdot \rightarrow^+$ équivaut à celle de $\rightarrow$).

Ici encore, ce résultat s'étend naturellement à la simulation faible (la preuve de cette extension, comme celle de la proposition 4.9, est non triviale : elle nécessite une nouvelle induction bien fondée suivant les grandes lignes de la preuve précédente ; on l'omet puisque ce résultat sera généralisé dans la suite) :

Proposition 4.13. *Si R est une simulation faible modulo transitivité telle que $\overline{R}^+ \cdot \overset{\mathcal{I}}{\Leftarrow}$ supporte l'induction, alors $R^\star$ est une simulation faible.*

Ce résultat permet de contourner la première limitation évoquée précédemment : la technique modulo correspondante peut s'appliquer dans des systèmes non réactifs, où les transitions silencieuses ne terminent pas. Par contre la seconde limitation persiste : il faut que le candidat tout entier satisfasse l'hypothèse de terminaison.

Raffinement de cette généralisation

On va donc chercher à séparer l'hypothèse de terminaison de celle de commutation. On introduit pour cela une troisième relation (S), pour porter l'hypothèse de terminaison :

Théorème 4.14. *Soient R, S et $\rightarrow$ trois relations binaires sur $\mathcal{P}$.*

$$\textit{Si} \begin{cases} \overline{S}^+ \cdot \leftarrow^+ & \textit{supporte l'induction} \\ S \subseteq R & \\ \leftarrow \cdot R \subseteq S^\star \cdot R \cdot \leftarrow^\star & \end{cases} \qquad \textit{alors } R \textit{ et } \rightarrow \textit{ commutent.}$$

L'hypothèse locale de commutation peut être représentée par le diagramme suivant :

$$\begin{array}{ccc} \cdot & R & \\ \downarrow & \subseteq & \downarrow \\ & S^\star \cdot R & \cdot^\star \end{array}$$

Démonstration. On démontre que R et $\rightarrow$ commutent à moitié, cela suffisant à conclure par le lemme 4.2. On raisonne par induction bien fondée sur $\mathcal{P} \times \mathbb{N}$,

muni du produit lexicographique $\langle \overline{S}^+ \cdot \leftarrow^+, <_{\mathbb{N}} \rangle_{\text{lex}}$, avec le prédicat suivant :

$$\varphi(u,n) : \ \forall p, p'_0, q \in \mathcal{P}, \ \begin{cases} u \rightarrow^\star p \rightarrow^n p'_0 \\ p \ R \ q \end{cases} \quad \text{impliquent } p'_0 \ R^\star \cdot \leftarrow^\star q \ .$$

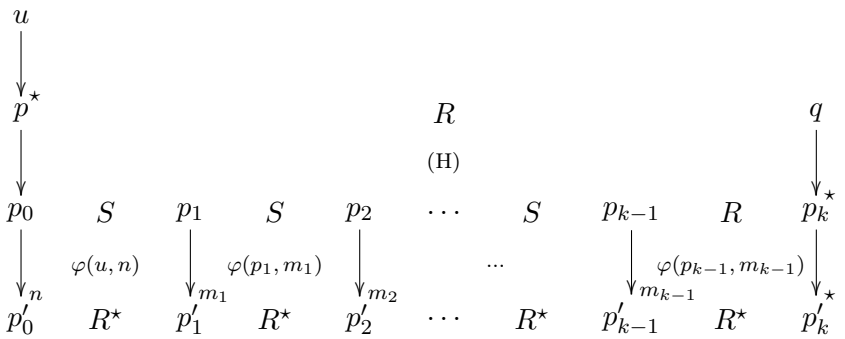
Le cas $n = 0$ est immédiat, supposons donc $\varphi(n)$, et montrons $\varphi(n+1)$. Ce second cas est illustré par le diagramme donné après la preuve. Soient p, p_0, p'_0, q tels que $p \ R \ q$ et $u \rightarrow p \rightarrow p_0 \rightarrow^n p'_0$, en appliquant l'hypothèse de commutation à $p_0 \leftarrow p \ R \ q$, il existe $k > 0$ et $p_1, \ldots, p_k$ tels que $q \rightarrow^\star p_k$, $p_{k-1} \ R \ p_k$ et $\forall i \in [1; k-1], \ p_{i-1} \ S \ p_i$.

On définit maintenant par récurrence interne une séquence $(p'_i)_{0<i\leq k}$ telle que l'on ait $\forall i \in [1; k]$, $p_{i-1} \ R^\star \ p'_i \leftarrow^\star p_i$.

- si $i = 1$, on applique l'hypothèse d'induction externe : $\varphi(u, n)$, au point p_0 ($p'_0 \leftarrow^n p_0 \ R \ p_1$) (rappelons que l'on a $S \subseteq R$) : il existe p'_1 tel que $p'_0 \ R^\star \ p'_1$ et $p_1 \rightarrow^\star p'_1$. C'est ici qu'est nécessaire l'introduction artificielle de la variable u dans la définition de φ.
- sinon, $i > 1$, supposons la séquence construite jusqu'à $i-1$, et remarquons que $u \rightarrow^+ \cdot \ S^+ \ p_{i-1}$, de telle sorte que l'on peut obtenir p'_i en appliquant l'hypothèse d'induction externe, $\varphi(p_{i-1}, m_{i-1})$, à $p'_{i-1} \leftarrow^{m_{i-1}} p_{i-1} \ R \ p_i$ (m_{i-1} étant le nombre arbitraire de transitions entre reliant p_{i-1} à p'_{i-1}, on utilise là aussi le fait que $S \subseteq R$).

On peut conclure : on a $p'_0 \ R^\star \ p'_k \leftarrow^\star q$, d'où $\varphi(u, n+1)$, puis $\forall u \in \mathcal{P}, \ \forall n \in \mathbb{N}, \ \varphi(u, n)$ par induction bien fondée. Comme on peut toujours choisir $u = p$ dans la définition de φ, on a bien obtenu la commutation à moitié de R sur $\rightarrow$. ■

Le diagramme ci-dessous permet de mieux visualiser le second cas de la preuve :



Ce théorème généralise le précédent (théorème 4.12) : il suffit de considérer le cas $R = S^=$. L'avantage majeur de ce raffinement est qu'il permet son utilisation pour prouver la commutation de relations ne satisfaisant pas nécessairement la condition de terminaison : cette condition n'est exigée que sur une relation auxiliaire, utilisée seulement pour la preuve (elle n'apparaît plus dans les conclusions du théorème).

On peut lire ce théorème comme suit : pour montrer que deux relations commutent, montrons tout d'abord qu'elles commutent localement, puis rassemblons dans une relation S tous les couples utilisés en « position de transitivité », dans chacun des diagrammes de commutation locale que l'on vient de fermer. Il suffit alors d'étudier la terminaison de $S^+\cdot \to^+$: si cette relation normalise fortement, les deux relations initiales commutent ; sinon c'est qu'on a plus ou moins « triché », en utilisant la transitivité pour esquiver l'une des offres proposées par la commutation locale.

De façon plus technique, lorsque l'on considère la preuve (notamment le diagramme qui la suit), on s'aperçoit que l'utilisation de la relation auxiliaire S permet de cibler exactement l'hypothèse de terminaison dont on a besoin pour la preuve : seuls les couples à travers lesquels on doit transférer l'hypothèse d'induction sont concernés par l'hypothèse de terminaison. En particulier, sur le diagramme, le dernier couple $\langle p_{k-1}, p_k\rangle$, qui ne sera jamais traversé par l'hypothèse d'induction, n'a pas besoin d'appartenir à S.

Terminaison relative

Geser étudie dans sa thèse [Ges90] la notion de *terminaison relative* : une relation binaire R *termine relativement à* une seconde relation S lorsque $S^\star \cdot R \cdot S^\star$ est fortement normalisante. C'est intuitivement le cas lorsque S « n'entrave pas » la terminaison de R : rajouter des transitions selon S ne permet pas de construire une séquence infinie comportant une infinité d'étapes selon R.

Notre hypothèse de terminaison est plus faible que cette notion : si R termine relativement à $\to$ (ou l'inverse), alors $R^+\cdot \to^+$ est fortement normalisante. On peut donc utiliser les nombreuses techniques définies par Geser pour montrer des résultats de terminaison relative. Nous passerons par cette notion lors de l'étude des systèmes réactifs (section 4.5) ainsi qu'au chapitre 6 : comme expliqué à la section 4.3.3, le cas où l'hypothèse de terminaison peut être prouvée en passant par la terminaison relative nous semble fréquent dans le cadre de la bisimulation.

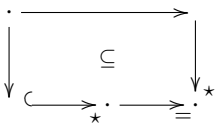
Notons aussi que l'on trouve dans la thèse de Geser un résultat assez proche du théorème 4.14 (le lemme de « coopération locale », dû à Bellegarde et Lescanne [BL87]) : l'hypothèse de terminaison est plus forte, $\to$ doit terminer relativement à S, mais le diagramme local de commutation est un peu plus permissif.

Applications à la réécriture ?

De manière complètement fortuite, le théorème 4.14 admet un corollaire qui pourrait être utile pour démontrer des résultats de confluence. En effet, le cas $R = \to^=$ qui correspondait au lemme de Newman dans sa version confluence dans le théorème 4.12, donne ici une réelle généralisation de ce

lemme : la relation dont on souhaite prouver la confluence n'a pas besoin d'être elle-même fortement normalisante, puisque l'on peut utiliser une relation auxiliaire pour spécifier plus précisément l'argument de terminaison :

Corollaire 4.15. *Soient $\to$ et $\hookrightarrow$ deux relations binaires sur $\mathcal{P}$ telles que $\hookrightarrow \subseteq \to$. Si $\hookrightarrow^+ \cdot \to^+$ est fortement normalisante, et si ces deux relations satisfont le diagramme suivant ($\leftarrow \cdot \to \subseteq \hookrightarrow^\star \cdot \to^= \cdot \leftarrow^\star$), alors $\to$ est confluente.*



Remarquons que si l'on prend la relation vide pour $\hookrightarrow$, on obtient le lemme de « confluence forte » de Gérard Huet [Hue80], et que l'on retombe toujours sur le lemme de Newman lorsque l'on égalise les deux relations.

Nous n'avons pour l'instant pas d'exemple d'application concrète utilisant les nouvelles possibilités offertes par ce corollaire. Notamment, il ne nous semble pas qu'il puisse aider à montrer la confluence du λ-calcul pur, sans passer par les réductions parallèles [Bar84] : nous n'avons pas trouvé le moyen de définir une sous-réduction fortement normalisante de la β-réduction, qui suffise à clore les diagrammes de confluence locale selon la contrainte imposée par notre corollaire.

Version symétrique du résultat ?

L'énoncé du théorème 4.14 est étrangement « asymétrique », et pose naturellement la question de la généralisation « symétrique » suivante :

> « *Soient R, S, $\to$ et $\hookrightarrow$ quatre relations binaires sur $\mathcal{P}$.*
>
> $$\textit{Si} \begin{cases} \overline{S}^+ \cdot \hookleftarrow^+ & \textit{supporte l'induction} \\ S \subseteq R \textit{ et } \hookrightarrow \subseteq \to & \\ \leftarrow \cdot R \subseteq S^\star \cdot R \cdot \leftarrow \cdot \hookleftarrow^\star & \end{cases} \qquad \textit{alors } R \textit{ et } \to \textit{ commutent.}$$
>
> ».

Cette généralisation n'est cependant pas valide, comme le montre le contre-exemple suivant :

$$R \triangleq \{\langle 1,1\rangle, \langle 2,3\rangle, \langle 3,4\rangle, \langle 4,5\rangle\} \qquad S \triangleq \{\langle 3,4\rangle\}$$
$$\to \triangleq \{\langle 5,5\rangle, \langle 4,3\rangle, \langle 3,2\rangle, \langle 2,1\rangle\} \qquad \hookrightarrow \triangleq \{\langle 3,2\rangle\} \quad .$$

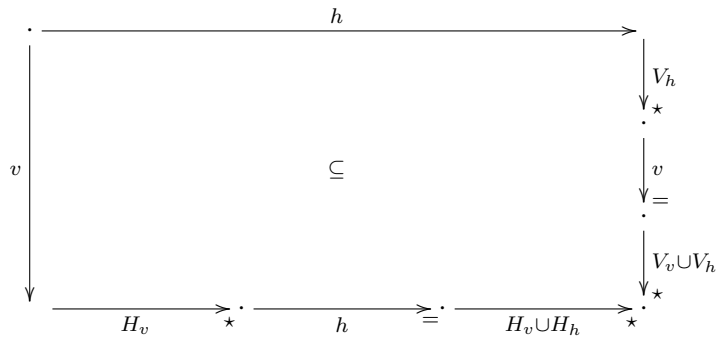
4.1.3 Diagrammes décroissants

Avant d'étendre les résultats précédents au cadre abstrait des algèbres relationnelles, et à la bisimilarité faible, nous montrons comment le théorème 4.14 peut s'obtenir à l'aide de la théorie des « diagrammes décroissants » de Vincent van Oostrom [vO94] (théorie complétée dans [BKvO98] et reprise dans [TeR03, chapitre 14]). Le théorème principal [vO94, théorème 3.7] peut se reformuler comme suit ; nous l'expliquons brièvement ci-dessous.

Théorème 4.16 (Commutation par les diagrammes décroissants)**.**
Soient I un ensemble muni d'un ordre partiel $(\prec)$ supportant l'induction, et $(\underset{i}{\longrightarrow})_{i\in I}$ une collections de relations indexées par I. Pour tous $J \subseteq I$ et $i \in I$, on note J_i et $\underset{J}{\longrightarrow}$ la partie et la relation suivantes :

$$J_i \triangleq \{j \in J \mid j \prec i\} \quad , \qquad \underset{J}{\longrightarrow} \triangleq \bigcup_{j\in J} \underset{j}{\longrightarrow} \quad .$$

Pour toutes parties $H, V \subseteq I$, si le diagramme décroissant *ci-dessous est satisfait pour tous $h \in H$ et $v \in V$,*



alors $\underset{H}{\longrightarrow}$ et $\underset{V}{\longrightarrow}$ commutent.

L'idée générale est de contraindre les diagrammes de commutation locale, de manière à ce que leur utilisation, lors du « pavage » du diagramme de commutation globale, fasse systématiquement progresser, en faisant diminuer strictement la « taille » du problème (on évite ainsi le contre-exemple (1.18), où l'on se retrouve au point de départ après avoir utilisé deux diagrammes de commutation locale). Ce sont les étiquettes (I) [1] qui sont utilisées pour assurer cette progression, en s'appuyant sur le support de l'induction de la relation $(\prec)$. Comme ces étiquettes n'apparaissent plus dans le résultat de commutation final, on est libre d'étiqueter comme on le souhaite les relations dont on veut prouver la commutation ($\underset{H}{\longrightarrow}$ et $\underset{V}{\longrightarrow}$).

Cédric Fournet utilise cette méthode dans sa thèse [Fou98, chapitre 4], afin d'obtenir des techniques modulo pour la *bisimulation barbelée* [MS92a] :

1. Ces dernières n'ont rien à voir avec les étiquettes des LTS.

en choisissant un ordre adéquat entre les différentes relations (expansion, bi-expansion et bisimilarité barbelées), le candidat de bisimulation et la relation de réduction, les conditions imposées par le théorème des diagrammes décroissants permettent de retrouver aisément les différentes techniques standard.

Pour pouvoir passer à la bisimilarité *étiquetée*, dont il est question dans cette thèse, il faudrait étendre légèrement le théorème 4.16 afin de garantir la conservation stricte des transitions visibles. En effet, ce théorème, qui établit un résultat de commutation « pure », laisserait répondre plusieurs transitions selon a lors d'une offre visible selon a. Cette extension est relativement simple [vO05] : il suffit de renforcer légèrement l'invariant utilisé dans la preuve du théorème ; elle nécessite cependant de refaire tout le travail… Le problème ne se pose pas avec la bisimilarité barbelée, qui remplace les étiquettes visibles par des *barbes*, et où tous les jeux sont silencieux.

On peut cependant démontrer sans modification de la théorie le théorème 4.14, qui n'établit pour l'instant qu'un résultat de commutation pure. Cela est cependant assez délicat : alors que Fournet utilise un ordre partiel qui supporte trivialement l'induction, car portant sur un nombre fini d'étiquettes (les étiquettes correspondent aux différentes relations mises en jeu), il nous faut ici transférer l'hypothèse de terminaison depuis les processus vers les étiquettes. Comme le fait van Oostrom pour démontrer le lemme de Newman [vO94, corollaire 4.4], l'astuce consiste à utiliser les processus pour étiqueter les relations (technique courante pour prouver des résultats de terminaison [MOZ96]).

On commence par démontrer à nouveau le théorème 4.12, qui est un peu plus simple, et permet ainsi de mieux illustrer la méthode. Dans les deux cas, on utilise la caractérisation du support de l'induction par la forte normalisation (théorème 4.7) afin de rendre les preuves plus intuitives ; on pourrait cependant s'en passer et travailler constructivement, en n'utilisant que la notion de support de l'induction.

Démonstration du théorème 4.12, par les diagrammes décroissants.
Soient R et $\to$ deux relations qui commutent localement, et telles que $R^+ \cdot \to^+$ normalise fortement ; montrons qu'elles commutent.

Soit $I \triangleq \mathcal{P} \times \{0,1\} \times \mathcal{P}$; définissons la famille de relations indexée par I comme suit : pour tous $p, q \in \mathcal{P}$,

$$p' \xrightarrow[\langle p,0,q\rangle]{} q' \quad \text{si} \quad p' = p,\ q' = q \quad \text{et } p'\ R\ q' \ ,$$
$$p' \xrightarrow[\langle p,1,q\rangle]{} q' \quad \text{si} \quad p' = p,\ q' = q \quad \text{et } p' \to q' \ .$$

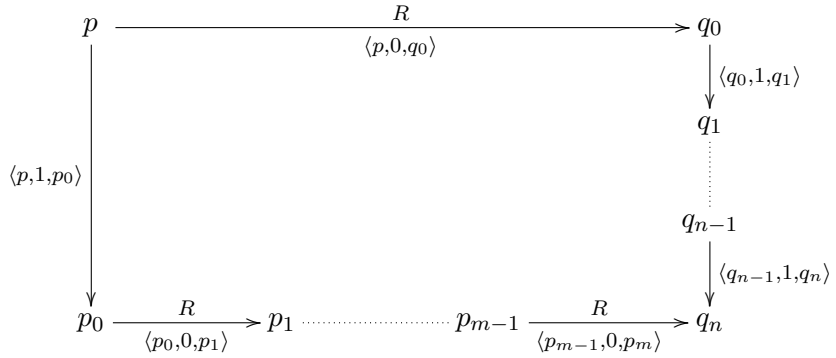
En posant $H \triangleq \mathcal{P} \times \{0\} \times \mathcal{P}$ et $V \triangleq \mathcal{P} \times \{1\} \times \mathcal{P}$, on a $\underset{H}{\longrightarrow} = R$ et $\underset{V}{\longrightarrow} = \to$.

Définissons ensuite l'ordre partiel sur I est ensuite, comme la fermeture transitive de la relation $(\prec_1)$ suivante :

$$\langle p', m, p\rangle \succ_1 \langle q, n, q'\rangle \;\text{ si }\; p\,(R \cup \to)^\star\, q \;\text{ et } n = 1 - m \qquad \prec \triangleq \prec_1^+ \;.$$

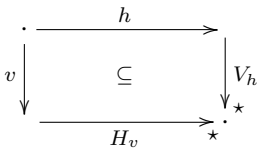
La relation $\succ_1$ est fortement normalisante : de toute séquence infinie selon cette relation, on déduit une séquence infinie selon $(R \cup \to)$, comportant une infinité de pas selon R et une infinité de pas selon $\to$, ce qui contredit l'hypothèse de forte normalisation de $R^+ \cdot \to^+$. L'ordre partiel $\prec$ supporte donc l'induction, par le théorème 4.7 et la proposition 4.11(4).

Les seuls diagrammes à vérifier sont de la forme $p_0 \xleftarrow[\langle p,1,p_0\rangle]{} p \xrightarrow[\langle p,0,q_0\rangle]{} q_0$, et l'on a par définition $p \to p_0$ et $p\ R\ q_0$. L'hypothèse de commutation locale nous donne alors $p_1, \ldots, p_m$ et $q_1, \ldots, q_n$ tels que $p_m = q_n$, $\forall i < m,\ p_i\ R\ p_{i+1}$ et $\forall i < n,\ q_i \to q_{i+1}$:



On étiquette ensuite ces réductions comme on l'a fait sur le diagramme précédent, ce qui permet de vérifier que l'on a bien fermé le diagramme selon la contrainte imposée par le théorème des diagrammes décroissants : on a $\forall i < m,\ \langle p, 1, p_0\rangle \succ_1 \langle p_i, 0, p_{i+1}\rangle$ et $\forall i < n,\ \langle p, 0, q_0\rangle \succ_1 \langle q_i, 1, q_{i+1}\rangle$. ■

Notons que la preuve précédente utilise en fait seulement le diagramme local suivant, qui est à première vue un cas très particulier de celui du théorème 4.16. Cependant, par notre définition de la relation $(\prec)$, il se trouve être équivalent : les libertés dont on n'essaie pas de profiter ne sont que « virtuelles », une fois fixé l'ordre partiel $(\prec)$.



Cette preuve se comprend mieux « à l'envers » : une fois prise la décision d'étiqueter les relations par les processus, il suffit de considérer le diagramme de commutation que l'on a en hypothèse, et de définir l'ordre

partiel de manière à rendre le diagramme décroissant, tout en s'assurant qu'il supporte l'induction. Si cette méthode échoue, il faut repenser la manière dont on a étiqueté les relations (dans le cas énumérable, la méthode est *complète* [BKvO98] : si les relations commutent effectivement, alors il existe un moyen d'étiqueter les diagrammes de commutation locale, qui les rende décroissants selon un ordre partiel supportant l'induction – rien n'assure cependant que cet étiquetage soit simple, ou naturel...).

En suivant cette stratégie, on aboutit à la preuve suivante du théorème 4.14 ; la difficulté consiste à bien étiqueter la relation R (la relation S prenant la place de la relation R de la preuve précédente) :

Démonstration du théorème 4.14, par les diagrammes décroissants.
Soient R, S et $\to$ trois relations telles que $S^+ \cdot \to^+$ normalise fortement, $S \subseteq R$ et $\leftarrow \cdot R \subseteq S^\star \cdot R \cdot \leftarrow^\star$; montrons que R et $\to$ commutent.

Soit $I \triangleq (\mathcal{P} \times \{0,1\} \times \mathcal{P}) \uplus \mathcal{P}$ (où $\uplus$ dénote l'union disjointe de deux ensembles) ; définissons la famille de relations indexée par I comme suit : pour tous $p, q \in \mathcal{P}$,

$$\begin{aligned}
p' &\xrightarrow[\langle p,0,q\rangle]{} q' \quad \text{si } \; p' = p,\; q' = q \;\text{ et } p' \; S \; q' \;, \\
p' &\xrightarrow[\langle p,1,q\rangle]{} q' \quad \text{si } \; p' = p,\; q' = q \;\text{ et } p' \to q' \;, \\
p' &\xrightarrow[q]{} q' \quad \text{si } \; q \to^\star q' \;\text{ et } p' \; R \; q' \;.
\end{aligned}$$

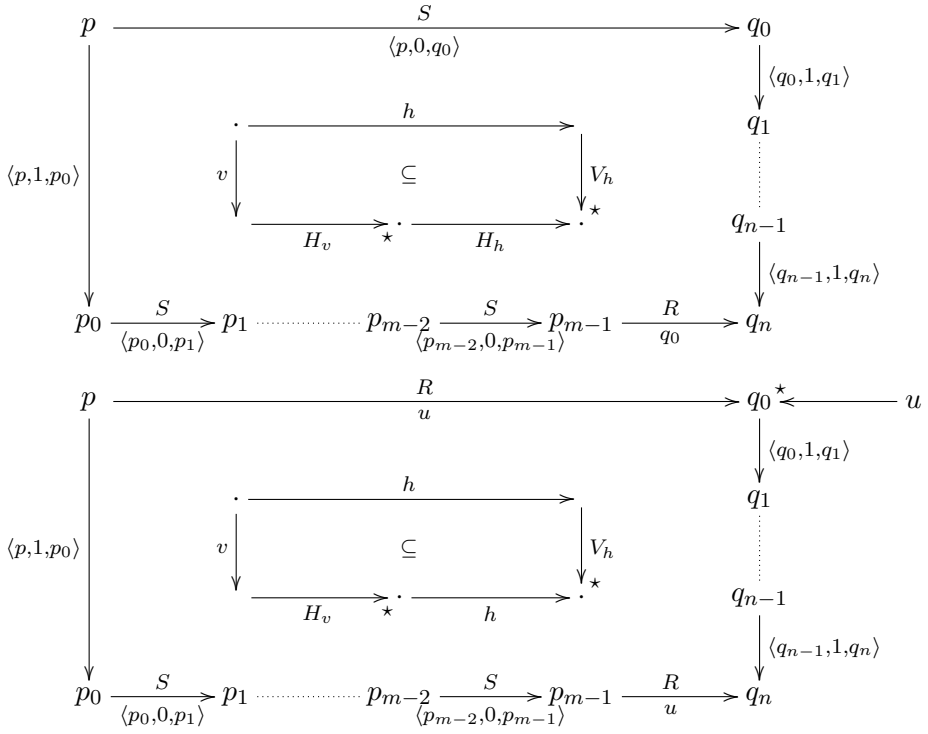
Posons $H \triangleq (\mathcal{P} \times \{0\} \times \mathcal{P}) \uplus \mathcal{P}$ et $V \triangleq \mathcal{P} \times \{1\} \times \mathcal{P}$; on a $\underset{H}{\longrightarrow} = R$ et $\underset{V}{\longrightarrow} = \to$. Soit $\prec$ la relation sur I, définie comme la fermeture transitive de la relation $\prec_1$ suivante :

$$\begin{aligned}
\langle p', m, p\rangle &\succ_1 \langle q, n, q'\rangle \qquad \text{si } \; p \; (S \cup \to)^\star \; q \;\text{ et } n = 1 - m \\
\langle p', m, p\rangle &\succ_1 q \qquad \text{si } \; p \; (S \cup \to)^\star \; q \;\text{ et } m = 0 \\
p &\succ_1 \langle q, n, q'\rangle \qquad \text{si } \; p \; (S \cup \to)^\star \; q \;\text{ et } n = 1 \;.
\end{aligned}$$

Le support de l'induction de $(\prec)$ est plus délicat que précédemment : il faut remarquer que l'on peut extraire de toute séquence infinie selon $\succ_1$ une séquence infinie ne comportant pas d'élément de la forme q_0 : en effet, de tels éléments apparaissent nécessairement entre un élément de la forme $\langle p', 0, p\rangle$ et un autre de la forme $\langle q, 1, q'\rangle$, de telle sorte que l'on peut les retirer : on a dans ce cas $\langle p', 0, p\rangle \succ_1 \langle q, 1, q'\rangle$. On extrait ensuite de cette sous-séquence une séquence de processus selon $(S \cup \to)$, qui contredit l'hypothèse de forte normalisation.

Il nous reste à vérifier que les deux diagrammes suivants sont autorisés par

le théorème 4.16 (le diagramme réellement satisfait est dessiné à l'intérieur) :



■

Cette preuve n'est pas des plus simples, même une fois comprise la méthode des diagrammes décroissants ; elle en illustre néanmoins l'expressivité. Il nous semble que l'avantage principal de cette méthode est qu'elle capture une fois pour toutes l'application du raisonnement par induction bien fondée lors de la fermeture de diagrammes de commutation : on n'a plus, comme dans la première preuve que l'on a donnée, à imbriquer « dangereusement » des récurrences et des inductions lexicographiques bien fondées. On reste cependant toujours forcé de manipuler les processus, afin d'exprimer l'hypothèse de support de l'induction.

4.2 Terminaison et commutation abstraites

Mises à part les notions de support de l'induction et de forte normalisation, l'ensemble des définitions et résultats de la section précédente peuvent être énoncés dans une algèbre relationnelle quelconque : on n'y parle que de compositions de relations, de fermetures transitives, d'inclusions, etc... Les éléments de $\mathcal{P}$ n'y sont notamment jamais mentionnés. En utilisant les résultats de Doornbos, Backhouse et van der Woude [DBvdW97], qui permettent justement de caractériser la notion de bonne fondation dans une

algèbre relationnelle quelconque, nous montrons comment prouver les résultats précédents dans ce cadre plus général.

On se place donc à nouveau dans une algèbre relationnelle (X), jusqu'à la fin du chapitre.

4.2.1 Support abstrait de l'induction

Nous commençons par une synthèse de la partie de [DBvdW97] qui nous sera utile par la suite. Notre présentation s'éloigne quelque peu de celle de cet article, et n'en couvre qu'une partie. Nous donnons quelques indices sur ce que nous omettons, et nous invitons vivement le lecteur à s'y référer pour toute étude plus approfondie dépassant le cadre de l'utilisation que l'on fait de ces résultats.

Cet article introduit une distinction entre la notion de *bonne fondation* d'une relation, et celle de relation *supportant l'induction*, c'est-à-dire pour laquelle le principe d'induction est valide. Si ces deux notions coïncident dans toute algèbre relationnelle *complémentée* (en particulier dans toute algèbre relationnelle propre), ce n'est pas le cas en général : les relations supportant l'induction sont bien fondées, mais la réciproque n'est pas toujours vraie. On s'intéressera donc uniquement à la notion de support de l'induction, qui nous fournit une puissante méthode de preuve : libre à nous de caractériser ensuite cette notion en termes de forte normalisation dans le cadre des relations binaires (théorème 4.7), ou par n'importe quel autre moyen, dans toute autre algèbre relationnelle potentielle.

Facteurs

On commence par définir les *facteurs*, initialement étudiés par Robert Dilworth [Dil39] puis Garrett Birkhoff [Bir40] dans le cadre des treillis monoïdaux, où ils portent le nom de *résidus* (dans les allégories, ce sont les opérations de *division* [FS90]) ; bien qu'essentiels, nous ne les utiliserons que très peu.

Définition 4.17 (Facteurs)**.** Soient $x, y \in X$ deux relations. On définit le *facteur à gauche* de x par y et le *facteur à droite* de x par y comme suit :

$$y/x \triangleq \bigvee \{z \in X \mid z \cdot x \sqsubseteq y\} \quad ,$$
$$x\backslash y \triangleq \bigvee \{z \in X \mid x \cdot z \sqsubseteq y\} \quad .$$

La continuité du treillis complet monoïdal (2.18) permet d'en donner les propriétés suivantes :

Proposition 4.18. *Pour toutes relations $x, y, z \in X$, on a :*

$$z \sqsubseteq y/x \Leftrightarrow z \cdot x \sqsubseteq y \qquad z \sqsubseteq x\backslash y \Leftrightarrow x \cdot z \sqsubseteq y \ , \tag{4.4}$$
$$(y/x) \cdot x \sqsubseteq y \qquad x \cdot (x\backslash y) \sqsubseteq y \ . \tag{4.5}$$

Démonstration.

(4.4) : Supposons $z \sqsubseteq y/x$; on a $z \cdot x \sqsubseteq \bigvee \{z \in X \mid z \cdot x \sqsubseteq y\} \cdot x$ par (2.17), puis $z \cdot x \sqsubseteq \bigvee \{z \cdot x \mid z \cdot x \sqsubseteq y\}$ par la continuité du produit (2.18) ; on en déduit $z \cdot x \sqsubseteq y$ puisque l'on a $\bigvee \{z \cdot x \mid z \cdot x \sqsubseteq y\} \sqsubseteq y$. L'implication réciproque est immédiate, par définition des bornes supérieures. L'autre équivalence est similaire.

(4.5) : Il suffit de choisir respectivement $z = y/x$ et $z = x\backslash y$ dans les équivalences précédentes : leurs membres gauches deviennent ainsi triviaux. ■

Ces bonnes propriétés proviennent essentiellement de la continuité du produit (2.18). Doornbos et al. définissent ces facteurs par la propriété (4.4), en s'appuyant sur la théorie des « connections de Galois » [Ore44] : ce sera en effet la seule propriété de ces opérateurs qui nous sera utile. La propriété « d'annulation » (4.5) qui en découle justifie la terminologie et les notations de ces facteurs.

Monotypes, conditions

Les algèbres relationnelles sont initialement conçues pour représenter les relations binaires sur un ensemble $\mathcal{P}$ quelconque. L'une de leurs particularités est qu'elle peuvent aussi représenter les parties de $\mathcal{P}$. Par exemple, dans le cas des relations binaires une partie $\mathcal{Q}$ peut être représentée par :

1. la relation identité, restreinte à $\mathcal{Q}$: $\{\langle q, q\rangle \mid q \in \mathcal{Q}\}$,
2. la relation universelle, restreinte à $\mathcal{Q}$ à gauche : $\{\langle q, p\rangle \mid p \in \mathcal{P},\ q \in \mathcal{Q}\}$,
3. la relation universelle, restreinte à $\mathcal{Q}$ à droite : $\{\langle p, q\rangle \mid p \in \mathcal{P},\ q \in \mathcal{Q}\}$.

Revenant au niveau abstrait, cela correspond à trois catégories de relations :

1. les relations majorées par 1, que l'on appellera *monotypes* et que l'on notera A, B dans la suite – ce sont les morphismes *co-réflexifs* que l'on trouve dans les allégories [FS90] ;
2. les relations g telles que $g \cdot \top = g$, appelées *conditions à gauche* ;
3. symétriquement, les relations d telles que $\top \cdot d = d$, appelées *conditions à droite*.

Des isomorphismes de treillis relient ces trois représentations, ce qui fait que l'on peut travailler indifféremment avec chacune d'elles. Comme on va le voir, toutes ces représentations seront utiles par la suite, mais les monotypes ayant de meilleures propriétés calculatoires, ils seront privilégiés.

La proposition suivante renforce l'intuition derrière les monotypes : ils forment une sous-algèbre relationnelle, où la converse est l'identité, et le produit coïncide avec les bornes inférieures. La preuve de cette proposition repose très fortement sur la loi de modularité (3.4).

Proposition 4.19. *Soient $A, B \sqsubseteq 1$ deux monotypes. On a :*

$$A = \overline{A} = A \cdot A \ , \tag{4.6}$$
$$A \cdot B = A \wedge B \sqsubseteq 1 \ . \tag{4.7}$$

Démonstration. Pour tous monotypes $A, B \sqsubseteq 1$, on a tout d'abord $A \cdot B \sqsubseteq A$ et $A \cdot B \sqsubseteq B$, par (2.17).

(4.6) : La loi de modularité (3.4) nous donne $A = (A \cdot 1) \wedge 1 \sqsubseteq A \cdot (1 \wedge (\overline{A} \cdot 1)) = A \cdot \overline{A}$. On en déduit $A \sqsubseteq \overline{A}$, d'où $A = \overline{A}$ par (3.1) et (3.2), puis $A \sqsubseteq A \cdot \overline{A} = A \cdot A \sqsubseteq A$.

(4.7) : On a $A \cdot B \sqsubseteq A \wedge B \sqsubseteq 1$ par la première remarque, puis on utilise à nouveau la loi de modularité (3.4) : $A \wedge B \sqsubseteq A \cdot (1 \wedge (\overline{A} \cdot B)) = A \cdot \overline{A} \cdot B = A \cdot B$. ■

Domaine, restriction, plus faible pré-condition

Toute relation binaire R définit naturellement deux parties : son *domaine*, $\{p \in \mathcal{P} \mid \exists q \in \mathcal{P},\ p\ R\ q\}$, et son *co-domaine*, $\{q \in \mathcal{P} \mid \exists p \in \mathcal{P},\ p\ R\ q\}$. Au niveau abstrait, le domaine d'une relation x peut être représenté par la condition à gauche $x \cdot \top$ (ou par le monotype $(x \cdot \top) \wedge 1$), son co-domaine pouvant l'être par la condition à droite $\top \cdot x$ (ou par le monotype $(\top \cdot x) \wedge 1$). On peut ensuite aisément représenter les opérations de restriction du domaine ou du co-domaine d'une relation $x : A \cdot x$ est la *restriction du domaine* de x au monotype A, et $x \cdot A$ est la *restriction du co-domaine* de x au monotype A. Ces intuitions nous mènent à la définition suivante, que l'on explique ci-dessous :

Définition 4.20 (Facteurs monotypes)**.** Soient $x \in X$ une relation et $A \sqsubseteq 1$ un monotype. On définit le *facteur monotype à gauche* de x par A et le *facteur monotype à droite* de x par A comme les monotypes suivants :

$$A \not/ x \triangleq \bigvee \{B \sqsubseteq 1 \mid B \cdot x \sqsubseteq \top \cdot A\} \ ,$$
$$x \not\backslash A \triangleq \bigvee \{B \sqsubseteq 1 \mid x \cdot B \sqsubseteq A \cdot \top\} \ .$$

Suivant les interprétations données précédemment, le facteur monotype à gauche de x par A est le plus grand monotype B tel que le co-domaine de la restriction du domaine de x à B soit contenu dans A (On a $B \cdot x \sqsubseteq \top \cdot A \Leftrightarrow \top \cdot B \cdot x \sqsubseteq \top \cdot A$; $B \cdot x$ est la restriction du domaine de x à B ; $\top \cdot B \cdot x$ est le co-domaine de cette restriction, représenté par une condition à droite ; et $\top \cdot A$ est la représentation par une condition à droite du monotype A). De manière plus concrète, lorsque x est une relation binaire R, et que A correspond à une partie $\mathcal{Q}$ de $\mathcal{P}$, c'est l'ensemble des éléments de $\mathcal{P}$ dont tous les successeurs par R sont dans $\mathcal{Q}$:

$$\mathcal{Q} \not/ R = \{p \in \mathcal{P} \mid \forall q \in \mathcal{P},\ p\ R\ q \Rightarrow q \in \mathcal{Q}\} \ .$$

Symétriquement, le facteur monotype à droite de x par A est le plus grand monotype B tel que le domaine de la restriction du co-domaine de x à B soit contenu dans A. Dans les relations binaires c'est donc l'ensemble des éléments de $\mathcal{P}$ dont tous les antécédents par R sont dans $\mathcal{Q}$:

$$R\backslash\mathcal{Q} = \{p \in \mathcal{P} \mid \forall q \in \mathcal{P},\ q\ R\ p \Rightarrow q \in \mathcal{Q}\} \quad .$$

On a donc retrouvé au niveau abstrait les notions de plus faible pré- et post-conditions que l'on a, entre autres, en logique de Hoare [Hoa69]. Cette interprétation rend très naturelles les propriétés (4.11), (4.12) et (4.13) de la proposition suivante ; nous détaillons les autres ci-dessous.

Proposition 4.21. *Pour tous $x, y \in X$ et $A, B \sqsubseteq 1$, on a :*

$$x\backslash A = A/\overline{x} \tag{4.8}$$

$$B \sqsubseteq x\backslash A \Leftrightarrow x \cdot B \sqsubseteq A \cdot \top \qquad B \sqsubseteq A/x \Leftrightarrow B \cdot x \sqsubseteq \top \cdot A \tag{4.9}$$

$$x \cdot (x\backslash A) \sqsubseteq A \cdot x \qquad (A/x) \cdot x \sqsubseteq x \cdot A \tag{4.10}$$

$$(x \cdot y)\backslash A = y\backslash(x\backslash A) \qquad A/(x \cdot y) = (A/y)/x \tag{4.11}$$

$$1\backslash A = A/1 = A \tag{4.12}$$

$$x\backslash 1 = 1/x = 1 \tag{4.13}$$

Démonstration. Une fois démontré (4.8), il suffit de démontrer la partie gauche des propriétés.

(4.8) Pour tout $B \sqsubseteq 1$, on a $B \cdot x \sqsubseteq \top \cdot A \Leftrightarrow \overline{x} \cdot B \sqsubseteq A \cdot \top$, par (3.6), (3.3), (4.6), et $\overline{\top} = \top$, qui s'obtient par (3.7). On en déduit l'égalité des bornes supérieures définissant $x\backslash A$ et $A/\overline{x}$.

(4.9) Si $B \sqsubseteq x\backslash A$, alors $x \cdot B \sqsubseteq x \cdot \bigvee \{B \sqsubseteq 1 \mid B,\ x \cdot B \sqsubseteq A \cdot \top\}$; et on a $x \cdot \bigvee \{B \sqsubseteq 1 \mid B,\ x \cdot B \sqsubseteq A \cdot \top\} = \bigvee \{x \cdot B \mid x \cdot B \sqsubseteq A \cdot \top\} \sqsubseteq A \cdot \top$ par la continuité du produit (2.18). L'implication réciproque est immédiate.

(4.10) En choisissant $B = x\backslash A$ dans l'équivalence précédente, on obtient : $x \cdot (x\backslash A) \sqsubseteq A \cdot \top$. Comme on a aussi $x \cdot (x\backslash A) \sqsubseteq x$, on en déduit $x \cdot (x\backslash A) \sqsubseteq (A \cdot \top) \wedge x$. La loi de modularité (3.4) nous permet de conclure : on a $(A \cdot \top) \wedge x \sqsubseteq A(\top \wedge (\overline{A} \cdot x)) = A \cdot A \cdot x = A \cdot x$ (avec (4.6)).

(4.11) Soit $B = y\backslash(x\backslash A)$, on a $y \cdot B \sqsubseteq (x\backslash A) \cdot \top$ par (4.9), puis $x \cdot y \cdot B \sqsubseteq x \cdot (x\backslash A) \cdot \top \sqsubseteq A \cdot x \cdot \top \sqsubseteq A \cdot \top$ par (2.17) et (4.10) ; d'où $B \sqsubseteq (x \cdot y)\backslash A$ par (4.9).

Posons maintenant $B = (x \cdot y)\backslash A$, et $C = (y \cdot B \cdot \top) \wedge 1$, de telle sorte que la loi de modularité (3.8) nous donne $y \cdot B = (y \cdot B) \wedge \top \sqsubseteq C \cdot \top$ ($\star$). Par (4.10), on a $x \cdot y \cdot B \cdot \top \sqsubseteq A \cdot x \cdot y \cdot \top \sqsubseteq A \cdot \top$, puis $x \cdot C \sqsubseteq A \cdot \top$. On en déduit $C \sqsubseteq x\backslash A$ par (4.9), et $C \cdot \top \sqsubseteq (x\backslash A) \cdot \top$. Couplé avec ($\star$), on obtient $y \cdot B \sqsubseteq (x\backslash A) \cdot \top$, d'où $B \sqsubseteq y\backslash(x\backslash A)$ par (4.9).

(4.12) On a $1 \cdot A \sqsubseteq A \cdot \top$, d'où $A \sqsubseteq 1 \backslash A$ par définition. Ensuite, on a $1 \cdot (1 \backslash A) \sqsubseteq A \cdot 1$ par (4.10), d'où $1 \backslash A \sqsubseteq A$.

(4.13) On a $x \backslash 1 \sqsubseteq 1$ par définition ; comme $x \cdot 1 \sqsubseteq 1 \cdot \top$ on a aussi $1 \sqsubseteq x \backslash 1$. ■

Notons qu'ici aussi, la loi de modularité des algèbres relationnelles (3.4) joue un rôle essentiel. Les facteurs monotypes à gauche et à droite ont des rôles essentiellement symétriques (4.8), ce qui permet de se concentrer sur les facteurs monotypes à droite (colonne de gauche). La propriété (4.9) est en fait définissante : elle nous permet d'oublier la définition explicite que l'on a donnée, qui sert principalement à assurer l'existence d'un opérateur satisfaisant (4.9) (voir [DBvdW97] pour une définition par les connections de Galois, qui nécessite une étape intermédiaire que l'on a omise ici). La propriété d'annulation (4.10) correspond à celle des facteurs (4.5) ; on peut l'interpréter comme suit : la restriction du co-domaine de x à l'ensemble des éléments dont tous les antécédents sont dans A est contenue dans la restriction du domaine de x à A. La preuve que nous donnons pour (4.11) nous paraît compliquée ; nous n'en avons cependant pas trouvé de plus simple.

La proposition suivante exprime la décroissance des facteurs monotypes vis-à-vis de leur argument relationnel (x), et leur croissance vis-à-vis de leur argument monotype (A) (respectivement celui sous la barre et celui sur la barre, de façon mnémotechnique).

Proposition 4.22. *Pour tous $x, y \in X$ et $A, B \sqsubseteq 1$, on a :*

$$x \sqsubseteq y \quad \Rightarrow \quad y \backslash A \sqsubseteq x \backslash A,\ A / y \sqsubseteq A / x \ , \tag{4.14}$$

$$A \sqsubseteq B \quad \Rightarrow \quad x \backslash A \sqsubseteq x \backslash B,\ A / x \sqsubseteq B / x \ . \tag{4.15}$$

Démonstration. Pour les facteurs monotypes à droite, il suffit de remarquer que si $x \sqsubseteq y$, alors $y \cdot C \sqsubseteq A \cdot \top$ implique $x \cdot C \sqsubseteq A \cdot \top$, et de façon similaire, que si $A \sqsubseteq B$, alors $x \cdot C \sqsubseteq A \cdot \top$ implique $x \cdot C \sqsubseteq B \cdot \top$. Le cas des facteurs monotypes à gauche est identique. ■

Support abstrait de l'induction

On a maintenant suffisamment d'outils pour donner la définition abstraite de la propriété de support de l'induction :

Définition 4.23 (Support (abstrait) de l'induction)**.** Une relation $x \in X$ *supporte l'induction* si

$$\forall A \sqsubseteq 1,\ x \backslash A \sqsubseteq A \Rightarrow A = 1 \ .$$

La proposition suivante valide cette définition dans le cadre des relations binaires : toute relation binaire bien fondée au niveau concret supporte l'induction au niveau abstrait. Par le théorème 4.7, le support de l'induction

caractérise aussi les relations binaires dont les converses sont fortement normalisantes ; notons que l'on n'a cependant pas de caractérisation *directe* de la notion de forte normalisation au niveau abstrait.

Proposition 4.24. *Dans toute algèbre relationnelle propre, une relation supporte l'induction si et seulement si elle supporte l'induction, au sens de la définition 4.4.*

Démonstration. Soit R une relation binaire ; il suffit de remarquer qu'un monotype A satisfait $R\backslash A \subseteq A$ si et seulement si la partie $\mathcal{Q}$ correspondante à A satisfait (4.2) (en prenant $R = \prec$) : le monotype $R\backslash A$ correspond à l'ensemble $\{p \in \mathcal{P} \mid \forall q \in \mathcal{P},\ q\ R\ p \Rightarrow q \in \mathcal{Q}\}$. ■

Notons que pour toute relation x, le plus petit monotype A satisfaisant $x\backslash A \sqsubseteq A \Rightarrow A$ n'est autre que le plus petit point fixe de la fonction croissante $B \mapsto x\backslash B$; ce plus petit monotype permet donc de caractériser de façon abstraite l'ensemble des éléments accessibles par x.

La définition précédente correspond ainsi exactement à la définition 4.4 ; elle nous fournit un *principe d'induction* : toute relation supportant l'induction fournit une méthode pour démontrer qu'un monotype vaut 1 (1 étant le monotype maximal, il représente la partie pleine, $\mathcal{P}$, ou encore, le prédicat universellement vrai sur $\mathcal{P}$). Le choix de ce monotype correspond donc au choix du prédicat sur lequel on fait l'induction dans les preuves habituelles.

Les trois derniers points de la proposition 4.11 (page 113) s'étendent naturellement au niveau abstrait : on retrouve à ce niveau les propriétés usuelles les concernant la bonne fondation. Les deux premiers points de cette dernière proposition, concernant le produit lexicographique et l'ordre naturel sur les entiers, ne seront pas utiles : de simples récurrences nous permettront de remplacer l'usage que l'on en faisait.

Proposition 4.25. *Soient $x, y \in X$ deux relations.*

1. *Si $x \sqsubseteq y$ et y supporte l'induction, alors x supporte l'induction.*
2. *x supporte l'induction si et seulement si sa fermeture transitive (x^+) la supporte.*
3. *$x \cdot y$ supporte l'induction si et seulement si $y \cdot x$ la supporte.*

Démonstration.

1. Soit $A \sqsubseteq 1$ un monotype tel que $x\backslash A \sqsubseteq A$. Comme $x \sqsubseteq y$, par (4.14), on a $y\backslash A \sqsubseteq x\backslash A \sqsubseteq A$, d'où $A = 1$ puisque x supporte l'induction.
2. Supposons que x supporte l'induction ; soit $A \sqsubseteq 1$ un monotype tel que $x^+\backslash A \sqsubseteq A$ (H). On a :

$$\begin{aligned} x\backslash(x^\star\backslash A) &= x^+\backslash A && \text{(par (4.11))} \\ &= (x^+ \cdot x^\star)\backslash A \\ &= x^\star\backslash(x^+\backslash A) && \text{(par (4.11))} \\ &= x^\star\backslash A\ , && \text{((4.15) et (H))} \end{aligned}$$

d'où $x^\star \backslash A = 1$ puisque x supporte l'induction. On peut finalement conclure : $x^\star$ étant réflexif, $1 = x^\star \backslash A \sqsubseteq A$ par (4.14) et (4.12). L'implication réciproque provient du point précédent : on a $x \sqsubseteq x^+$.

3. Par symétrie, il suffit de prouver l'implication directe ; supposons donc que $x \cdot y$ supporte l'induction, et considérons un monotype $A \sqsubseteq 1$ tel que $(y \cdot x) \backslash A \sqsubseteq A$ (H). On a :

$$\begin{aligned} (x \cdot y) \backslash (y \backslash A) &= (y \cdot x \cdot y) \backslash A && \text{(par (4.11))} \\ &= y \backslash ((y \cdot x) \backslash A) && \text{(par (4.11))} \\ &\sqsubseteq y \backslash A \ . && \text{((4.15) et (H))} \end{aligned}$$

$x \cdot y$ supportant l'induction, on obtient donc $y \backslash A = 1$, ce qui permet de conclure : on a $1 = x \backslash 1 = x \backslash (y \backslash A) = (y \cdot x) \backslash A \sqsubseteq A$, par (4.13), (4.11) et (H). ■

Bonne fondation classique

Les définitions concrètes et abstraites de support de l'induction sont constructives : elles fournissent un principe d'induction permettant de prouver qu'un prédicat est vrai sur tout le domaine considéré. Au contraire, la définition usuelle de bonne fondation est essentiellement classique :

Définition 4.26 (Relation bien fondée)**.** Une relation binaire $\prec$ sur $\mathcal{P}$ est *bien fondée* si toute partie non vide $\mathcal{Q} \subseteq \mathcal{P}$ admet un élément *minimal*, i.e., un élément $p \in \mathcal{Q}$ tel que $\forall q \in \mathcal{Q},\ q \not\prec p$.

Une relation bien fondée permet de montrer qu'un prédicat n'est jamais satisfait ; en effet, on obtient aisément qu'une relation binaire $\prec$ est bien fondée si et seulement si :

$$\forall \mathcal{Q} \subseteq \mathcal{P},\ (\forall p \in \mathcal{Q},\ \exists q \in \mathcal{Q},\ q \prec p) \Rightarrow \mathcal{Q} = \emptyset \ . \tag{4.16}$$

Cette caractérisation permet d'obtenir une définition abstraite : une relation $x \in X$ est *bien fondée* si :

$$\forall A \sqsubseteq 1,\ A \sqsubseteq \top \cdot A \cdot x \Rightarrow A = \bot \ , \tag{4.17}$$

ou de manière équivalente [DBvdW97], sans utiliser les monotypes, si :

$$\forall y \in X,\ y \sqsubseteq y \cdot x \Rightarrow y = \bot \ . \tag{4.18}$$

Dans les deux cas, on s'aperçoit que l'on a en fait une méthode de preuve co-inductive pour prouver qu'un monotype ou une relation est vide (les génératrices étant respectivement les fonctions croissantes $B \mapsto (\top \cdot B \cdot x) \wedge 1$ et $y \mapsto y \cdot x$). L'existence de techniques modulo pour cette méthode co-inductive n'est donc pas à exclure (de même que des techniques modulo pour le support de l'induction). Nous laissons toutefois cette idée pour de futures recherches.

Doornbos, Backhouse et van der Woude [DBvdW97] montrent que toute relation supportant l'induction est bien fondée, mais en utilisant l'hypothèse de distributivité du treillis, que l'on ne fait pas. Nous n'avons pas essayé de prouver cette implication sans l'aide de cette hypothèse. La réciproque est vraie lorsque l'algèbre est complémentée (algèbre de Boole), c'est à dire dans des cadres classiques, où l'on a une *négation* suffisamment forte. Nous n'utiliserons que la notion constructive de support de l'induction ; du fait de son aspect classique, il n'est pas clair que nous puissions obtenir l'ensemble des résultats qui suivent en utilisant la notion de bonne fondation. Nous laissons cette question pour des recherches ultérieures.

Notons enfin qu'il existe d'autre cadres permettant de caractériser la notion de terminaison de façon abstraite : Georg Struth [Str06] considère par exemple les ω-algèbres [Coh00] (des algèbres de Kleene munies d'une opération d'itération infinie). La technique de preuve associée aux objets bien fondés y est également co-inductive, et correspond à peu près au principe co-inductif établi par (4.18). Nous reviendrons plus bas sur les limitations de cette approche.

Principe d'induction spécifique aux diagrammes de commutation

Dans la suite, nous n'utiliserons le principe d'induction abstraite que sous une forme bien particulière, pour fermer des diagrammes de commutation. Le théorème suivant capture cette forme particulière d'induction, que nous expliquons ci-dessous ; la preuve de ce théorème provient de la preuve abstraite du lemme de Newman que donnent Doornbos et al. [DBvdW97], mais l'idée d'isoler ce théorème pour en faire une méthode générique nous est propre.

Théorème 4.27 (Principe d'induction bien fondée, pour les diagrammes)**.** *Soient $x, y, z, t \in X$ quatre relations. Si t supporte l'induction, et si pour tout monotype $A \sqsubseteq 1$ tel que $x \cdot A \cdot y \sqsubseteq z$, on a $x \cdot (t\backslash A) \cdot y \sqsubseteq z$, alors $x \cdot y \sqsubseteq z$.*

Démonstration. Considérons le monotype suivant : $A \triangleq ((x\backslash z)/y) \wedge 1$. Pour tout monotype $B \sqsubseteq 1$, on a :

$$\begin{aligned} B \sqsubseteq A \quad &\Leftrightarrow \quad B \sqsubseteq (x\backslash z)/y \\ &\Leftrightarrow \quad B \cdot y \sqsubseteq x\backslash z && \text{(par (4.4))} \\ &\Leftrightarrow \quad x \cdot B \cdot y \sqsubseteq z \ . && \text{(par (4.4))} \end{aligned}$$

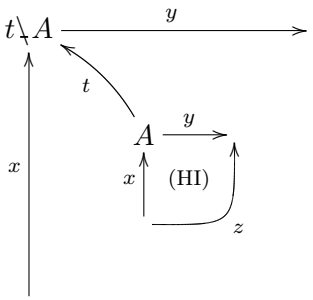
En particulier, en choisissant $B = A$, on obtient :

$$x \cdot A \cdot y \sqsubseteq z \ . \qquad \text{(HI)}$$

Par hypothèse, (HI) implique $x \cdot (t\backslash A) \cdot y \sqsubseteq z$, c'est-à-dire $t\backslash A \sqsubseteq A$, en choisissant $B = t\backslash A$ dans le raisonnement précédent. Comme t supporte l'induction, on en déduit $A = 1$, ce qui rend (HI) équivalente au résultat attendu : $x \cdot y = x \cdot A \cdot y \sqsubseteq z$. ∎

Intuitivement, le monotype A correspond à l'ensemble des « coins » à partir desquels les diagrammes se ferment : c'est ce qu'exprime (HI), qui n'est rien d'autre que l'hypothèse d'induction. La méthode inductive nous permet de montrer a posteriori que $A = 1$, c'est-à-dire que les diagrammes se ferment à partir de n'importe quel coin.

Pour comprendre l'étape inductive demandée par ce principe d'induction (si $x \cdot A \cdot y \sqsubseteq z$ alors $x \cdot (t \backslash A) \cdot y \sqsubseteq z$), il faut interpréter le monotype $t \backslash A$ comme une version désactivée de A, qui attend de « passer » par t pour s'activer (formellement la propriété (4.10) nous indique que $t \cdot (t \backslash A) \sqsubseteq A \cdot t$). On doit ainsi fermer le grand diagramme suivant, sachant que tout diagramme « plus petit » se ferme, un diagramme étant plus petit lorsque son coin est relié par t au diagramme initial :



Nous reviendrons dans la section suivante sur l'intuition de cette étape inductive en nous appuyant sur la preuve du théorème 4.28.

La quantification universelle sur le monotype A dans l'énoncé peut paraître étrange ; elle permet juste de « cacher l'implantation » du théorème : la définition précise du monotype A utilisé dans la preuve est inutile, il suffit en pratique de savoir que les diagrammes se ferment à partir de A.

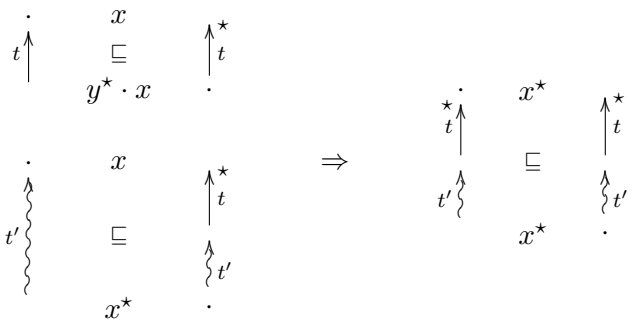
Notons finalement que pour appliquer ce principe d'induction, la première chose à faire est de « choisir le coin » d'où l'on souhaite démarrer l'induction : si l'on désire par exemple fermer un diagramme de la forme $x \cdot y \cdot y' \sqsubseteq z$, on a au moins quatre possibilités pour placer le coin : $A \cdot x \cdot y \cdot y'$, $x \cdot A \cdot y \cdot y'$, $x \cdot y \cdot A \cdot y'$ et $x \cdot y \cdot y' \cdot A$. Ce choix dépend bien entendu du résultat à prouver. Comme l'explique Struth [Str06], la principale limitation des ω-algèbres provient du fait que l'on ne peut raisonner sur des diagrammes par induction bien fondée qu'en plaçant le coin à l'une des extrémités du diagramme (premier et dernier choix dans l'exemple précédent). Par exemple, on ne peut pas obtenir dans ce cadre le lemme de Newman ou la généralisation que nous proposons (théorème 4.14), dont les preuves s'obtiennent par induction à partir du coin supérieur gauche.

4.2.2 Preuve calculatoire du résultat de commutation

Nous utilisons maintenant les résultats précédents pour établir le théorème 4.14 dans une algèbre relationnelle quelconque. La preuve que l'on

obtient suit exactement les différentes étapes de la preuve que l'on a donnée à la section précédente, mais s'effectue de façon très algébrique : une fois fixé le monotype correspondant au prédicat de l'induction bien fondée, il suffit de « calculer », en utilisant les propriétés précédemment établies.

Le théorème est donné ci-dessous ; nous donnons la preuve de façon très détaillée (seuls l'usage de la continuité du produit (2.18), et en particulier le fait qu'il respecte l'ordre partiel (2.17), sont implicites) ; nous l'expliquons et la comparons à celle du théorème 4.14 dans un second temps. L'énoncé de ce théorème peut être donné de façon diagrammatique comme suit : pour toutes relations x, y, t, t' telles que $y \sqsubseteq x$, si $\overline{y}^+ \cdot t^+$ supporte l'induction, alors les deux diagrammes de gauche ci-dessous impliquent celui de droite :



On retrouve le théorème 4.14 en prenant R et S pour x et y, $\leftarrow$ pour t, et la relation identité (I) pour t', ce qui rend la seconde hypothèse de commutation triviale. L'ajout de cette quatrième relation permettra de passer facilement à la simulation faible dans la section suivante : elle sera instanciée par des relations de la forme $\overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow}$.

Théorème 4.28. *Soient $x, y, t, t' \in X$ quatre relations,*

$$\textit{Si} \begin{cases} y \sqsubseteq x \ , \\ \overline{y}^+ \cdot t^+ \textit{ supporte l'induction,} \\ t \cdot x \sqsubseteq y^\star \cdot x \cdot t^\star \ , \\ t' \cdot x \sqsubseteq x^\star \cdot t' \cdot t^\star \ , \end{cases} \qquad \textit{alors } t' \cdot t^\star \cdot x^\star \sqsubseteq x^\star \cdot t' \cdot t^\star \ .$$

Démonstration. Posons $t'' \triangleq t' \cdot t^\star$, nous allons montrer $t'' \cdot x \sqsubseteq x^\star \cdot t''$, ce qui mène au résultat attendu par (2.20). On applique pour cela le principe d'induction spécifique aux diagrammes (théorème 4.27) : soit $A \sqsubseteq 1$ un monotype tel que :

$$t'' \cdot A \cdot x \sqsubseteq x^\star \cdot t'' \ . \tag{HI}$$

Il nous faut montrer $t'' \cdot ((\overline{y}^+ \cdot t^+) \backslash A) \cdot x \sqsubseteq x^\star \cdot t''$. On procède par récurrence, en utilisant le prédicat suivant :

$$\Phi(n) : \ t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^+) \backslash A) \cdot x \sqsubseteq x^\star \cdot t'' \ .$$

– le cas $n = 0$ est fourni par la seconde hypothèse de commutation :

$$t' \cdot t^0 \cdot ((\overline{y}^+ \cdot t^+)\backslash A) \cdot x \sqsubseteq t' \cdot x \sqsubseteq x^\star \cdot t'' \ ,$$

– supposons $\Phi(n)$, et montrons $\Phi(n+1)$:

$$\begin{array}{lll}
(a) & t' \cdot t^n \cdot t \cdot ((\overline{y}^+ \cdot t^+)\backslash A) \cdot x & \\
& = t' \cdot t^n \cdot t \cdot (t\backslash((\overline{y}^+ \cdot t^\star)\backslash A)) \cdot x & \text{(par (4.11))} \\
(b) & \sqsubseteq t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^\star)\backslash A) \cdot t \cdot x & \text{(par (4.10))} \\
(c) & \sqsubseteq t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^\star)\backslash A) \cdot y^\star \cdot x \cdot t^\star & \\
& & \text{(première hypothèse de commutation)}
\end{array}$$

Il suffit donc de montrer $t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^\star)\backslash A) \cdot y^k \cdot x \sqsubseteq x^\star \cdot t''$, pour tout $k \in \mathbb{N}$. On considère les deux cas suivants :

– soit $k = 0$ et on a :

$$\begin{array}{ll}
t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^\star)\backslash A) \cdot y^0 \cdot x & \\
\sqsubseteq t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^+)\backslash A) \cdot x & \text{(par (4.14))} \\
\sqsubseteq x^\star \cdot t'' & \text{(par récurrence : } \Phi(n)\text{)}
\end{array}$$

– soit $k > 0$, auquel cas on a :

$$\begin{array}{ll}
t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^\star)\backslash A) \cdot y \cdot y^{k-1} \cdot x & \\
\sqsubseteq t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^\star)\backslash A) \cdot ((\overline{y}^+ \cdot t^\star)\backslash A) \cdot y^k \cdot x & \text{(par (4.6))} \\
\sqsubseteq t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^+)\backslash A) \cdot (\overline{y}^+\backslash A) \cdot y^k \cdot x & \text{(par (4.14))} \\
\sqsubseteq t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^+)\backslash A) \cdot (A/y^+) \cdot y^k \cdot x & \text{(par (4.8))} \\
\sqsubseteq t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^+)\backslash A) \cdot y \cdot (A/y^\star) \cdot y^{k-1} \cdot x & \text{((4.11) et (4.10))} \\
\sqsubseteq t' \cdot t^n \cdot ((\overline{y}^+ \cdot t^+)\backslash A) \cdot x \cdot (A/y^\star) \cdot y^{k-1} \cdot x & (y \sqsubseteq x) \\
\sqsubseteq x^\star \cdot t'' \cdot (A/y^\star) \cdot y^{k-1} \cdot x & \text{(par récurrence : } \Phi(n)\text{)} \\
\sqsubseteq x^\star \cdot x^\star \cdot t'' & (\Psi(k-1)\text{, prouvé ci-dessous}) \\
\sqsubseteq x^\star \cdot t'' \ . & (x^\star \text{ est transitif})
\end{array}$$

L'avant-dernière étape repose sur l'inégalité suivante, que nous démontrons a posteriori, par récurrence sur k :

$$\forall k \in \mathbb{N},\ \Psi(k) :\ t'' \cdot (A/y^\star) \cdot y^k \cdot x \sqsubseteq x^\star \cdot t'' \ ,$$

– pour $k = 0$, on a :

$$\begin{array}{lll}
 & t'' \cdot (A/y^\star) \cdot y^0 \cdot x \sqsubseteq t'' \cdot (A/1) \cdot x & \text{(par (4.14))} \\
(*) & \sqsubseteq t'' \cdot A \cdot x & \text{(par (4.12))} \\
 & \sqsubseteq x^\star \cdot t'' & \text{(par (HI))}
\end{array}$$

– supposant $\Psi(k)$, montrons $\Psi(k+1)$:

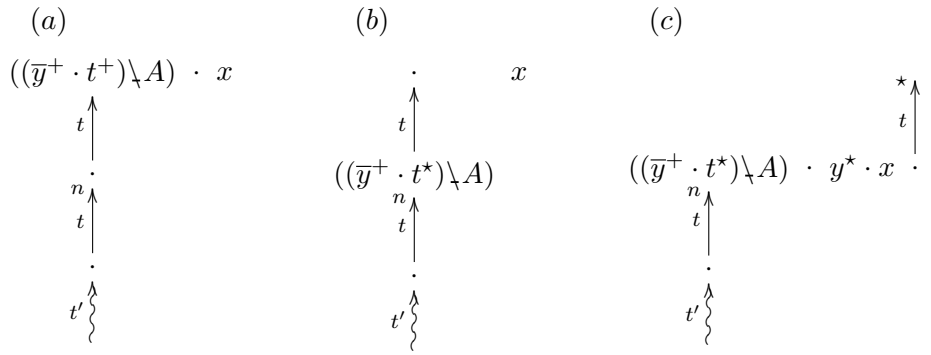
$$
\begin{array}{lll}
 & t'' \cdot (A/y^\star) \cdot y^{k+1} \cdot x & \\
 & = t'' \cdot (A/y^\star) \cdot (A/y^\star) \cdot y \cdot y^k \cdot x & \text{(par (4.6))} \\
 & \sqsubseteq t'' \cdot (A/1) \cdot (A/y^+) \cdot y \cdot y^k \cdot x & \text{(par (4.14))} \\
(*) & \sqsubseteq t'' \cdot A \cdot (A/y^+) \cdot y \cdot y^k \cdot x & \text{(par (4.14))} \\
 & \sqsubseteq t'' \cdot A \cdot y \cdot (A/y^\star) \cdot y^k \cdot x & \text{((4.11) et (4.10))} \\
 & \sqsubseteq x^\star \cdot t'' \cdot (A/y^\star) \cdot y^k \cdot x & (y \sqsubseteq x \text{ et (HI)}) \\
 & \sqsubseteq x^\star \cdot x^\star \cdot t'' & \text{(par récurrence : } \Psi(k)) \\
 & \sqsubseteq x^\star \cdot t'' & (x^\star \text{ est transitif})
\end{array}
$$

■

Mise en parallèle avec la preuve « concrète »

Cette preuve est sensiblement plus longue que celle donnée à la section précédente, mais cela est principalement dû au niveau de détail auquel on s'est astreint ici. S'il faut initialement une certaine gymnastique pour s'habituer à cette présentation des diagrammes de commutation en cours de fermeture, et des preuves par induction bien fondée, nous pensons tout comme les auteurs de [DBvdW97] que cette approche est vraiment simplificatrice. Nous essayons de justifier ce point de vue ci-dessous, en donnant les intuitions sur lesquelles reposent cette preuve essentiellement calculatoire [2].

Notons tout d'abord que toute étape de cette preuve peut être interprétée de façon diagrammatique : les étapes marquées (a) (b) et (c) peuvent par exemple être représentées ainsi :



2. Cette preuve suit par sa méthodologie la preuve du lemme de Newman qui est proposée dans [DBvdW97]. Ce résultat étant bien plus simple, il peut être utile de s'y référer. Les preuves des lemmes 4.40, 4.43 et 4.48, ainsi que du théorème 4.57, s'obtiendront selon la même méthode. Comme elles sont plus simples, elles peuvent aussi aider pour la compréhension de celle présentée ici.

Au fur et à mesure des calculs on « tord » en quelque sorte l'expression, jusqu'à avoir fermé le diagramme.

Comme indiqué précédemment, le monotype A correspond à l'ensemble des « coins supérieur gauche » à partir desquels tout diagramme de la forme $t'' \cdot x$ se ferme en $x^\star \cdot t''$; c'est exactement ce qu'exprime l'hypothèse d'induction (HI). La méthode de preuve par induction nous demande ensuite de fermer tout diagramme de la forme $t'' \cdot ((\overline{y}^+ \cdot t^+) \backslash A) \cdot x$; il faut alors comprendre le monotype $((\overline{y}^+ \cdot t^+) \backslash A)$ comme l'hypothèse d'induction, en position « inactive » : il faut que ce monotype franchisse au moins t, puis $\overline{y}$, avant de s'activer en un A, qui permettra de fermer tout diagramme potentiel, par (HI). L'étape entre (a) et (b) consiste justement à faire glisser $((\overline{y}^+ \cdot t^+) \backslash A)$ le long de la transition t ; ce faisant, l'hypothèse est légèrement transformée : elle ne doit maintenant franchir que $\overline{y}^+ \cdot t^\star$. Plus intuitivement, elle doit toujours franchir une étape $\overline{y}$ avant de devenir active, mais elle peut encore passer autant de transitions t que nécessaire.

L'hypothèse d'induction est ainsi « activée » à deux endroits dans la preuve, marqués d'une astérisque $(*)$; bien que ce soit plutôt le rôle de la propriété (4.10), c'est (4.12) qui est utilisée dans les deux cas. Remarquons aussi l'utilisation de (4.6), qui nous permet de dupliquer l'hypothèse d'induction, avant de la faire glisser dans deux parties différentes du diagramme (cas $k > 0$ de l'étude de cas dans la seconde partie de la première récurrence), ou de la sauvegarder avant de l'activer pour fermer un diagramme (cas inductif de la seconde récurrence).

Cette preuve suit de très près celle du théorème 4.14 : l'induction sur la relation $\overline{y}^+ \cdot t^+$ correspond à la première composante de l'induction lexicographique que l'on fait dans la preuve de la section précédente ; la seconde composante de cette induction se traduit par la récurrence sur n, avec le prédicat Φ ; la récurrence sur k avec le prédicat Ψ correspond elle à la construction par récurrence interne de la séquence $(p_i)_{i \leq k}$ dans la preuve initiale. Notons que l'on ne retrouve pas de trace dans cette nouvelle preuve de la variable u, introduite de manière artificielle dans la preuve initiale pour pouvoir transporter l'hypothèse d'induction : les lois régissant le comportement du monotype correspondant au prédicat inductif capturent de façon plus naturelle le comportement de cette hypothèse.

L'avantage principal de cette méthode est de ne plus avoir à manipuler les processus, même lors des inductions bien fondées. Cela permet d'une part de mieux comprendre la façon dont peut s'utiliser l'hypothèse d'induction dans une preuve donnée, et d'autre part d'obtenir des preuves plus facilement formalisables dans des assistants de preuve. Nous avons en effet formalisé dans l'assistant de preuve Coq [CH84] l'ensemble des résultats de [Pou05b] (qui contient en partie les résultats de commutation présentés ici, mais dans le cas d'une algèbre relationnelle propre), et une partie non négligeable de ces développements [Pou05c] consiste en des manipulations peu intéressantes

de l'existentielle définissant la composition relationnelle, afin de faire passer les inductions bien fondées.

Nous envisageons de reprendre cette formalisation, en suivant l'approche proposée ici, ce qui devrait grandement simplifier la tâche... L'enjeu principal serait de réussir à définir un ensemble de lemmes et de tactiques permettant d'effectuer les calculs progressifs d'inclusion aussi simplement que sur le papier. Le principe d'induction que nous avons isolé va dans cette direction (sa formulation est très « orientée Coq »), nous reviendrons sur cette idée au chapitre 7, page 7.

Remarque 4.29 (Diagrammes décroissants abstraits). Le théorème 4.16 des diagrammes décroissants [vO94] peut être étendu trivialement au cas d'une algèbre relationnelle quelconque : sa preuve ne mentionne jamais les termes. L'ensemble d'étiquettes et l'ordre partiel bien fondé qui y sont supposés restent cependant « concrets » (les abstraire ne fait aucun sens). Il n'est donc pas évident que l'on puisse obtenir des résultats tel que le théorème 4.28 à l'aide de cette version abstraite des diagrammes décroissants : il faudrait être capable de déduire d'une hypothèse de terminaison abstraite (la relation $\overline{y}^+ \cdot t^+$ supporte l'induction), un étiquetage et un ordre partiel bien fondé concrets. Nous laissons cette question pour de futures recherches : il serait clairement utile de pouvoir combiner l'approche sémantique fournie par les diagrammes décroissants, à notre approche calculatoire, plus facilement mécanisable.

4.3 Application à la bisimulation faible

Le théorème 4.28 mène presque directement au théorème suivant, qui en fait une technique modulo pour la simulation faible, comme pour la bisimulation faible. Nous montrons comment l'enrichir par des techniques plus standard : modulo faible à droite, modulo contexte, etc... puis nous discutons des problèmes de modularité dont souffrent ces techniques.

4.3.1 Résultat pur

Théorème 4.30. *Soient x, y deux relations.*

$$\textit{Si} \begin{cases} y \sqsubseteq x \\ \overline{y}^+ \cdot \overset{\tau}{\Leftarrow} \ \textit{supporte l'induction} \\ \overset{\tau}{\leftarrow} \cdot x \sqsubseteq y^\star \cdot x \cdot \overset{\widehat{\tau}}{\Leftarrow} \\ \forall a \in \mathcal{L}^{\mathrm{v}},\ \overset{a}{\leftarrow} \cdot x \sqsubseteq x^\star \cdot \overset{a}{\Leftarrow} \end{cases} \quad \textit{alors } x^\star \textit{ est une simulation faible.}$$

Si de plus x est symétrique, $x^\star$ est une bisimulation faible.

Démonstration. En appliquant le théorème 4.28 à x, y, $t = \overset{\tau}{\leftarrow}$, et $t' = 1$, on obtient tout d'abord la commutation de x et $\overset{\tau}{\rightarrow}$:

$$\overset{\widehat{\tau}}{\Leftarrow} \cdot x^\star \sqsubseteq x^\star \cdot \overset{\widehat{\tau}}{\Leftarrow} \tag{4.19}$$

Soit $a \in \mathcal{L}^{\mathrm{v}}$, en combinant la seconde hypothèse de commutation avec (4.19), on a $\overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot x \sqsubseteq x^\star \cdot \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot \overset{\tau}{\leftarrow}^\star$, ce qui nous permet d'appliquer une seconde fois le théorème 4.28 à x, y, $t = \overset{\tau}{\leftarrow}$, et $t' = \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow}$; et d'obtenir ainsi :

$$\overset{a}{\Leftarrow} \cdot x^\star \sqsubseteq x^\star \cdot \overset{a}{\Leftarrow} \ . \tag{4.20}$$

Finalement, (4.19) et (4.20) indiquent que $x^\star$ est une **w**-simulation. ■

Notons que modulo transitivité est complètement autorisée lors des offres visibles : l'hypothèse de support de l'induction (de terminaison) est très précise : elle ne contraint que la partie de x qui est utilisée en position de transitivité à gauche, lors des offres silencieuses.

Comme pour le théorème 4.14, l'utilisation d'une relation auxiliaire y permet de découpler l'argument de terminaison de celui de simulation : ce théorème permet d'obtenir des simulations qui ne satisfont pas nécessairement l'argument de terminaison, qui n'est requis que pour une partie de la simulation finale ($x^\star$). Nous verrons dans la section suivante un résultat fondé sur une autre hypothèse de terminaison, mais qui ne permet pas cette séparation ; cela rend la symétrisation du résultat plus délicate, et pose réellement la question de son applicabilité : obtenir une simulation non triviale satisfaisant une hypothèse de terminaison n'est pas facile a priori, puisque cela implique que cette hypothèse de terminaison soit préservée le long du jeu de simulation (cf. remarque 4.42).

Toujours comme dans le cadre de la commutation (page 116), ce théorème peut s'interpréter ainsi : pour montrer qu'une relation binaire symétrique est contenue dans la bisimilarité faible, montrons que c'est une simulation faible modulo transitivité, puis collectons dans une relation S l'ensemble des couples utilisés en position de transitivité, lors des offres silencieuses ; si $S^+ \cdot \overset{\tau}{\Rightarrow}$ normalise fortement, c'est bon, sinon, c'est que l'on a triché.

4.3.2 Résultat enrichi

Une première solution, pour pouvoir combiner cette techniques à des techniques plus standard, consiste à en dériver des préordres contrôlés ; l'hypothèse de terminaison exprime alors très clairement l'idée qu'un préordre contrôlé ne doit pas être capable d'annuler des offres silencieuses, en « remontant » le temps que les transitions silencieuses représentent :

Théorème 4.31. *Pour toute relation $\succ$ contenue dans la bisimilarité faible ($\succ \sqsubseteq \approx$), telle que $\prec^+ \cdot \overset{\tau}{\Leftarrow}$ supporte l'induction, $\succ$ est une relation contrôlée.*

Démonstration. Soit $z \in X$ une relation telle que $z \rightsquigarrow_{\mathbf{w}_t} \succ^\star \cdot z$; il nous faut montrer que $(\approx \vee z)^\star$ est une $\mathbf{w}_t$-simulation. Il suffit pour cela d'appliquer le théorème 4.30 à $y = \succ$ et $x = \approx \vee z$. Les deux premières hypothèses sont immédiates à vérifier, les deux dernières proviennent de l'hypothèse de progression faite sur z, et du fait que $\approx$ est une $\mathbf{w}$-simulation (pour la dernière on utilise aussi l'inégalité $(\succ^\star \cdot z)^\star \sqsubseteq x^\star$). La relation $x^\star$ est donc une simulation faible, ainsi que $(x \vee \approx)^\star = (x^\star \vee \approx)^\star$. ■

Les corollaires 3.49, 3.50 ou 3.52 nous permettent ensuite d'utiliser de telles relations pour réécrire le processus en position de transitivité : on autorise faible modulo faible à condition que l'ensemble des couples faiblement bisimilaires réellement utilisés satisfassent l'argument de terminaison. Le cas du corollaire 3.50 donne par exemple :

Corollaire 4.32. *Soient $\succ \sqsubseteq \approx$ une relation telle que $\prec^+ \cdot \overset{\tau}{\Leftarrow}$ supporte l'induction. Toute relation symétrique $x \in X$ satisfaisant les deux diagrammes ci-dessous est majorée par la bisimilarité faible.*

$$\begin{array}{ccc} \cdot & x & \\ {\scriptstyle\tau}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle\widehat{\tau}} \\ & \succ^\star \cdot x^= \cdot \approx & \cdot \end{array} \qquad \begin{array}{ccc} \cdot & x & \\ {\scriptstyle a}\downarrow & \sqsubseteq & \Downarrow{\scriptstyle a} \\ & (x \vee \approx)^\star & \cdot \end{array}$$

Ce corollaire nous demande de prouver $\succ \sqsubseteq \approx$, avant de pouvoir prouver $x \sqsubseteq \approx$: il nous faut établir quelques résultats préliminaires de bisimulation faible ($\succ$) avant de pouvoir les utiliser pour le résultat principal (x). Le théorème 4.30 nous permet cependant de faire d'une pierre deux coups, en prouvant en même temps $\succ \sqsubseteq \approx$ et $x \sqsubseteq \approx$ ($y = \succ$) : la faible bisimilarité de certains des couples utilisés en position de transitivité peut être prouvée en s'appuyant sur celle des couples du résultat principal.

Pour pouvoir combiner cette technique dans toute sa généralité à d'autres techniques standard, il nous faut tout d'abord l'exprimer en tant que fonction correcte. On introduit pour cela la famille suivante de fonctions, qui dépendent d'une relation qui servira à porter l'hypothèse de terminaison :

Définition 4.33 (Fermeture transitive contrainte à gauche)**.** Soit $\succ \in X$ une relation, on définit comme suit la *fonction $t_\succ$ de fermeture transitive contrainte à gauche* :

$$\begin{aligned} t_\succ : X &\to X \\ x &\mapsto (x \wedge \succ)^\star \cdot x \end{aligned}$$

On obtient alors la proposition suivante :

Proposition 4.34. *Soit $\succ \in X$ une relation. Si $\prec^+ \cdot \overset{\tau}{\Leftarrow}$ supporte l'induction, alors la fonction $t_\succ$ est correcte via $\mathrm{id}_X^\star$ pour $\mathbf{w}_t$.*

Démonstration. Soit $x \in X$ une relation telle que $x \rightsquigarrow_{\mathbf{w}_\mathrm{t}} t_\succ(x)$. En appliquant le théorème 4.30 à x et $y = x \wedge \succ$, on obtient que $x^\star$ est une $\mathbf{w}$-simulation, c'est donc en particulier une $\mathbf{w}_\mathrm{t}$-simulation (comme on a $t_\succ(x)^\star = x^\star$, le cas d'une offre visible dans la définition de $\mathbf{w}_\mathrm{t}$ correspond bien à celui du théorème 4.30). ■

Par le lemme 3.24, la correction de $t_\succ$ pour $\mathbf{w}$ via $\mathrm{id}_X^\star$ en découle sous la même hypothèse de terminaison. Notons aussi que lorsque l'on utilise $\mathbf{w}_\mathrm{t}$, on ne perd pas la transitivité sur les actions visibles, puisque $t_\succ(x)^\star = x^\star$. Par le théorème 2.52, il ne reste ensuite qu'à prouver la compatibilité avec $t_\succ$ des techniques modulo que l'on souhaite rajouter :

Lemme 4.35. *Soit $\succ \in X$ une relation. Chacune des fonctions suivantes est compatible avec $t_\succ$:*

1. *la fonction de fermeture réflexive* ($\mathrm{id}_X^= : \ x \mapsto x^=$) ;
2. *la post-composition par une relation réflexive z* ($x \mapsto x \cdot z$) ;
3. *toute clôture $\mathcal{C}$ compatible avec $\mathrm{id}_X^\star$, et telle que $\mathcal{C}(\succ) \sqsubseteq \succ$.*

Démonstration. Soit $x \in X$ une relation quelconque.

1. $t_\succ(x)^= = ((x \wedge \succ)^\star \cdot x) \vee 1 = (x \wedge \succ)^\star \cdot x^= \sqsubseteq (x^= \wedge \succ)^\star \cdot x^= = t_\succ(x^=)$.
2. $t_\succ(x) \cdot z = (x \wedge \succ)^\star \cdot x \cdot z \sqsubseteq (x \cdot z \wedge \succ)^\star \cdot x \cdot z = t_\succ(x \cdot z)$.
3. On a $\mathcal{C}(t_\succ(x)) = \mathcal{C}((x \wedge \succ)^\star \cdot x) \sqsubseteq \mathcal{C}(x \wedge \succ)^\star \cdot \mathcal{C}(x) \sqsubseteq (\mathcal{C}(x) \wedge \mathcal{C}(\succ))^\star \cdot \mathcal{C}(x)$, et on conclut par l'hypothèse entre $\mathcal{C}$ et $\succ$. ■

Dans le troisième point, on retrouve le fait que la clôture $\mathcal{C}$ doive préserver la relation qui porte l'argument de terminaison ($\succ$). C'est exactement le même phénomène que dans le cas du théorème 3.47 ; l'hypothèse de compatibilité avec $\mathrm{id}_X^\star$ sera par ailleurs satisfaite par les fonctions de fermeture par contexte (proposition 5.14).

On obtient finalement le théorème suivant, dont la preuve se résume à prouver la symétrie de la fonction témoin. Sans le théorème 2.52 il nous faudrait reprendre intégralement la preuve du théorème 4.28, en rajoutant dès le départ (i.e. dans le monotype représentant le prédicat inductif) tous les raffinements apportés ici...

Théorème 4.36. *Soient $\succ \in X$ une relation et $\mathcal{C}$ une clôture compatible avec $\mathrm{id}_X^\star$ et $\mathbf{w}$ (resp. $\mathbf{w}_t$) telle que $\mathcal{C}(\succ) \sqsubseteq \succ$. Si $\prec^+ \cdot \overset{\tau}{\Leftarrow}$ supporte l'induction, alors la fonction*

$$x \mapsto ((\mathcal{C}(x) \vee \approx) \wedge \succ)^\star \cdot \mathcal{C}(x) \cdot \approx$$

est correcte pour $\mathbf{w}$ (resp. $\mathbf{w}_t$) via une fonction symétrique.

Démonstration. Soit $f = \mathcal{C} \mathbin{\hat{\cdot}} \widehat{\approx}$. Par les propositions 2.50 et 3.17(2), et par le lemme 4.35, f est compatible avec $\mathbf{w}$ (resp. $\mathbf{w}_\mathrm{t}$) et $t_\succ$. D'après la proposition 4.34, $t_\succ$ est correcte pour $\mathbf{w}_\mathrm{t}$ via $\mathrm{id}_X^\star$ (ainsi que pour $\mathbf{w}$ via la même fonction, par le lemme 3.24). On peut donc appliquer le théorème 2.52 : $t_\succ \circ f$ est correcte pour $\mathbf{w}$ (resp. $\mathbf{w}_\mathrm{t}$) via $\mathrm{id}_X^\star \circ f^\omega$.

Par (3.17), on a $f^\omega = f$, la fonction témoin obtenue est donc en fait $f^\star$; on la complète ensuite par le lemme 2.58 en $\widehat{\approx} \mathbin{\hat{\cdot}} f^\star$. Cette dernière fonction est symétrique : elle est égale à $(\mathcal{C} \vee \widehat{\approx})^\star$ par (2.22). ■

En reformulant ce résultat en termes concrets dans une algèbre relationnelle propre, et en caractérisant les relations supportant l'induction à l'aide de la notion de normalisation forte, on obtient immédiatement le corollaire suivant :

Corollaire 4.37. *Soient $\succ$ une relation binaire, et $\mathcal{C}$ une clôture compatible avec $\mathrm{id}_X^\star$ et $\mathbf{w}$ telle que $\mathcal{C}(\succ) \subseteq \succ$. Si $\succ^+ \cdot \overset{\tau}{\Rightarrow}$ est fortement normalisante, alors toute relation binaire R symétrique et satisfaisant le diagramme suivant est contenue dans la bisimilarité faible.*

$$\begin{array}{ccc} \cdot & R & \\ {\scriptstyle\alpha}\downarrow & \subseteq & \Downarrow{\scriptstyle\widehat{\alpha}} \\ & ((\mathcal{C}(R) \cup \approx) \cap \succ)^\star \cdot \mathcal{C}(R) \cdot \approx & \cdot \end{array}$$

Comme pour le théorème 4.30, modulo transitivité y est autorisée à condition de respecter une condition de terminaison sur les couples réellement utilisés en position de transitivité, et on a de plus droit à modulo contexte, et modulo $\approx$.

4.3.3 Commentaires

Sur l'argument de terminaison

Il existe effectivement des relations $\succ$ telles que $\succ^+ \cdot \overset{\tau}{\Rightarrow}$ normalise fortement ; la relation universelle, restreinte à droite aux processus *stables* est un exemple assez trivial :

$$\left\{ \langle p, q \rangle \mid p, q \in \mathcal{P},\ \forall q' \in \mathcal{P},\ q \overset{\tau}{\not\to} q' \right\} .$$

On retrouve ainsi un résultat déjà présent dans [SM92], et qui peut facilement se prouver directement : on peut réécrire le processus de gauche dans les jeux de bisimulation faible en utilisant de la bisimilarité faible, à condition de le réécrire en un processus stable.

Bien que cet exemple montre qu'il n'est pas nécessaire que $\succ$ ou $\overset{\tau}{\to}$ termine, le cas le plus fréquent nous semble cependant être celui où l'une de ces deux relations termine relativement à l'autre. Nous étudierons à la

section 4.5 les conséquences du théorème 4.30 dans le cas où $\xrightarrow{\tau}$ termine (LTS réactif). Cette hypothèse sera satisfaite dans l'exemple développé au chapitre 6, et on aura une situation où la terminaison de $\succ^+ \cdot \stackrel{\tau}{\Rightarrow}$ provient effectivement de celle des transitions silencieuses (lemme 6.43) : on y exhibera une relation $\succ$ (qui ne termine pas, étant symétrique), et une mesure $t : \mathcal{P} \to \mathbb{N}^3$ telle que l'on ait, pour tous processus p, q :

$$p \xrightarrow{\tau} q \Rightarrow t(p) >_{lex} t(q) \qquad \text{et} \qquad p \succ q \Rightarrow t(p) = t(q) \ ,$$

cette mesure assurant donc la terminaison de $\xrightarrow{\tau}$ relativement à $\succ$.

Bien que nous n'ayons pas d'exemple utile en pratique, une situation symétrique, où c'est la relation $\succ$ qui terminerait relativement à $\xrightarrow{\tau}$, nous paraît parfaitement imaginable.

Sur le manque de modularité

Comme le montre le contre-exemple suivant, l'union de deux relations satisfaisant l'hypothèse de terminaison ne la satisfait pas nécessairement :

$$\succ_1 \triangleq \{\langle a, a \mid \mathbf{0}\rangle\} \qquad a \succ_1 a \mid \mathbf{0} \succ_2 \tau.a \qquad (4.21)$$
$$\succ_2 \triangleq \{\langle a \mid \mathbf{0}, \tau.a\rangle\}$$

On ne peut donc pas définir la « plus grande relation satisfaisant cette hypothèse de terminaison », ni étendre de façon modulaire une preuve reposant sur les techniques précédentes : dès que l'on ajoute des couples à la relation $\succ$, il faut prouver à nouveau l'hypothèse de terminaison dans son ensemble.

Comme annoncé au chapitre précédent, ce contre-exemple montre par ailleurs que l'union de deux relations contrôlées ne l'est pas nécessairement : $\succ_1$ et $\succ_2$ sont toutes deux contenues dans la bisimilarité faible, ce sont en fait deux relations contrôlées par le théorème 4.31. Cependant, leur union ne peut pas être contrôlée : la relation $(\succ_1 \cup \succ_2)^\star$ contient le couple $\langle a, \tau.a\rangle$, qui ne peut pas être utilisé en position de transitivité dans les jeux de simulation faible (contre-exemple (1.15)).

Remarque 4.38. Ce contre-exemple montre aussi que l'argument de terminaison peut être sensible à des équivalences, même très fortes, telles que la congruence structurelle : si $\succ_2 \cdot \stackrel{\tau}{\Rightarrow}$ termine « syntaxiquement », ce n'est plus le cas dès que l'on considère les processus modulo congruence structurelle : on a en général $a \mid 0 \equiv a$. Plus formellement, le contre-exemple précédent montre la nécessité de la condition de commutation imposée par le théorème 3.51 : sans cette condition, on obtiendrait que la relation $(\succsim \cup \succ_2)$ est contrôlée, ce qui est faux pour les mêmes raisons que précédemment (on a $\succ_1 \subseteq \succsim$).

Naturellement, ce contre-exemple est très artificiel : on a fait exprès de trouver un argument de terminaison tellement syntaxique qu'il s'effondre dès

que l'on raisonne modulo congruence structurelle ; il montre néanmoins qu'il faut faire attention lorsque l'on combine différentes techniques modulo.

4.4 Un autre argument de terminaison

Une alternative possible au lemme de Newman est le lemme de « commutation stricte » de Geser [Ges90, page 47] (ou [TeR03, exercice 1.3.15]) :

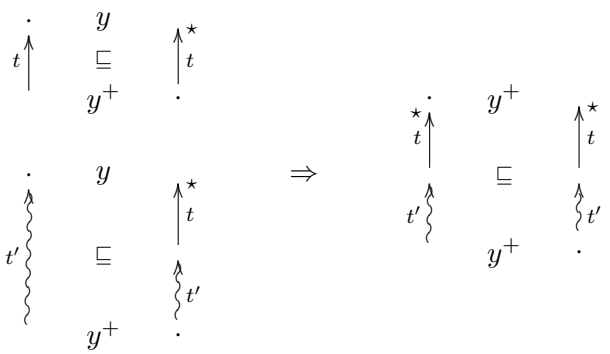
Lemme 4.39 (Commutation stricte, [Ges90])**.** *Soient R et $\to$ deux relations binaires.*

$$\text{Si } \begin{cases} R \text{ est fortement normalisante} \\ \leftarrow \cdot\, R \sqsubseteq R^+ \cdot \leftarrow^\star \end{cases} \qquad \text{alors } R \text{ et } \to \text{ commutent.}$$

Nous montrons dans cette section comment convertir ce résultat de commutation en une nouvelle famille de techniques modulo pour la bisimulation faible. Cette famille de techniques ne bénéficie cependant pas de certaines bonnes propriétés satisfaites par les techniques définies à la section précédente, et nous ne savons pas si elles sont applicables en pratique. Cette section nous sert principalement à illustrer un peu plus la méthode de preuve par induction bien fondée dans les algèbres relationnelles, ainsi que la façon dont les résultats de commutation pure peuvent être étendus en des techniques modulo pour la bisimulation faible. Elle peut éventuellement être ignorée : seul le lemme 4.40 sera utilisé par la suite (pour le lemme 4.48 de la section 4.5.1), de façon légèrement détournée.

4.4.1 Modulo transitivité stricte

Nous commençons par établir dans une algèbre relationnelle quelconque le lemme suivant, qui généralise légèrement le lemme 4.39 (il suffit de prendre $t' = 1$). Il s'énonce comme suit de façon diagrammatique : pour toutes relations y, t, t', si $\overline{y}$ supporte l'induction, alors les deux diagrammes de gauche ci-dessous impliquent le troisième. Comme pour le théorème 4.28, la relation t' est destinée à être instanciée par des relations de la forme $\overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow}$.



Lemme 4.40. *Soient $y, t, t' \in X$ trois relations,*

$$\textit{Si} \begin{cases} (H_1)\ \overline{y}\ \textit{supporte l'induction,} \\ (H_2)\ t \cdot y \sqsubseteq y^+ \cdot t^\star\ , \\ (H_3)\ t' \cdot y \sqsubseteq y^+ \cdot t' \cdot t^\star\ , \end{cases} \qquad \textit{alors } t' \cdot t^\star \cdot y^+ \sqsubseteq y^+ \cdot t' \cdot t^\star\ .$$

Démonstration. Posons $t'' \triangleq t' \cdot t^\star$, et appliquons le principe d'induction spécifique aux diagrammes (théorème 4.27) : soit $A \sqsubseteq 1$ un monotype tel que $A \cdot t'' \cdot y^+ \sqsubseteq y^+ \cdot t''$ (IH), ce qui entraîne en particulier

$$A \cdot t'' \cdot y^\star \sqsubseteq y^\star \cdot t''\ . \tag{IH'}$$

On utilise le support de l'induction de $\overline{y}^+$, que l'on obtient par la proposition 4.25(2). Il nous faut montrer $(A \not/ y^+) \cdot t'' \cdot y^+ \sqsubseteq y^+ \cdot t''$; on procède par récurrence, avec le prédicat $\varphi(n) :\ (A \not/ y^+) \cdot t' \cdot t^n \cdot y^+ \sqsubseteq y^+ \cdot t''$.

– Si $n = 0$, on a

$$\begin{aligned} (A \not/ y^+) \cdot t' \cdot t^0 \cdot y^+ &= (A \not/ y^+) \cdot t' \cdot y \cdot y^\star && \text{(par définition)} \\ &\sqsubseteq (A \not/ y^+) \cdot y^+ \cdot t'' \cdot y^\star && \text{(par } (H_3)\text{)} \\ &\sqsubseteq y^+ \cdot y^\star \cdot t'' = y^+ \cdot t''\ . && \text{(par (IH'))} \end{aligned}$$

– Sinon, supposons $\varphi(n)$ et montrons $\varphi(n+1)$: on a

$$\begin{aligned} &(A \not/ y^+) \cdot t' \cdot t^{n+1} \cdot y^+ \\ &= (A \not/ y^+) \cdot t' \cdot t^n \cdot t \cdot y \cdot y^\star && \text{(définition et lemme 2.12)} \\ &\sqsubseteq (A \not/ y^+) \cdot t' \cdot t^n \cdot y^+ \cdot t^\star \cdot y^\star && \text{(par } (H_2)\text{)} \\ &= (A \not/ y^+) \cdot (A \not/ y^+) \cdot t' \cdot t^n \cdot y^+ \cdot t^\star \cdot y^\star && \text{(par (4.6))} \\ &\sqsubseteq (A \not/ y^+) \cdot y^+ \cdot t'' \cdot t^\star \cdot y^\star && \text{(par récurrence : } \varphi(n)\text{)} \\ &= (A \not/ y^+) \cdot y^+ \cdot t'' \cdot y^\star \\ &\sqsubseteq y^+ \cdot A \cdot t'' \cdot y^\star && \text{(par (4.10))} \\ &\sqsubseteq y^+ \cdot y^\star \cdot t'' = y^+ \cdot t''\ . && \text{(par (IH'))} \end{aligned}$$

■

Alors que la preuve du théorème 4.28 procédait par induction bien fondée à partir du coin supérieur gauche du diagramme, notons que l'on procède ici par induction bien fondée à partir du coin inférieur gauche du diagramme ; la preuve « classique », dans les relations binaires, procède de même.

On déduit du lemme précédent la technique « simulation faible modulo transitivité stricte » :

Théorème 4.41. *Soit $y \in X$ une relation.*

$$\textit{Si} \begin{cases} \overline{y}\ \textit{supporte l'induction} \\ \forall \alpha \in \mathcal{L},\ \overset{\alpha}{\longleftarrow} \cdot y \sqsubseteq y^+ \cdot \overset{\widehat{\alpha}}{\Longleftarrow} \end{cases} \qquad \textit{alors } y^+ \textit{ est une } \mathbf{w}\textit{-simulation.}$$

Démonstration. Par le lemme 4.40, appliqué à y, $t = \xleftarrow{\tau}$ et $t' = 1$, on obtient :

$$\xLeftarrow{\tau} \cdot y^+ \sqsubseteq y^+ \cdot \xLeftarrow{\widehat{\tau}} \ . \tag{4.22}$$

Soit $a \in \mathcal{L}^{\mathrm{v}}$, en combinant l'hypothèse de commutation à (4.22), on a

$$\xLeftarrow{\widehat{\tau}} \cdot \xleftarrow{a} \cdot y \sqsubseteq \xLeftarrow{\widehat{\tau}} \cdot y^+ \cdot \xLeftarrow{a} \sqsubseteq y^+ \cdot \xLeftarrow{a} \ ,$$

ce qui nous permet d'appliquer le lemme 4.40 une seconde fois, à y, $t = \xleftarrow{\tau}$, et $t' = \xLeftarrow{\widehat{\tau}} \cdot \xleftarrow{a}$, et d'obtenir ainsi $\xLeftarrow{a} \cdot y^+ \sqsubseteq y^+ \cdot \xLeftarrow{a}$. La relation y^+ est donc une **w**-simulation. ■

Remarque 4.42. Par opposition au théorème 4.30, la contrainte de terminaison porte ici sur la simulation entière. Ce résultat ne peut donc s'appliquer que pour définir des simulations faibles qui normalisent fortement. De telles relations étant nécessairement irréflexives, le jeu de simulation faible doit donc s'efforcer de maintenir au fil des transitions une distance entre le processus de gauche et celui de droite (le contre-exemple 4.1b montre qu'on ne peut pas remplacer y^+ par $y^\star$ dans l'énoncé du théorème 4.41, même seulement pour les offres visibles : la relation R exhibée est fortement normalisante). Nous ne savons pas si une telle contrainte est réaliste : il semble a priori courant de conclure de nombreux cas dans les jeux de simulation en retombant sur le même processus après un certain nombre de réductions. C'est pour cette raison que l'applicabilité de cette technique modulo en pratique nous semble douteuse.

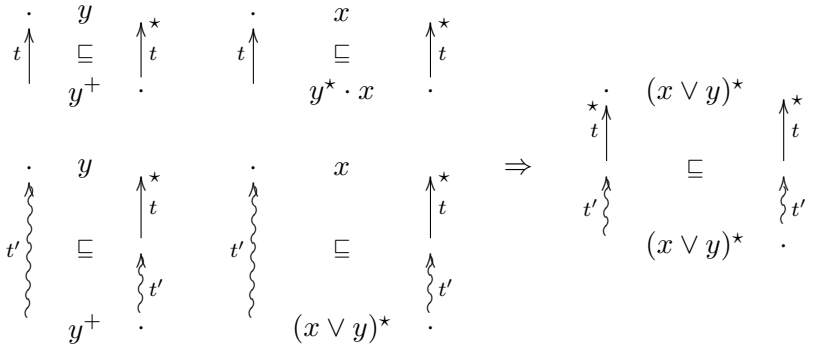
Pour la même raison, cette technique est difficile à symétriser : une relation symétrique ne peut pas supporter l'induction (à moins qu'elle ne soit vide), on ne peut donc pas directement étendre le théorème précédent à la bisimulation faible, comme nous l'avions fait dans le théorème 4.30. Une solution consiste à demander que la bisimulation faible modulo transitivité supporte l'induction dans les deux directions (de telles relations ne sont pas nécessairement triviales) ; une autre possibilité consiste à utiliser d'autres techniques pour la partie de droite à gauche du jeu.

Bien entendu, ce résultat « pur » pourrait être enrichi par des fonctions standard (mise à part modulo réflexivité), en utilisant comme à la section précédente le théorème 2.52.

4.4.2 Une autre famille de préordres contrôlés

S'il est improbable que le théorème précédent puisse servir pour une preuve « finale » de bisimilarité, on peut néanmoins espérer qu'il puisse servir pour établir des résultats préliminaires, à utiliser dans une preuve plus importante. Dans ce cas, il apparaît que l'argument de terminaison précédent donne lieu à une nouvelle famille de préordres contrôlés.

Pour le montrer, il nous faut généraliser un peu plus le lemme 4.40, en introduisant une relation supplémentaire, x, qui correspond intuitivement à une simulation modulo y à gauche. L'énoncé du lemme ainsi obtenu est un peu brutal, comme précédemment, nous en donnons d'abord sa version diagrammatique : pour toutes relations x, y, t, t', si $\overline{y}$ supporte l'induction, alors les quatre diagrammes de gauche ci-dessous impliquent celui de droite. On retrouve le lemme 4.40 en considérant le cas $x = y$; notons cependant que ce dernier lemme est une étape préliminaire nécessaire à la preuve de cette généralisation.



Lemme 4.43. *Soient $x, y, t, t' \in X$ quatre relations,*

$$Si \begin{cases} (H_1)\ \overline{y} \text{ supporte l'induction,} \\ (H_2)\ t \cdot y \sqsubseteq y^+ \cdot t^\star\,, \\ (H_3)\ t' \cdot y \sqsubseteq y^+ \cdot t' \cdot t^\star\,, \\ (H'_2)\ t \cdot x \sqsubseteq y^\star \cdot x \cdot t^\star\,, \\ (H'_3)\ t' \cdot x \sqsubseteq (x \vee y)^\star \cdot t' \cdot t^\star\,, \end{cases} \quad \text{alors } t' \cdot t^\star \cdot (x \vee y)^\star \sqsubseteq (x \vee y)^\star \cdot t' \cdot t^\star\,.$$

Démonstration. On pose $t'' \triangleq t' \cdot t^\star$ et $z \triangleq x \vee y$. Les trois premières hypothèses étant exactement celles du lemme 4.40, celui-ci s'applique : on a

$$t'' \cdot y^+ \sqsubseteq y^+ \cdot t''\ . \tag{4.23}$$

On démontre ensuite $t'' \cdot y^\star \cdot x \sqsubseteq z^\star \cdot t''$, en appliquant le théorème 4.27 : soit $A \sqsubseteq 1$ un monotype tel que :

$$A \cdot t'' \cdot y^\star \cdot x \sqsubseteq z^\star \cdot t''\ . \tag{IH}$$

Il nous faut montrer que $(A/y^+) \cdot t'' \cdot y^\star \cdot x \sqsubseteq z^\star \cdot t''$; on considère pour cela le prédicat $\varphi(n)$: $(A/y^+) \cdot t' \cdot t^n \cdot y^\star \cdot x \sqsubseteq z^\star \cdot t''$, que l'on montre par récurrence sur n. On a dans tous les cas

$$\begin{aligned} (A/y^+) \cdot t' \cdot t^n \cdot y^+ \cdot x &\sqsubseteq (A/y^+) \cdot y^+ \cdot t'' \cdot x && \text{(lemme 4.40)} \\ &\sqsubseteq y^+ \cdot A \cdot t'' \cdot x && \text{(par (4.10))} \\ &\sqsubseteq y^+ \cdot z^\star \cdot t'' && \text{(par (IH))} \\ &\sqsubseteq z^\star \cdot t''\ . \end{aligned}$$

On distingue ensuite selon n pour prouver $(A \not/ y^+) \cdot t' \cdot t^n \cdot x \sqsubseteq z^\star \cdot t''$.
- le cas $n = 0$ vient de (H'_3).
- sinon, supposons $\varphi(n)$ et montrons $\varphi(n+1)$:

$$\begin{aligned}
&(A \not/ y^+) \cdot t' \cdot t^n \cdot t \cdot x \\
&\sqsubseteq (A \not/ y^+) \cdot t' \cdot t^n \cdot y^\star \cdot x \cdot t^\star && \text{(par } (H'_2)\text{)} \\
&\sqsubseteq z^\star \cdot t'' \cdot t^\star = z^\star \cdot t'' \ . && \text{(par récurrence : } \varphi(n)\text{)}
\end{aligned}$$

■

Le résultat annoncé en découle :

Théorème 4.44. *Soit $\succ$ une simulation faible contenue dans la bisimilarité faible. Si $\prec$ supporte l'induction, alors $\succ^\star$ est un préordre contrôlé.*

Démonstration. Par le lemme 3.45, il suffit de montrer que $\succ$ est une relation contrôlée. Soit x une $\mathbf{w}_\mathrm{t}$-simulation modulo pré-composition par $\succ^\star$:

$$\overset{\tau}{\leftarrow} \cdot x \sqsubseteq \succ^\star \cdot x \cdot \overset{\widehat{\tau}}{\Leftarrow} \ , \qquad \forall a \in \mathcal{L}^\mathrm{v},\ \overset{a}{\leftarrow} \cdot x \sqsubseteq x^\star \cdot \overset{a}{\Leftarrow} \ .$$

Posons $z = x \vee \succ$; par une première application du lemme 4.43, à x, $y = \succ$, $t = \overset{\tau}{\leftarrow}$, et $t' = 1$, on obtient tout d'abord la commutation de $z^\star$ et $\overset{\tau}{\rightarrow}$: $(\overset{\widehat{\tau}}{\Leftarrow} \cdot z^\star \sqsubseteq z^\star \cdot \overset{\widehat{\tau}}{\Leftarrow})$, ce qui permet de déduire $\forall a \in \mathcal{L}^\mathrm{v},\ \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow} \cdot x \sqsubseteq z^\star \cdot \overset{a}{\Leftarrow}$.

Par une seconde application du lemme 4.43, à x, $y = \succ$, $t = \overset{\tau}{\leftarrow}$, et $t' = \overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow}$, on obtient alors que $z^\star$ est une $\mathbf{w}$-simulation. On en déduit que $(z^\star \vee \approx)^\star$ est aussi une $\mathbf{w}$-simulation ; comme $\succ \sqsubseteq \approx$, on a $(z^\star \vee \approx)^\star = (x \vee \approx)^\star$, ce qui permet de conclure. ■

Ce théorème est à comparer au théorème 4.31, qui donne une famille de préordres contrôlés fondés sur le premier argument de terminaison : alors que l'on y demandait une relation *quelconque* contenue dans bisimilarité faible et satisfaisant l'argument de terminaison, on doit fournir ici une *simulation* contenue dans bisimilarité faible et satisfaisant l'argument de terminaison. Comme indiqué plus haut, ce couplage que l'on requiert entre les jeux de simulation et l'argument de terminaison nous semble difficile à obtenir en pratique.

Notons aussi que l'on conserve les problèmes de modularité que l'on avait précédemment : les relation $\succ_1$ et $\succ_2$ du contre-exemple (4.21) s'étendent naturellement en des simulations fortement normalisantes et contenues dans $\approx$, mais dont l'union ne saurait être une relation contrôlée :

$$\succ'_1 \triangleq \{\langle a, a \mid \mathbf{0}\rangle,\ \langle \mathbf{0}, \mathbf{0} \mid \mathbf{0}\rangle\} \qquad \succ'_2 \triangleq \{\langle a \mid \mathbf{0}, \tau.a\rangle,\ \langle \mathbf{0} \mid \mathbf{0}, \mathbf{0}\rangle\}$$

De plus, $\succ'_1$ étant contenue dans l'expansion (ou à plus forte raison, dans la congruence structurelle), on a aussi l'incompatibilité de ce genre de techniques avec modulo expansion.

4.5 Le cas des LTS réactifs

Nous nous intéressons dans cette section aux LTS dits *réactifs* [AD07], dont les transitions silencieuses normalisent fortement. L'élaboration ($\gtrapprox$) admet sous cette hypothèse de très bonnes propriétés en termes de techniques modulo, et l'on obtient des résultats importants pour la suite concernant la τ-inertie de tels systèmes.

Définition 4.45 (LTS réactif)**.** Un LTS est *réactif* lorsque $\xleftarrow{\tau}$ supporte l'induction.

La terminologie « réactive » n'est pas standard ; Amadio l'utilise [AD07] car il considère un langage dérivé d'ESTEREL [BG92], où chaque *instant* doit terminer ; cet adjectif nous semble néanmoins intuitif lorsque l'on parle d'un LTS quelconque : un système est réactif s'il ne contient pas de *divergence* (de séquence infinie de transitions silencieuses).

La bisimilarité faible n'est pas sensible aux divergences [Wal90] : dans CCS, on a $\mathbf{0} \approx !\tau$, où le processus de droite est clairement divergent. Cela peut être problématique selon le contexte : si l'on souhaite différencier de tels termes, il faut généralement adopter une équivalence un peu plus fine (la bisimilarité progressive ($\simeq$), ou « finalement progressive » [Nes96]).

Bien que certains algorithmes reposent sur une introduction volontaire de divergences (*data-link protocol*, encodage du choix dans le π-calcul, etc...), la réactivité est généralement désirable : le comportement interne de tels systèmes est plus facilement compréhensible ; la question d'une équivalence plus fine que $\approx$ ne se pose plus, et on montre dans cette section que cette hypothèse permet d'instancier les résultats de la section 4.3 de diverses façons, donnant ainsi lieu à des techniques modulo ne mentionnant pas d'autre hypothèse de terminaison.

4.5.1 Les bonnes propriétés de l'élaboration

Remarquons tout d'abord que dans un LTS réactif, l'hypothèse de terminaison du théorème 4.30 et de ses dérivés devient une hypothèse de terminaison relative : comme $\xrightarrow{\tau}$ termine, la terminaison de $y^+ \cdot \xRightarrow{\tau}$ équivaut à la terminaison de $\xrightarrow{\tau}$ relativement à y. Cela ne renforce pas le théorème puisque cette hypothèse de terminaison relative implique toujours la terminaison de $y^+ \cdot \xRightarrow{\tau}$ (en utilisant la proposition 4.25), mais cela nous semble donner une meilleure intuition. Notons que la preuve de l'implication sur laquelle repose cette intuition est non triviale : il faut passer, par exemple, par le théorème « d'héritage de la terminaison par transitivité » de Geser [Ges90].

Le lemme suivant établit dans une algèbre relationnelle une version légèrement simplifiée du « critère de quasi-commutation » de Bachmair et Dershowitz [BD86] : sous une hypothèse de commutation particulière entre deux

relations x et t, la terminaison de t entraîne sa terminaison relativement à x. L'intuition de la preuve dans le cas concret est donnée plus bas.

Lemme 4.46 (Quasi-commutation [BD86])**.** *Pour toutes relations $x, t \in X$ telles que $t \cdot x \sqsubseteq x^\star \cdot t$, t supporte l'induction si et seulement si $x^\star \cdot t$ la supporte.*

Démonstration. La réciproque vient de la proposition 4.25(1) ; montrons l'implication directe. Par (2.19), on a tout d'abord

$$t \cdot x^\star \sqsubseteq x^\star \cdot t \ . \tag{4.24}$$

Considérons ensuite un monotype $A \sqsubseteq 1$ tel que $(x^\star \cdot t)\backslash A \sqsubseteq A$, nous devons montrer $A = 1$. On a :

$$\begin{aligned}
t\backslash(x^\star\backslash A) &= (x^\star \cdot t)\backslash A && \text{(par (4.11))}\\
&= (x^\star \cdot x^\star \cdot t)\backslash A\\
&\sqsubseteq (x^\star \cdot t \cdot x^\star)\backslash A && \text{((4.14) et (4.24))}\\
&= x^\star\backslash((x^\star \cdot t)\backslash A) && \text{(par (4.11))}\\
&\sqsubseteq x^\star\backslash A \ . && \text{((4.15) et hypothèse sur } A)
\end{aligned}$$

Comme t supporte l'induction, on en déduit $x^\star\backslash A = 1$. Cela suffit à conclure : $x^\star$ étant réflexif, on a $x^\star\backslash A \sqsubseteq A$ (par (4.14) et (4.12)). ■

Notons que l'hypothèse de commutation du critère de Bachmair et Dershowitz est plus faible : $t \cdot x \sqsubseteq (x \vee t)^\star \cdot t$ suffit à obtenir l'équivalence ; nous n'aurons cependant pas besoin de ce raffinement par la suite.

Techniques modulo pour la bisimulation faible

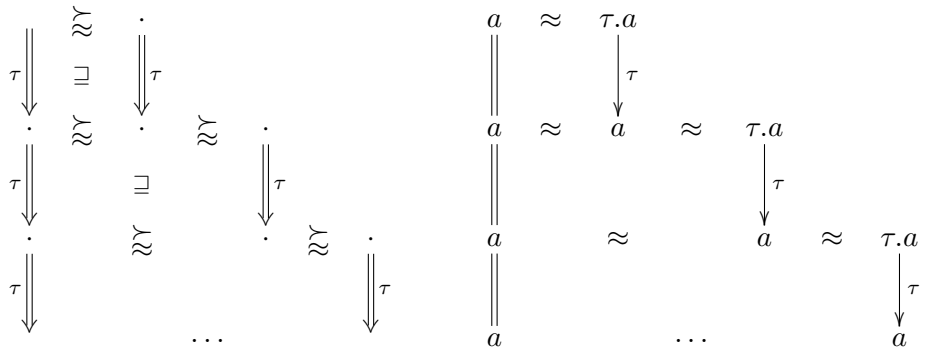
On déduit immédiatement de ce lemme un corollaire du théorème 4.31 : dans tout LTS réactif l'élaboration peut être utilisée en tant que technique modulo pour la bisimulation faible.

Corollaire 4.47. *Dans tout LTS réactif, l'élaboration ($\gtrsim\!\!\!\!\!\approx$) est un préordre contrôlé.*

Démonstration. On applique le théorème 4.31 : l'élaboration est contenue dans la bisimilarité faible (proposition 3.28), et l'hypothèse de terminaison ($\lesssim\!\!\!\!\!\approx^+ \cdot \xleftarrow{\tau}$ supporte l'induction) s'obtient en appliquant le lemme 4.46 à $x = \lesssim\!\!\!\!\!\approx$ et $t = \xleftarrow{\tau}$: par (3.13), on a $\xleftarrow{\tau} \cdot \lesssim\!\!\!\!\!\approx \sqsubseteq \lesssim\!\!\!\!\!\approx \cdot \xleftarrow{\tau}$, et l'élaboration étant réflexive et transitive, on a $\lesssim\!\!\!\!\!\approx^+ = \lesssim\!\!\!\!\!\approx^\star = \lesssim\!\!\!\!\!\approx$. ■

Intuitivement, la preuve concrète du lemme 4.46, dans le cas de l'élaboration, consiste à déduire une séquence infinie de transitions silencieuses en *factorisant* une séquence infinie selon $\gtrsim\!\!\!\!\!\approx \cdot \xRightarrow{\tau}$ (représentée par un « escalier

infini » ci-dessous), afin d'obtenir une contradiction. Le même argument ne s'applique pas à la bisimilarité faible, pour laquelle on a seulement l'inégalité suivante : $\overset{\tau}{\Leftarrow} \cdot \approx \sqsubseteq \approx \cdot \overset{\widehat{\tau}}{\Leftarrow}$. Comme illustré sur le diagramme de droite, lorsque l'on applique la même méthode que précédemment, on s'aperçoit que la séquence que l'on construit peut stationner, nous empêchant ainsi d'aboutir à une contradiction.



Dans le cas des systèmes réactifs, l'élaboration peut donc avantageusement prendre la place de l'expansion dans les jeux de bisimulation faible (corollaires 3.49 et 3.50). On verra de plus au chapitre suivant que l'élaboration est une congruence dans le cas de CCS : la condition de clôture imposée par le corollaire 3.49 est donc automatiquement satisfaite.

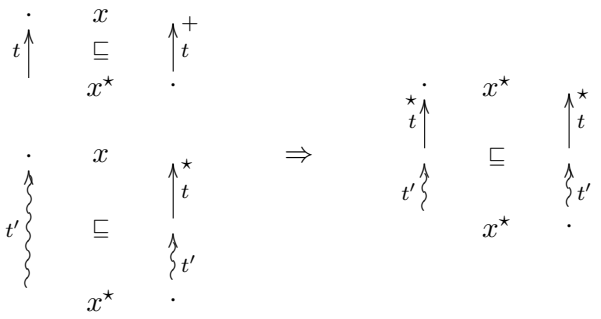
Notons que le corollaire 4.47 n'a pas le problème de la robustesse de l'argument de terminaison vis-à-vis d'équivalences relativement fortes (telles que la congruence structurelle), évoqué à la remarque 4.38 : de telles équivalences sont généralement contenues dans l'élaboration, et sont donc déjà prises en compte.

Rappelons que l'expansion et l'élaboration sont incomparables, mais que l'élaboration est moralement plus proche de la bisimilarité faible que l'expansion (remarque 3.29). Les deux techniques modulo leur correspondant sont par ailleurs incompatibles : on a $a \succsim a|\tau \gtrsim \tau.a$, ce qui permet de rejeter la technique « modulo expansion et élaboration ». Ce n'est pas très étonnant, puisque l'élaboration ne correspond pas, comme l'expansion, à une fonction **w**-compatible (cf. les problèmes de modularité des préordres contrôlés, et le théorème 3.51). Le fait que l'élaboration soit définie de façon co-inductive et soit une congruence lui confère cependant de bonnes propriétés : elle admet notamment elle-même des techniques modulo.

Techniques modulo pour l'élaboration

Nous allons montrer que l'élaboration supporte la technique modulo transitivité, dès que le LTS est réactif. On commence pour cela par montrer un lemme technique, qui étend le lemme de « commutation stricte » de Geser (lemme 4.39). On utilise comme précédemment une relation auxiliaire, t', qui

sera instanciée par $\overset{\widehat{\tau}}{\Leftarrow} \cdot \overset{a}{\leftarrow}$; ce lemme, représenté de façon diagrammatique ci-dessous, peut être vu comme le converse du lemme 4.40 : alors que la terminaison et la transitivité stricte concernaient la relation « horizontale », elles concernent ici la relation « verticale », $\xrightarrow{\tau}$.



Lemme 4.48. *Soient $x, t, t' \in X$ trois relations,*

$$\textit{Si} \begin{cases} (H_1)\ t \textit{ supporte l'induction,} \\ (H_2)\ t \cdot x \sqsubseteq x^\star \cdot t^+ \ , \\ (H_2')\ t' \cdot x \sqsubseteq x^\star \cdot t' \cdot t^\star \ , \end{cases} \qquad \textit{alors } t' \cdot t^\star \cdot x^\star \sqsubseteq x^\star \cdot t' \cdot t^\star \ .$$

Démonstration. On applique tout d'abord le lemme 4.40, à $y = \overline{t}$, $t = \overline{x}$ et $t' = 1$: les hypothèses (H_1) et (H_2) coïncident, et l'hypothèse (H_3) du lemme 4.40 est triviale, puisque $t' = 1$. On obtient donc :

$$t^+ \cdot x^\star \sqsubseteq x^\star \cdot t^+ \ . \tag{4.25}$$

On pose ensuite $t'' \triangleq t' \cdot t^\star$, et on démontre $t' \cdot x^\star \sqsubseteq x^\star \cdot t''$, ce qui suffit à partir de (4.25). On applique pour cela le principe d'induction spécifique aux diagrammes (théorème 4.27) : soit $A \sqsubseteq 1$ un monotype tel que

$$t' \cdot x^\star \cdot A \sqsubseteq x^\star \cdot t'' \ . \tag{IH}$$

Il nous faut montrer $t' \cdot x^\star \cdot (t^+ \backslash A) \sqsubseteq x^\star \cdot t''$; on procède par récurrence sur le prédicat $\varphi(n) :\ t' \cdot x^n \cdot (t^+ \backslash A) \sqsubseteq x^\star \cdot t''$.

- le cas $n = 0$ est trivial ;
- supposons $\varphi(n)$, et montrons $\varphi(n+1)$. L'hypothèse (H_2') donne

$$t' \cdot x^{n+1} \cdot (t^+ \backslash A) \sqsubseteq x^\star \cdot t'' \cdot x^n \cdot (t^+ \backslash A) \ ,$$

et on considère les deux cas suivants ($t'' = t' \vee t' \cdot t^+$) :

$$\begin{aligned} x^\star \cdot t' \cdot x^n \cdot (t^+ \backslash A) &\sqsubseteq x^\star \cdot x^\star \cdot t'' \cdot (t^+ \backslash A) && \text{(par récurrence : } \varphi(n)) \\ &\sqsubseteq x^\star \cdot t'' \end{aligned}$$

$$\begin{aligned} x^\star \cdot t' \cdot t^+ \cdot x^n \cdot (t^+ \backslash A) &\sqsubseteq x^\star \cdot t' \cdot x^\star \cdot t^+ \cdot (t^+ \backslash A) && \text{(par (4.25))} \\ &\sqsubseteq x^\star \cdot t' \cdot x^\star \cdot A \cdot t^+ && \text{(par (4.10))} \\ &\sqsubseteq x^\star \cdot x^\star \cdot t'' \cdot t^+ && \text{(par (IH))} \\ &\sqsubseteq x^\star \cdot t'' \end{aligned}$$

■

On en déduit la correction de la technique modulo transitivité pour la simulation progressive ($\simeq$) :

Proposition 4.49. *Dans tout LTS réactif, la fonction de fermeture réflexive transitive* ($\mathrm{id}^\star_X : x \mapsto x^\star$) *est correcte pour* $\mathbf{p}$ *via elle-même.*

Démonstration. Soit $x \in X$ une $\mathbf{p}$-simulation modulo transitivité : $x \rightsquigarrow_{\mathbf{p}} x^\star$. En appliquant le lemme 4.48 à x, $t = t' = \xleftarrow{\tau}$ (de telle sorte que les deux hypothèses de commutation coïncident), on obtient tout d'abord :

$$\Leftarrow^{\tau} \cdot x^\star \sqsubseteq x^\star \cdot \Leftarrow^{\tau} \ .$$

On en déduit $\Leftarrow^{\widehat{\tau}} \cdot \xleftarrow{a} \cdot x \sqsubseteq x^\star \cdot \Leftarrow^{a}$, pour toute étiquette visible $a \in \mathcal{L}^{\mathrm{v}}$, ce qui permet d'appliquer le lemme 4.48 une seconde fois, à x, $t = \xleftarrow{\tau}$ et $t' = \Leftarrow^{\widehat{\tau}} \cdot \xleftarrow{a}$, pour obtenir $\Leftarrow^{a} \cdot x^\star \sqsubseteq x^\star \cdot \Leftarrow^{a}$. La relation $x^\star$ est donc bien une $\mathbf{p}$-simulation. ■

Corollaire 4.50. *Dans tout LTS réactif, la fonction de fermeture réflexive transitive est correcte pour la génératrice de la bisimilarité progressive,* $\overleftrightarrow{\mathbf{p}}$.

Passer à l'élaboration est plus délicat, puisque la transitivité n'est a priori pas autorisée dans la partie de gauche à droite du jeu, qui est un jeu de simulation faible : la correction de la technique ne peut donc pas s'obtenir par le théorème 2.57, comme l'utilisation simultanée de deux techniques correctes. De manière assez inattendue, la partie de droite à gauche va cependant nous permettre d'obtenir l'hypothèse de terminaison qui autorise la transitivité dans les jeux de simulation faible (théorème 4.30) :

Théorème 4.51. *Dans tout LTS réactif, la fonction de fermeture réflexive transitive est correcte pour la génératrice de l'élaboration,* $\mathbf{w} \wedge \overline{\mathbf{p}}$.

Démonstration. Soit $x \in X$ une élaboration modulo transitivité :

$$x \rightsquigarrow_{\mathbf{w}} x^\star \qquad\qquad \overline{x} \rightsquigarrow_{\mathbf{p}} \overline{x^\star} \ .$$

Par la proposition 4.49, $\overline{x}^\star$ est une **p**-simulation, il nous suffit donc de montrer que $x^\star$ est une **w**-simulation. On applique pour cela le théorème 4.30 en prenant $x = y$; il n'y a plus qu'à montrer que $\overline{x}^+ \cdot \overset{\tau}{\Leftarrow}$ supporte l'induction, ce que l'on obtient par le lemme 4.46 : comme $\overline{x} \rightsquigarrow_{\mathbf{p}} \overline{x}^\star$, on a en particulier $\overset{\tau}{\Leftarrow} \cdot \overline{x} \sqsubseteq \overline{x}^\star \cdot \overset{\tau}{\Leftarrow}$. ■

La même technique pouvant être appliquée des deux côtés, on retrouve un énoncé simple, comme dans le cas de la bisimilarité forte (théorème 3.41), bien que l'on considère une relation asymétrique. Cela nous évite d'avoir à calculer pour accorder des fonctions dans le corollaire suivant : il suffit de vérifier une condition de compatibilité pour pouvoir appliquer le théorème 2.52 :

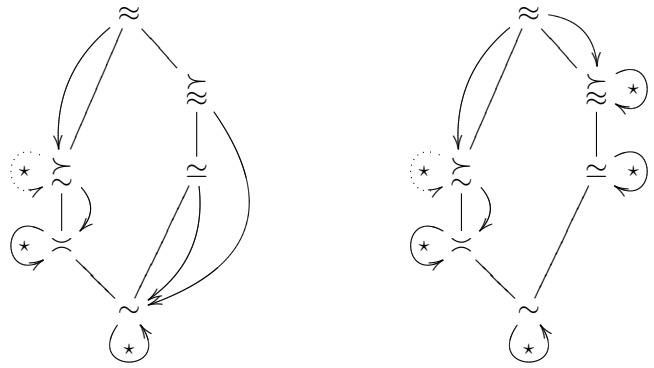
Corollaire 4.52. *Supposons le LTS réactif ; soit $\mathcal{C}$ une clôture compatible avec $\mathrm{id}^\star_X$, **p** et **w**. Toute relation x satisfaisant les deux diagrammes ci-dessous est contenue dans l'élaboration $(\gtrsim)$.*

Démonstration. Application directe du théorème 2.52. ■

L'hypothèse de compatibilité de la clôture avec $\mathrm{id}^\star_X$, que l'on faisait aussi dans le théorème 4.36, mais pas dans les autres résultats impliquant des clôtures, sera vérifiée en pratique (proposition 5.14).

La technique précédente peut être utilisée pour démontrer des résultats de bisimilarité faible, puisque $\gtrsim \sqsubseteq \approx$. On s'aperçoit aussi que l'on est très proche de la technique « faible modulo transitivité » : la différence ne tient qu'à un accent circonflexe manquant dans le diagramme de droite. Cette différence est toutefois importante : le processus de gauche est forcé de répondre au moins une transition silencieuse à chaque offre silencieuse. C'est justement ce genre de contrainte, imposé à chaque étape du jeu, qui nous empêchait d'utiliser l'expansion dans le cas de la GcPan [HPS05] : si l'optimisation est « globalement » plus efficace que la version initiale, elle ne l'est pas à chaque étape du jeu de bisimulation. Il pourrait être intéressant de chercher un notion d'élaboration un peu plus faible, suivant les idées de la « bisimulation finalement progressive » [Nes96], où le processus de gauche ne serait forcé de répondre une transition silencieuse qu'après un certain nombre d'offres silencieuses de la part du processus de droite.

Nous pouvons maintenant compléter le graphique que nous avions donné à la section 3.4.4 : la figure 4.1 récapitule les différentes techniques modulo disponibles pour les équivalences et préordres comportementaux que l'on a considérés (une flèche cyclique contenant une étoile indique que la



(lorsque le LTS est réactif)

FIGURE 4.1 – Spectre des techniques modulo.

transitivité est supportée – dans le cas de l'expansion, elle ne l'est qu'à moitié, d'où les pointillés. Les autres flèches indiquent, lorsque cette technique n'est pas supportée, les relations qui peuvent être utilisées pour y remédier). Les techniques élaboration modulo $\sim$ et bisimulation progressive modulo $\sim$, mentionnées sur le graphe de gauche, s'obtiennent aisément ; elles pourraient même êtres raffinées en considérant des versions « progressives » de l'expansion (i.e., les préordres $\boldsymbol{\nu}(\mathbf{e} \wedge \overline{\mathbf{p}})$ et $\boldsymbol{\nu}(\mathbf{s} \wedge \overline{\mathbf{p}})$, respectivement).

Remarquons finalement que l'hypothèse de réactivité est nécessaire : la relation $\{\langle a \mid !\tau, \tau.a \mid !\tau\rangle, \langle \tau.a \mid !\tau, !\tau\rangle, \langle !\tau, !\tau\rangle\}$ est une élaboration modulo transitivité, sur un LTS non réactif.

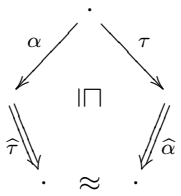
4.5.2 Méthodes associées à la τ-inertie

On revient maintenant sur les résultats annoncés à la section 3.3.2 : dans le cas des systèmes réactifs, on peut donner des techniques puissantes pour démontrer la τ-inertie, puis le faible déterminisme ; conditions qui entraînent la coïncidence de la bisimilarité faible avec la 2-similarité faible (section 3.3.2).

Montrer la τ-inertie

L'idée principale de cette section consiste à remarquer que dans un LTS réactif, la relation $\xrightarrow{\tau}$ elle même satisfait l'hypothèse de terminaison du théorème 4.30 : $\xleftarrow{\tau}^{+} \cdot \Leftarrow^{\tau} = \Leftarrow^{\tau}$ supporte l'induction si et seulement si le LTS est réactif. Le théorème 4.30 admet ainsi pour corollaire un résultat de Groote et Sellink [GS96, théorème 3.5] ; qui s'exprime comme suit selon notre terminologie :

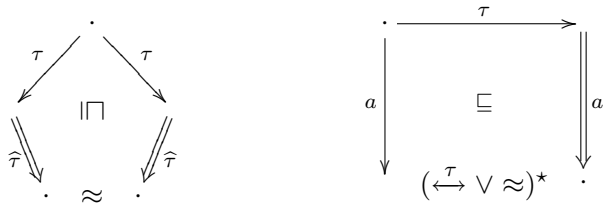
Corollaire 4.53 ([GS96]). *Un LTS réactif est τ-inerte si et seulement si le diagramme suivant est satisfait :*



Démonstration. L'implication directe est immédiate ; pour la réciproque, on applique le théorème 4.30 à $x = \xrightarrow{\tau} \vee \xleftarrow{\tau} \vee \approx$ et $y = \xrightarrow{\tau}$. La relation x est symétrique et majore y, et on a $y^+ \cdot \overset{\tau}{\Rightarrow} = \overset{\tau}{\Rightarrow}$, de telle sorte que l'hypothèse de terminaison corresponde à l'hypothèse de réactivité. On vérifie ensuite que les diagrammes locaux sont satisfaits (notons en particulier que les diagrammes concernant $\xleftarrow{\tau}$ sont toujours trivialement satisfaits : on a $\xleftarrow{\alpha} \cdot \xleftarrow{\tau} \sqsubseteq 1 \cdot \overset{\alpha}{\Leftarrow}$). ■

On peut en fait faire mieux, le théorème 4.30 autorisant complètement la transitivité lors des offres visibles (on note dans la suite $\xleftrightarrow{\tau}$ la symétrisée de la relation $\xrightarrow{\tau}$) :

Corollaire 4.54. *Un LTS réactif est τ-inerte si et seulement si les diagrammes suivants sont satisfaits :*

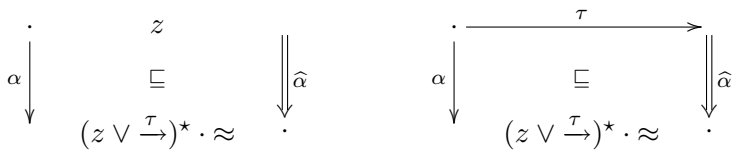


Remarquons que l'on a toujours $(\xrightarrow{\tau} \vee \approx)^\star = \overset{\hat{\tau}}{\Rightarrow} \cdot \approx$, ce qui rend en quelque sorte optimaux les diagrammes silencieux des deux derniers corollaires : on a en fait la transitivité, à condition d'utiliser $\xrightarrow{\tau}$ dans le bon sens (de manière similaire, on a par ailleurs $(\xleftarrow{\tau} \vee \approx)^\star = (\overset{\hat{\tau}}{\Rightarrow} \cdot \approx \cdot \overset{\hat{\tau}}{\Leftarrow})^\star$, ce qui peut permettre de simplifier le diagramme précédent pour le cas visible).

La « complétude » annoncée par les deux corollaires précédents est trompeuse : si tout LTS τ-inerte satisfait effectivement les diagrammes donnés *a posteriori*, on n'est pas assuré de pouvoir démontrer la τ-inertie de tout LTS qui l'est en utilisant ces diagrammes. En effet, comme la bisimilarité faible apparaît dans ces diagrammes, il est possible qu'il faille connaître en partie le résultat de τ-inertie pour pouvoir fermer ces diagrammes... Nous aurons un exemple concret de cette situation problématique au chapitre 6 (théorème 6.44) ; le corollaire ci-dessous nous permettra de s'en sortir. On y utilise une relation auxiliaire (z) qui permet en quelque sorte de renforcer

l'hypothèse de co-induction : on prouve que $z \vee \xrightarrow{\tau}$ est contenue dans la bisimilarité, ce qui permet d'utiliser z pour répondre aux offres concernant $\xrightarrow{\tau}$, et inversement.

Corollaire 4.55. *Supposons le LTS réactif ; soit z une relation symétrique, telle que $z^\star \cdot \xleftarrow{\tau}$ supporte l'induction (i.e., $\xrightarrow{\tau}$ termine modulo z). Si les diagrammes suivants sont satisfaits, alors $\xrightarrow{\tau}, z \sqsubseteq \approx$, et le LTS est τ-inerte.*



Démonstration. On applique le théorème 4.30, aux relations $x = (z \vee \xrightarrow{\tau} \vee \approx)$ et $y = (z \vee \xrightarrow{\tau})$. Pour l'hypothèse de terminaison, on a

$$\overline{y}^+ \cdot \xLeftarrow{\tau} = (z \vee \xleftarrow{\tau})^+ \cdot \xLeftarrow{\tau} = (z^\star \cdot \xLeftarrow{\tau})^+ = (z^\star \cdot \xleftarrow{\tau})^+ \ ,$$

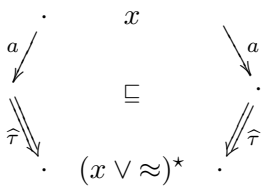
ce qui permet de conclure, par la proposition 4.25. ■

(notons que l'on pourrait remplacer $\xrightarrow{\tau}$ par sa symétrisée $(\xleftrightarrow{\tau})$ lors des offres visibles).

Utiliser la τ-inertie

Les corollaires précédents permettent de montrer qu'un LTS réactif est τ-inerte ; nous montrons maintenant comment ces propriétés peuvent être mises à profit.

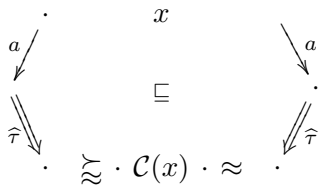
Dans un LTS réactif et τ-inerte, les processus peuvent systématiquement être normalisés selon $\xrightarrow{\tau}$, modulo $\approx$. On peut ainsi toujours se ramener à l'étude de candidats de bisimulation « normalisés », où tous les processus mis en jeu sont stables, : il n'y a ainsi que des offres visibles à considérer, et l'on peut utiliser la transitivité, autorisée lors de ces offres, pour normaliser à nouveau les processus (les transitions visibles sont susceptibles de transformer un processus stable en un processus « actif ») :



Cette méthode ne fonctionne plus lorsque l'on a besoin de raisonner modulo contexte : les techniques modulo contexte ne sont pas compatibles avec la technique modulo transitivité lors des offres visibles (cf. section 5.3.3). Le corollaire suivant, qui complète le corollaire 4.47, permet toutefois de s'en sortir :

Corollaire 4.56. *Dans tout LTS réactif et τ-inerte, $\stackrel{\widehat{\tau}}{\Rightarrow}$ et $\stackrel{\widehat{\tau}}{\Rightarrow} \cdot \gtrapprox$ sont des préordres contrôlés.*

Par le corollaire 3.49 on peut en particulier normaliser les processus après chaque offre visible, tout en conservant la possibilité de raisonner modulo les contextes « d'évaluation » (i.e. tels que $\mathcal{C}(\stackrel{\tau}{\rightarrow}) \sqsubseteq \stackrel{\widehat{\tau}}{\Rightarrow}$ – dans le cas de CCS cette condition est satisfaite par les contextes les plus importants en pratique : composition parallèle, réplication, restriction ; mais pas par le préfixe, dont l'intérêt en tant que technique modulo est limité). Cette technique correspond au diagramme suivant :



Déterminisme

On peut finalement démontrer un résultat annoncé à la section 3.3.2, qui correspond à un résultat de Roberto Amadio et Mehdi Dogguy [AD07, théorème 19(2)], et qui permet, entre autres, de montrer facilement qu'un LTS réactif est faiblement déterministe, et que la bisimilarité faible y coïncide avec la 2-similarité faible (par le corollaire 3.35).

Théorème 4.57. *Supposons le LTS réactif et τ-inerte. Pour toute étiquette visible $a \in \mathcal{L}^{\mathrm{v}}$, si $\stackrel{a}{\rightarrow}$ est déterministe modulo $\approx$, alors $\stackrel{a}{\Rightarrow}$ l'est aussi .*

Démonstration. Par le lemme 3.36, il suffit de montrer $\stackrel{a}{\leftarrow} \cdot \stackrel{a}{\Rightarrow} \sqsubseteq \approx$. On applique pour cela le principe d'induction spécifique aux diagrammes (théorème 4.27) : soit $A \sqsubseteq 1$ un monotype tel que :

$$\stackrel{a}{\leftarrow} \cdot A \cdot \stackrel{a}{\Rightarrow} \sqsubseteq \approx \ . \tag{IH}$$

Il nous faut montrer $\stackrel{a}{\leftarrow} \cdot (\stackrel{\tau}{\Leftarrow} \backslash A) \cdot \stackrel{a}{\Rightarrow} \sqsubseteq \approx$:

- On a $\stackrel{a}{\leftarrow} \cdot (\stackrel{\tau}{\Leftarrow} \backslash A) \cdot \stackrel{a}{\rightarrow} \cdot \stackrel{\widehat{\tau}}{\Rightarrow} \sqsubseteq \stackrel{a}{\leftarrow} \cdot \stackrel{a}{\rightarrow} \cdot \approx \sqsubseteq \approx$, par τ-inertie et déterminisme de $\stackrel{a}{\rightarrow}$ modulo $\approx$;

– il reste donc à montrer $\overset{a}{\leftarrow} \cdot (\overset{\tau}{\Leftarrow} \backslash A) \cdot \overset{\tau}{\Rightarrow} \cdot \overset{a}{\rightarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} \sqsubseteq \approx$:

$$
\begin{aligned}
& \overset{a}{\leftarrow} \cdot (\overset{\tau}{\Leftarrow} \backslash A) \cdot \overset{\tau}{\Rightarrow} \cdot \overset{a}{\rightarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} \\
&= \overset{a}{\leftarrow} \cdot (A \not/ \overset{\tau}{\Rightarrow}) \cdot \overset{\tau}{\Rightarrow} \cdot \overset{a}{\rightarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} && \text{(par (4.8))} \\
&\sqsubseteq \overset{a}{\leftarrow} \cdot \overset{\tau}{\Rightarrow} \cdot A \cdot \overset{a}{\rightarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} && \text{(par (4.10))} \\
&\sqsubseteq \approx \cdot \overset{a}{\Leftarrow} \cdot A \cdot \overset{a}{\rightarrow} \cdot \overset{\widehat{\tau}}{\Rightarrow} && (\tau\text{-inertie}) \\
&= \approx \cdot \overline{\overset{a}{\leftarrow} \cdot A \cdot \overset{a}{\Rightarrow}} \cdot \overset{\widehat{\tau}}{\Rightarrow} && \text{(par (4.6))} \\
&\sqsubseteq \approx \cdot \approx \cdot \overset{\widehat{\tau}}{\Rightarrow} && \text{(par (IH))} \\
&\sqsubseteq \approx && \text{(par } \tau\text{-inertie et transitivité)}
\end{aligned}
$$

■

4.6 Récapitulatif

Nous avons tout d'abord obtenu un nouveau résultat de commutation (théorème 4.14), que nous avons pu prouver dans le cas d'un algèbre relationnelle quelconque en utilisant la méthode de Doornbos et al. [DBvdW97] pour raisonner par induction de manière calculatoire. Une fois appliqué à la bisimulation faible, ce résultat permet d'autoriser modulo transitivité, à condition de respecter une condition de terminaison (théorème 4.30), il donne lieu à une famille de préordres contrôlés (théorème 4.31) et il peut être combiné à des techniques standard telles que modulo contexte (corollaire 4.37).

Nous avons ensuite exploré un autre argument de terminaison, qui donne aussi lieu à de nouvelles techniques modulo transitivité et à de nouveaux préordres contrôlés (théorèmes 4.41 et 4.44), mais cette famille de techniques semble difficile à mettre en application : l'argument de terminaison est trop fortement couplé au jeu de bisimulation.

Enfin, nous avons étudié quelques conséquences des résultats s'appuyant sur le premier argument de terminaison, dans le cas des systèmes réactifs. Dans ces systèmes, l'élaboration satisfait automatiquement l'argument de terminaison, ce qui en fait un préordre contrôlé et lui permet de supporter complètement la technique modulo transitivité (corollaires 4.47 et 4.52). L'ensemble des techniques modulo pour les différents préordres et équivalences comportementaux est résumé sur la figure 4.1. D'autre part, on obtient des techniques relativement puissantes pour démontrer ou utiliser la τ-inertie d'un système réactif : le théorème 4.30 admet pour corollaire une généralisation d'un résultat de Groote et Sellink [GS96] : le corollaire 4.55, que utiliserons au chapitre 6 lorsque nous aurons à prouver la correction d'une optimisation.

Chapitre 5

Techniques modulo contexte

Nous avons étudié dans les deux chapitres précédents des techniques modulo pour la bisimulation qui permettent de réduire par transitivité la taille des candidats de bisimulation : toutes nos preuves consistent en des assemblages de diagrammes plus ou moins évolués. Nous passons dans ce chapitre à l'étude des techniques « modulo contexte », qui permettent, elles, de réduire les candidats de bisimulation en nous donnant la possibilité de se focaliser sur la partie intéressante des processus qu'ils mettent en relation.

Dans le cas fréquent où la bisimilarité étiquetée est utilisée pour caractériser une congruence, et fournir ainsi des techniques de preuve pour cette congruence, tout l'intérêt de la méthode réside dans la capacité à remplacer la quantification universelle sur l'ensemble des contextes, par une quantification, plus raisonnable, sur l'ensemble des étiquettes : au lieu d'observer les processus par le biais des contextes, on les compare à l'aide de leurs transitions étiquetées. Ainsi, dans un jeu de bisimulation, lorsque l'on retombe sur deux processus déjà mis en relation « modulo contexte » :

$$\begin{array}{ccc} p & R & q \\ {\scriptstyle a}\downarrow & & \downarrow{\scriptstyle a} \\ c[p'] & & c[q'] \end{array}$$

$$p' \; R \; q'$$

avoir à rajouter le couple $\langle c[p'], c[q'] \rangle$ au candidat de bisimulation est contre-productif, puisque cela revient à réintroduire une partie de la quantification sur les contextes. Cela est d'autant plus frustrant que l'on a déjà vérifié, moralement, que p' et q' sont équivalents dans tout contexte : étant mis en relation par le candidat, leurs transitions étiquetées sont forcément étudiées. On souhaiterait donc pouvoir répondre à de telles offres dans la fermeture par contextes du candidat de bisimulation : si $\mathcal{C}$ est cette fonction de fermeture, dans l'exemple précédent, on a en particulier $c[p'] \; \mathcal{C}(R) \; c[q']$, ce qui nous permet de ne pas ajouter de couples.

Ces techniques, initialement introduites par Sangiorgi [San98], se sont révélées essentielles en pratique, par exemple pour les preuves des encodages du λ-calcul dans le π-calcul ; leur extension à des calculs « d'ordre supérieur », où elles sont d'autant plus utiles, est réputée délicate (λ-calcul [Las98], HOπ [SW01]).

L'une des difficultés provient du fait que l'on doive généralement considérer les contextes polyadiques : dès que le calcul considéré est capable de dupliquer les termes (c'est le cas du λ-calcul et du π-calcul), un contexte initialement monadique peut évoluer en un contexte polyadique. Les preuves deviennent rapidement pénibles : il faut faire des inductions structurelles sur ces contextes polyadiques. Sangiorgi utilise dans le cas fort la correction de modulo transitivité pour pouvoir se ramener au cas des contextes monadiques [SW01], mais il doit toujours raisonner par induction structurelle sur les contextes, même monadiques, et cette approche ne passe pas au cas faible puisque modulo transitivité n'y est plus correcte.

Nous définissons dans ce chapitre une nouvelle méthode, que nous appelons « méthode des contextes initiaux », qui permet de simplifier ces preuves, en les ramenant à une étude simple et systématique des interactions entre chaque construction syntaxique du langage considéré et le jeu de simulation choisi. On évite notamment, dans le cas fort comme dans le cas faible, les inductions structurelles sur les contextes. Cette méthode repose sur des « techniques modulo de second ordre », qui permettent de faciliter les preuves de compatibilité (les techniques modulo contexte s'exprimant généralement par le biais de fonctions compatibles).

La première section est un complément du chapitre 2, où l'on définit ces techniques modulo de second ordre, en restant dans le cadre très générique des treillis complets, éventuellement monoïdaux. Nous définissons ensuite notre méthode de façon générique (section 5.2), avant de l'appliquer à la section 5.3 au *Calcul des Systèmes Communiquants (CCS)* [Mil89] : on retrouve ainsi de façon uniforme l'ensemble des techniques modulo contexte disponibles pour chacun des préordres et équivalences comportementaux précédemment étudiés.

5.1 Techniques modulo pour la compatibilité

L'ensemble $X^{\langle X\rangle}$ des fonctions croissantes sur un treillis complet X, muni de la composition fonctionnelle et de l'identité, forme un treillis complet monoïdal (théorème 2.27). La théorie de la co-induction du chapitre 2 permet donc de caractériser certaines propriétés des fonctions, et d'obtenir des techniques modulo permettant de faciliter la preuve de telles propriétés.

Nous montrons dans cette section que la notion de compatibilité est une telle propriété : elle est générée par une progression, et admet donc une

théorie des techniques modulo. En d'autres termes on peut utiliser des techniques modulo dans l'espace des fonctions, afin de définir des techniques modulo dans le treillis complet initial.

On travaille dans cette section dans un treillis complet $\langle X, \sqsubseteq \rangle$, et on fixe une génératrice $s \in X^{\langle X \rangle}$; on appellera dans la suite *fonctionnelles* les fonctions croissantes sur $X^{\langle X \rangle}$.

Définition 5.1 (Progression de compatibilité)**.** On appelle *progression de compatibilité avec* s la relation suivante, entre fonctions croissantes :

$$\forall f, g \in X^{\langle X \rangle},\ f \overset{s}{\rightarrowtail} g \quad \text{si} \quad f \circ s \sqsubseteq s \circ g\ .$$

Proposition 5.2. *La relation* $\overset{s}{\rightarrowtail}$ *est une progression sur treillis complet monoïdal* $\langle X^{\langle X \rangle}, \sqsubseteq, \bigvee, \circ, \mathrm{id}_X \rangle$*, qui respecte le monoïde.*

Démonstration. (2.38) provient de la croissance des fonctions : si $f \sqsubseteq f' \overset{s}{\rightarrowtail} g' \sqsubseteq g$, alors $f \circ s \sqsubseteq f' \circ s \sqsubseteq s \circ g' \sqsubseteq s \circ g$. (2.39) s'obtient grâce à la continuité à gauche de la composition fonctionnelle (2.24) : soient $F \sqsubseteq X^{\langle X \rangle}$ et $g \in X^{\langle X \rangle}$ telles que $\forall f \in F,\ f \overset{s}{\rightarrowtail} g$, on a :

$$\bigvee F \circ s = \bigvee_{f \in F} (f \circ s) \sqsubseteq s \circ g\ .$$

Pour le respect du monoïde, on a clairement $\mathrm{id}_X \overset{s}{\rightarrowtail} \mathrm{id}_X$, ensuite, si $f \overset{s}{\rightarrowtail} f'$ et $g \overset{s}{\rightarrowtail} g'$, alors on a : $f \circ g \circ s \sqsubseteq f \circ s \circ g' \sqsubseteq s \circ f' \circ g'$, i.e., $f \circ g \overset{s}{\rightarrowtail} f' \circ g'$. ■

Par définition, les fonctions compatibles avec s sont les $\overset{s}{\rightarrowtail}$-simulations, et la plus grande d'entre elles est la $\overset{s}{\rightarrowtail}$-similarité. Remarquons que la définition de $\overset{s}{\rightarrowtail}$ est très proche de celle de la progression générant les simulations fortes ($\rightsquigarrow_{\mathbf{s}}$, définition 3.10) : on change simplement de treillis complet monoïdal, et on considère un LTS à une seule étiquette. Bien que le treillis complet monoïdal des fonctions croissantes ne soit pas continu, comme l'est celui des algèbres relationnelles, la continuité à gauche de la composition fonctionnelle (lemme 2.28) suffit à ce que cette définition fournisse effectivement une progression.

Notons aussi que c'est la mise en évidence des deux présentations équivalentes du chapitre 2 (génératrices v.s. progressions), qui permet de se rendre compte de cette caractérisation des fonctions compatibles : la notion de compatibilité appartient à la présentation basée sur les génératrices, mais se caractérise à l'aide d'une progression. Trouver directement la fonctionnelle dont les post-points fixes sont les fonctions compatibles ne semble pas évident, et inversement, il n'est pas immédiat de remarquer que la caractérisation que l'on donne ci-dessous donne lieu à une progression. Cette caractérisation, plus confortable en pratique, sera utilisée dans les deux sections suivantes.

Proposition 5.3. *Lorsque s est la génératrice associée à une progression $\rightsquigarrow$, pour toutes fonctions $f, g \in X^{\langle X\rangle}$, on a $f \overset{s}{\rightarrowtail} g$ si et seulement si :*

$$\forall x, y \in X,\ x \rightsquigarrow y \Rightarrow f(x) \rightsquigarrow g(y)\ .$$

Démonstration. Cette preuve est une adaptation de celle de la proposition 2.84(3) :

- si $f \overset{s}{\rightarrowtail} g$ et $x \rightsquigarrow y$, alors $x \sqsubseteq s(y)$, puis $f(x) \sqsubseteq f(s(y)) \sqsubseteq s(g(y))$, c'est-à-dire $f(x) \rightsquigarrow g(y)$;
- pour la réciproque, pour tout $x \in X$, on a $s(x) \sqsubseteq s(x)$, d'où $s(x) \rightsquigarrow x$, puis $f(s(x)) \rightsquigarrow g(x)$, i.e., $f(s(x)) \sqsubseteq s(g(x))$. On a donc $f \overset{s}{\rightarrowtail} g$. ■

Comme la progression $\overset{s}{\rightarrowtail}$ respecte le monoïde $\langle X^{\langle X\rangle}, \circ, \mathrm{id}_X\rangle$, les propositions 2.37 et 2.67 donnent une preuve alternative des bonnes propriétés de fermeture de la famille des fonctions compatibles avec s, initialement établies par la proposition 2.50. En passant dans l'espace des fonctions croissantes, cette même proposition 2.50, ainsi que la proposition 2.68, nous donnent les propriétés de fermeture de la famille des fonctionnelles monotones pour $\overset{s}{\rightarrowtail}$:

Corollaire 5.4. *Les fonctionnelles suivantes sont monotones pour $\overset{s}{\rightarrowtail}$:*

1. *la fonctionnelle identité :* $\mathrm{id}_{X^{\langle X\rangle}}$,
2. *la fonctionnelle constante $\widehat{f}$, pour toute fonction f compatible avec s ;*
3. *$\bigvee \Phi$, pour tout ensemble Φ de fonctionnelles monotones pour $\overset{s}{\rightarrowtail}$;*
4. *$\varphi \circ \varphi'$, pour toutes fonctionnelles φ et φ' monotones pour $\overset{s}{\rightarrowtail}$;*
5. *φ^ω, pour toute fonctionnelle φ monotone pour $\overset{s}{\rightarrowtail}$;*
6. *$\varphi \mathbin{\hat{\circ}} \varphi'$ pour toutes fonctionnelles φ et φ' monotones pour $\overset{s}{\rightarrowtail}$;*
7. *la* fonctionnelle de fermeture par itération*, $\omega : f \mapsto f^\omega$.*

Démonstration. Les cinq premiers points proviennent de la proposition 2.50 et les deux derniers de la proposition 2.68. ■

Nous revenons sur les notations de ce corollaire, qui sont un peu ambiguës : dans (4), $\varphi \circ \varphi'$ dénote la composition fonctionnelle de φ et φ, qui à une fonction f associe $\varphi(\varphi'(f))$. Au contraire, dans (6), $(\hat{\circ})$ est l'extension point à point de la composition fonctionnelle sur $X^{\langle X\rangle}$: $\varphi \mathbin{\hat{\circ}} \varphi' : f \mapsto \varphi(f) \circ \varphi'(f)$. Dans (5), φ^ω est la fermeture par itération de la fonctionnelle φ, tandis que la fonctionnelle ω, définie pour (7), est la fonctionnelle de fermeture réflexive transitive vis-à-vis de la composition des fonctions.

Le lemme 2.78, qui établit la correction des fonctions monotones, nous assure alors que pour toute fonctionnelle φ monotone pour $\rightsquigarrow_s$, et toute fonction f *compatible avec s modulo φ* (i.e., telle que $f \overset{s}{\rightarrowtail} \varphi(f)$), la fonction

$\varphi^\omega(f)$ est compatible avec s. On obtient ainsi la technique suivante, sur laquelle repose la méthode que l'on développe dans la section suivante pour les technique modulo contexte. La preuve se résume à simplifier $\varphi^\omega(f)$ afin de savoir en pratique quelle est la fonction dont on a prouvé la compatibilité (c'est seulement à cette fin que sont utilisées l'extensivité et la continuité de f) :

Théorème 5.5. *Pour toute fonction f extensive, continue, et compatible avec s modulo itération ($f \overset{s}{\rightarrowtail} f^\omega$), f^ω est une fermeture compatible avec s.*

Démonstration. La fonction f est une fermeture par la proposition 2.35(1). La fonctionnelle ω étant monotone avec $\overset{s}{\rightarrowtail}$ par le corollaire 5.4, il suffit de simplifier $\omega^\omega(f)$. La fonction f^ω étant idempotente (c'est une fermeture), on a $\omega^{(2)}(f) \triangleq (f^\omega)^\omega = f^\omega \triangleq \omega(f)$ par la proposition 2.35(1) ; on en déduit alors $\omega^\omega(f) = f^\omega$. ■

On retrouve aussi de façon immédiate le résultat suivant, qui correspond à un résultat que Sangiorgi utilise dans le cas de la bisimilarité forte [SW01, lemme 2.3.16]. Nous l'utiliserons pour comparer notre approche à celle de Sangiorgi dans la section suivante.

Proposition 5.6. *Supposons que X est un treillis complet monoïdal continu. Si s respecte le monoïde, alors pour toute fonction croissante f telle que $f \overset{s}{\rightarrowtail} f^\star$, $f^\star$ est compatible avec s.*

Démonstration. Par la proposition 2.68(2) la fonction $\mathrm{id}_X^\star$ est compatible avec s. Par le corollaire 5.4, on en déduit la monotonie pour $\overset{s}{\rightarrowtail}$ de la fonctionnelle $\varphi = \widehat{\mathrm{id}_X^\star} \mathbin{\hat{\circ}} \mathrm{id}_{X\langle X\rangle} : f \mapsto f^\star$. Le lemme 2.78 nous donne alors la compatibilité de $\varphi^\omega(f)$ avec s.

On simplifie ensuite cette fonction : par continuité du monoïde, on a $f \sqsubseteq \varphi(f) = f^\star = (f^\star)^\star = \varphi^{(2)}(f)$, d'où $\varphi^\omega(f) = \varphi(f) = f^\star$. ■

Nous verrons à la section suivante que le théorème 5.5 ne suffit pas toujours : il peut être utile de prouver la compatibilité d'une fonction en raisonnant modulo itération et modulo une fonction dont on connaît déjà la compatibilité. Les points (7) et (2) du corollaire 5.4 indiquent que c'est possible, et pourraient être utilisés pour valider la technique ; nous donnons cependant une preuve plus directe, qui nous permettra de montrer que l'on peut aussi raisonner modulo une fonction seulement correcte. Cette preuve passe par le lemme suivant :

Lemme 5.7. *Soient f une fonction extensive et continue, et g une fermeture. Si $f \overset{s}{\rightarrowtail} g \circ f^\omega$ et $f \overset{g}{\rightarrowtail} f^\omega$, alors f^ω est une fermeture compatible avec g et $(s \circ g)$.*

Démonstration. Le fait que f^ω soit une fermeture compatible avec g provient du théorème 5.5. Ensuite, comme $f \circ s \sqsubseteq s \circ g \circ f^\omega$ par hypothèse, on a :

$$\begin{aligned} f \circ s \circ g &\sqsubseteq s \circ g \circ f^\omega \circ g \\ &\sqsubseteq s \circ g \circ g \circ f^\omega && \text{(compatibilité de } f^\omega \text{ avec } g) \\ &= s \circ g \circ f^\omega && \text{(idempotence de } g) \end{aligned}$$

On en déduit la compatibilité de f^ω avec $(s \circ g)$ en appliquant à nouveau le théorème 5.5. ■

On obtient alors le théorème annoncé : si f est compatible modulo g et itération, où g est compatible, alors $g \circ f^\omega$ est compatible (les autres hypothèses sur f sont dans ce cas utilisées seulement pour simplifier les calculs – en pratique, f sera simplement compatible avec g, pas seulement modulo itération). Si g est seulement correcte, alors on obtient seulement la correction de $g \circ f^\omega$ (et l'hypothèse de compatibilité de f avec g est cette fois cruciale : elle correspond à la condition donnée par le théorème 2.52, pour pouvoir composer une fonction correcte et une fonction compatible).

Théorème 5.8. *Soient f une fonction extensive et continue, et g une fermeture telles que $f \overset{s}{\rightarrowtail} g \circ f^\omega$ et $f \overset{g}{\rightarrowtail} f^\omega$.*

1. *Si g est compatible avec s, alors $g \circ f^\omega$ est une fermeture compatible avec s.*
2. *Si g est correcte pour s via une fonction g', alors $g \circ f^\omega$ est une fermeture correcte pour s via $g' \circ f^\omega$.*

Démonstration. Dans les deux cas, on commence par appliquer le lemme 5.7 : f^ω est une fermeture compatible avec g et $(s \circ g)$, puis on déduit de la compatibilité avec g que $g \circ f^\omega$ est une fermeture : on a $g \circ f^\omega \circ g \circ f^\omega \sqsubseteq g \circ g \circ f^\omega \circ f^\omega = g \circ f^\omega$. Pour le premier point, on vérifie ensuite que

$$\begin{aligned} g \circ f^\omega \circ s &\sqsubseteq g \circ f^\omega \circ s \circ g && \text{(extensivité de } g) \\ &\sqsubseteq g \circ s \circ g \circ f^\omega && \text{(compatibilité de } f^\omega \text{ avec } s \circ g) \\ &\sqsubseteq s \circ g \circ g \circ f^\omega && \text{(compatibilité de } g \text{ avec } s) \\ &= s \circ g \circ f^\omega \ . && \text{(idempotence de } g) \end{aligned}$$

Pour le second, comme f^ω est compatible avec $(s \circ g)$, la proposition 2.49 nous donne $f^\omega(X_{s \circ g \circ f}) \subseteq X_{s \circ g}$, d'où l'on déduit $g' \circ f^\omega(X_{s \circ g \circ f}) \subseteq X_s$ par correction de g pour s via g'. ■

Nous utiliserons le premier point de ce théorème à la section 5.3, pour valider les techniques modulo contexte dans CCS, dans le cas faible ; nous n'avons pour l'instant pas d'exemple d'application du second point, qui nous semble cependant très important : il permet en quelque sorte de prouver la

correction d'une fonction par des méthodes de type compatibilité, mais à travers la correction d'une technique arbitraire.

Remarquons d'ailleurs que le second point est en fait une généralisation du théorème 2.52 : au lieu de composer une fonction correcte avec une fonction compatible sous une hypothèse de compatibilité, on y compose une fonction correcte avec une fonction compatible modulo itération (et modulo cette fonction correcte) sous une hypothèse de compatibilité modulo itération. La même analogie peut être faite entre les lemmes 2.51 et 5.7.

5.2 Méthode des contextes initiaux

Etudier les techniques modulo contexte nécessite de se placer dans une algèbre relationnelle propre : la notion même de contexte nécessite de pouvoir parler des objets reliés par les relations. Dans cette section, où l'on définit une méthode générique, il n'est cependant pas nécessaire de connaître la structure de ces objets. On se place donc dans l'algèbre relationnelle propre d'un ensemble quelconque $\mathcal{P}$; dont le domaine (l'ensemble des relations binaires) est noté $\mathcal{R}$.

Définition 5.9 (Contexte, clôture par contextes)**.** Pour tout entier $n \in \mathbb{N}$, un *contexte d'arité* n est une fonction c de $\mathcal{P}^n$ dans $\mathcal{P}$, dont l'application à n processus $p_1, \ldots, p_n$ est notée $c[p_1, \ldots, p_n]$. Les contextes d'arité 1 sont dit *monadiques*, ceux d'arité quelconque seront parfois dits *polyadiques*.

On associe à tout contexte c d'arité n la fonction suivante :

$$\begin{aligned} \lfloor c \rfloor : \mathcal{R} &\to \mathcal{R} \\ R &\mapsto \{\langle c[p_1, \ldots, p_n], c[q_1, \ldots, q_n]\rangle \mid \forall i \in [1; n],\ p_i\ R\ q_i\} \end{aligned}$$

On étend cette définition aux ensembles C de contextes, en posant :

$$\lfloor C \rfloor \triangleq \bigcup_{c \in C} \lfloor c \rfloor \ .$$

La fonction $\lfloor C \rfloor^\omega$ sera appelée *clôture par les contextes de* C.

Notre définition de contexte est très permissive : un contexte, même monadique, peut utiliser plusieurs fois chacun de ses arguments ; il peut de plus modifier ses arguments : lorsque le langage des termes contient des noms ou des variables, toute substitution, qu'elle soit des noms vers les noms, ou des variables vers les termes, forme un contexte monadique. On pourra donc considérer comme des techniques modulo contexte les techniques modulo « substitution injective » souvent utilisées dans le cas du π-calcul [SW01]. Un autre avantage à représenter les contextes par des fonctions, outre le fait que cela ne nécessite pas de fixer une syntaxe, est la facilité avec laquelle on peut les dénoter : comme nous le verrons à la section suivante dans le cas de

CCS, il n'y a pas besoin d'introduire la notion de « trou », de numéroter ces trous lorsque l'on a affaire à des contextes polyadiques, etc...

Un autre aspect important de notre approche est que nous passons le plus rapidement possible des contextes aux fonctions associées à ces contextes, évitant ainsi d'avoir à manipuler les arités (nous ne les manipulons que dans le lemme ci-dessous), et bénéficiant de l'expressivité que fournissent les différentes opérations sur les fonctions de relations. Pour se rendre compte des bénéfices de cette approche, il suffit par exemple d'essayer de définir un ensemble D de contextes donc la fonction associée, $\lfloor D \rfloor$, coïncide avec $\lfloor C \rfloor \circ \lfloor C \rfloor$, pour un ensemble C de contextes quelconques : dans le cas monadique, il suffit de prendre $D = \{c \circ c' \mid c, c' \in C\}$, mais le cas polyadique est loin d'être aussi simple.

Nous validons maintenant la définition de ces fonctions de clôture par contextes, en montrant que ce sont bien des clôtures (définition 3.38). Cela nécessite le lemme suivant, où l'on prouve, entre autres, la continuité des fonctions associées aux contextes. La continuité n'est pas requise par notre définition de clôture, mais elle nous semble être la bonne manière d'obtenir l'idempotence dans le théorème qui suit.

Lemme 5.10. *Pour tout contexte c, la fonction $\lfloor c \rfloor$ est continue, symétrique, et satisfait $\lfloor c \rfloor(I) \subseteq I$ et $\forall R, S,\ \lfloor c \rfloor(R \cdot S) \subseteq \lfloor c \rfloor(R) \cdot \lfloor c \rfloor(S)$*

Démonstration. La symétrie et les deux inégalités sont immédiates ; nous montrons la continuité. Notons $f = \lfloor c \rfloor$ et soit n l'arité de c.

Soit $\mathcal{S}$ une partie dirigée de $\mathcal{R}$, f étant clairement croissante, il suffit par (2.31) de montrer $f(\bigcup \mathcal{S}) \subseteq \bigcup f(\mathcal{S})$. Soient $p, q \in \mathcal{P}$ tels que $p\ f(\bigcup \mathcal{S})\ q$: on a $p = c[p_1 \dots p_n]$ et $q = c[q_1 \dots q_n]$, avec $p_i \bigcup \mathcal{S}\ q_i$ pour tout $i \in [1; n]$. Pour tout $i \in [1; n]$, il existe donc $S_i \in \mathcal{S}$ tel que $p_i\ S_i\ q_i$. $\mathcal{S}$ étant dirigée, on en déduit $S \in \mathcal{S}$ tel que l'on ait $p_i\ S\ q_i$ pour tout $i \in [1; n]$. On a donc $p\ f(S)\ q$, puis $p \bigcup f(\mathcal{S})\ q$. ■

(Notons que le fait que la continuité ne concerne que les parties dirigées est crucial, dès que le contexte n'est pas monadique).

Pour obtenir une fonction idempotente après une seule itération, il faut que la famille de contextes contienne au moins l'identité ; si l'on ne considérait que des contextes monadiques, cette condition ne serait pas nécessaire :

Théorème 5.11. *Pour tout ensemble C de contextes contenant au moins le contexte identité ($\mathrm{id}_{\mathcal{P}}$), la fonction de clôture par les contextes de C ($\lfloor C \rfloor^{\omega}$) est une clôture (continue et telle que $\lfloor C \rfloor^{\omega}(I) \subseteq I$).*

Démonstration. Les quatre propriétés mentionnées dans le lemme 5.10 étant préservées par les unions et par la composition fonctionnelle, on déduit de ce lemme que $\lfloor C \rfloor$, puis $\lfloor C \rfloor^{\omega}$ les satisfont. $\lfloor C \rfloor^{\omega}$ étant extensive par définition,

il ne nous reste à montrer que l'idempotence. On applique pour cela la proposition 2.35(1) : $\lfloor C \rfloor$ est continue par le point précédent, et son extensivité provient du fait que C contient le contexte identité : on a $\lfloor \mathrm{id}_{\mathcal{P}} \rfloor = \mathrm{id}_{\mathcal{R}}$. ■

L'idempotence est le point le plus important du théorème précédent : cette propriété indique que l'on capture bien avec $\lfloor C \rfloor^\omega$ l'ensemble des contextes accessibles à partir de C : toute fonction qui s'exprime comme une combinaison des fonctions $\{\lfloor c \rfloor \mid c \in C\}$, à l'aide de la composition fonctionnelle, des bornes supérieures, et des itérations, est contenue dans $\lfloor C \rfloor^\omega$. Supposons par exemple que C contient deux contextes c, d, on a :

$$\begin{aligned} & \lfloor c \rfloor \circ \lfloor d \rfloor^{(3)} \cup \lfloor d \rfloor \circ (\lfloor d \rfloor \cup \lfloor c \rfloor^\omega) \\ \subseteq\ & \lfloor C \rfloor^\omega \circ (\lfloor C \rfloor^\omega)^{(3)} \cup \lfloor C \rfloor^\omega \circ ((\lfloor C \rfloor^\omega) \cup (\lfloor C \rfloor^\omega)^\omega) \\ =\ & \lfloor C \rfloor^\omega \cup \lfloor C \rfloor^\omega \circ (\lfloor C \rfloor^\omega \cup \lfloor C \rfloor^\omega) \\ =\ & \lfloor C \rfloor^\omega \ . \end{aligned}$$

On peut donc se contenter de ne mettre dans C que des contextes très simples que l'on appellera *initiaux*, correspondant, lorsque les processus sont des termes, aux diverses constructions syntaxiques du langage.

Méthode des contextes initiaux

L'intérêt de pouvoir réduire l'ensemble C au strict minimum apparaît avec le corollaire suivant, qui permet de ramener la preuve de compatibilité de la clôture finale ($\lfloor C \rfloor^\omega$), i.e., la preuve de correction de la technique modulo contexte, à l'étude des interactions entre la génératrice et les différents contextes initiaux :

Corollaire 5.12 (Méthode des contextes initiaux)**.** *Soient s une génératrice et C un ensemble de contextes contenant l'identité. Si pour tout contexte $c \in C$, on a $\lfloor c \rfloor \overset{s}{\rightarrowtail} \lfloor C \rfloor^\omega$, alors $\lfloor C \rfloor^\omega$ est une clôture compatible avec s, et la s-similarité est une congruence vis-à-vis de $\lfloor C \rfloor^\omega$: $\lfloor C \rfloor^\omega(\boldsymbol{\nu} s) = \boldsymbol{\nu} s$.*

Démonstration. Application directe de des théorèmes 5.11 et 5.8 ; la propriété de congruence provient de la proposition 2.53. ■

De manière plus explicite, en utilisant la caractérisation de $\overset{s}{\rightarrowtail}$ en termes de progression (proposition 5.3), il suffit de montrer que pour tout contexte $c \in C$, et toutes relations R, S telles que $R \rightsquigarrow_s S$, on a $\lfloor c \rfloor(R) \rightsquigarrow_s \lfloor C \rfloor^\omega(S)$.

Telle qu'elle est présentée ici, cette méthode nous permet essentiellement d'éviter l'induction structurelle qui est nécessaire lorsque l'on prouve directement $\lfloor C \rfloor \overset{s}{\rightarrowtail} \lfloor C \rfloor$, pour un ensemble « complet » C de contextes. Le théorème 5.8 nous donne en fait la possibilité d'aller plus loin, en nous permettant de raisonner modulo une fonction dont on connaît déjà la compatibilité ; nous aurons besoin de cette possibilité à la section suivante, pour pouvoir raisonner modulo $\sim$ lorsque nous considérerons le cas faible :

Corollaire 5.13. *Soient s une génératrice, g une clôture compatible avec s, et C un ensemble de contextes contenant l'identité. Si pour tout contexte $c \in C$, $\lfloor c \rfloor$ est compatible avec g et $\lfloor c \rfloor \overset{s}{\rightarrowtail} g \circ \lfloor C \rfloor^\omega$, alors $g \circ \lfloor C \rfloor^\omega$ est une clôture compatible avec s.*

Démonstration. On applique tout d'abord le théorème 5.11 : $\lfloor C \rfloor^\omega$ est une clôture continue. On applique ensuite le théorème 5.8(1) ; on vérifie notamment que la compatibilité des fonctions $(\lfloor c \rfloor)_{c \in C}$ avec g entraînent la compatibilité de $\lfloor C \rfloor$ avec g, et en particulier modulo itération. $g \circ \lfloor C \rfloor^\omega$ est donc une fermeture compatible avec s ; et il ne reste qu'à vérifier que c'est en fait une clôture, ce qui est aisé puisque g en est une. ■

Avant de passer au cas concret de CCS, nous donnons une dernière proposition : au chapitre 4, le théorème 4.36 et le corollaire 4.52 demandent que la clôture soit aussi compatible avec $\mathrm{id}_X^\star$. La proposition suivante montre qu'il suffit que l'ensemble de contextes initiaux contienne aussi les contextes constants (ici encore, ce ne serait pas nécessaire avec des contextes monadiques, et le traitement des contextes polyadiques rend cette proposition plus délicate à prouver qu'elle n'en a l'air).

Proposition 5.14. *Pour tout ensemble C de contextes contenant l'identité et les contextes constants $\{\widehat{p} \mid p \in \mathcal{P}\}$, la clôture $\lfloor C \rfloor^\omega$ est compatible avec $\mathrm{id}_X^\star$.*

Démonstration. Posons $f = \lfloor C \rfloor^\omega$; f satisfait les propriétés du lemme 5.10, en déroulant les définitions, il nous faut montrer $\forall R,\ f(R^\star) \subseteq f(R)^\star$.

On démontre tout d'abord cette inégalité dans le cas d'une relation réflexive : par une simple récurrence, on a $\forall n \in \mathbb{N},\ f(R^n) \subseteq f(R)^n$; ensuite comme R est réflexive, la suite $(R^n)_{n \in \mathbb{N}}$ est croissante, et forme en particulier une partie dirigée. On peut donc conclure par la continuité de f.

On démontre ensuite que pour toute relation R, $f(R^=) \subseteq f(R)$: comme les contextes constants et identité sont supposés dans C, on a $\mathrm{id}_X^= \subseteq \lfloor C \rfloor$, puis $\mathrm{id}_X^= \subseteq f$, et $f \circ \mathrm{id}_X^= \subseteq f$, par idempotence.

Cette inégalité nous permet finalement de se ramener au cas réflexif : pour toute relation R, on a $f(R^\star) = f((R^=)^\star) \subseteq f(R^=)^\star \subseteq f(R)^\star$. ■

5.3 Le cas de CCS

A terme, nous pensons pouvoir appliquer la méthode précédente dans le cas du π-calcul [MPW92], voire du π-calcul d'ordre supérieur [SW01]. Par manque de temps, et par souci de simplicité, nous nous contentons ici d'étudier le cas du *Calcul des Systèmes Communiquants (CCS)* [Mil80, Mil89], dont la syntaxe et la sémantique sont définies sur la figure 5.1. On y suppose l'ensemble des étiquettes ($\mathcal{L}$) muni d'une involution $\bar{\cdot}$ telle que $\overline{\tau} = \tau$.

$$\begin{array}{rll} \text{étiquettes :} & \alpha, \beta \in \mathcal{L} & \overline{\tau} = \tau \\ \text{préfixes :} & a, b \in \mathcal{L}^{\mathrm{v}} & \overline{\overline{a}} = a \\ \text{processus } (\mathcal{P}) : & p, q ::= \mathbf{0} \mid \alpha.p \mid p \mid q \mid p + q \mid (\nu a)p \mid\ !p & \end{array}$$

$$[\textsc{Prf}]\ \frac{}{\alpha.p \xrightarrow{\alpha} p} \qquad [\textsc{Par}_g]\ \frac{p \xrightarrow{\alpha} p'}{p \mid q \xrightarrow{\alpha} p' \mid q} \qquad [\textsc{Par}_d]\ \frac{q \xrightarrow{\alpha} q'}{p \mid q \xrightarrow{\alpha} p \mid q'}$$

$$[\textsc{Com}]\ \frac{p \xrightarrow{a} p' \qquad q \xrightarrow{\overline{a}} q'}{p \mid q \xrightarrow{\tau} p' \mid q'} \qquad [\textsc{Chx}_g]\ \frac{p \xrightarrow{\alpha} p'}{p + q \xrightarrow{\alpha} p'} \qquad [\textsc{Chx}_d]\ \frac{q \xrightarrow{\alpha} q'}{p + q \xrightarrow{\alpha} q'}$$

$$[\textsc{Res}]\ \frac{p \xrightarrow{\alpha} p'}{(\alpha \neq a, \overline{a})}\ (\nu a)p \xrightarrow{\alpha} (\nu a)p' \qquad [\textsc{Rpl}]\ \frac{!p \mid p \xrightarrow{\alpha} p'}{!p \xrightarrow{\alpha} p'}$$

FIGURE 5.1 – Calcul des systèmes communiquants (CCS).

La version originale du calcul contient des définitions récursives, que nous remplaçons ici par l'opérateur de réplication (!), afin de se rapprocher du π-calcul, où c'est généralement cette construction qui est adoptée ; cela nous permet aussi de considérer cette opération comme un contexte, et de soulever ainsi certains problèmes. Les différences avec le calcul simpliste présenté en introduction sont les suivantes :

- l'opérateur de mise en parallèle permet aux processus de se synchroniser (de communiquer), par la règle [Com] ;
- il y a un opérateur de *restriction* (ν), qui empêche certaines communications avec l'extérieur (règle [Res]) ;
- les transitions de l'opérateur de *réplication* (!) sont définies différemment, afin de pouvoir prendre en compte les communications pouvant intervenir entre les différentes copies du processus répliqué.

Comme dans l'introduction, nous omettrons souvent les occurrences préfixées du processus vide ($\mathbf{0}$) : le processus $a.\overline{b}.\mathbf{0}$ sera ainsi écrit $a.\overline{b}$.

Nous commençons par définir informellement les contextes polyadiques standard : tout processus $p \in \mathcal{P}$, auquel on attribue un entier inférieur ou égal à n à chaque occurrence de $\mathbf{0}$, donne lieu un à contexte d'arité n : par exemple, le processus numéroté $a.\mathbf{0}_0 \mid b.(\mathbf{0}_1 \mid \mathbf{0}_1) \mid (\nu c)\mathbf{0}_3$ définit le contexte ternaire suivant : $p_1, p_2, p_3 \mapsto a.\mathbf{0} \mid b.(p_1 \mid p_1) \mid (\nu c)p_3$. On note C_{ccs} l'ensemble de ces contextes. Nous définissons ensuite les contextes initiaux, qui suivent exactement la syntaxe de CCS :

Définition 5.15 (Contextes initiaux de CCS)**.** Pour tous $a \in \mathcal{L}^{\mathrm{v}}$ et $\alpha \in \mathcal{L}$,

on définit les *contextes initiaux de CCS* comme suit :

$$\widehat{\mathbf{0}} : \emptyset \mapsto \mathbf{0} \qquad + : p, q \mapsto p + q \qquad (\nu a) : p \mapsto (\nu a)p$$
$$\alpha. : p \mapsto \alpha.p \qquad | : p, q \mapsto p \mid q \qquad ! : p \mapsto !p$$

On note C_i l'ensemble contenant ces contextes, le contexte identité et les contextes constants. On appelle *clôture par les contextes de CCS* la clôture associée à cet ensemble : $\mathcal{C}_{ccs} \triangleq \lfloor C_i \rfloor^\omega$.

$$C_i \triangleq \{\mathrm{id}_\mathcal{P}, |, +, !\} \cup \{\alpha. \mid \alpha \in \mathcal{L}\} \cup \{(\nu a) \mid a \in \mathcal{L}^\mathrm{v}\} \cup \{\widehat{p} \mid p \in \mathcal{P}\} \quad .$$

Remarquons que l'on inclut dans C_i tous les contextes constants, afin de pouvoir appliquer éventuellement la proposition 5.14 ; dans ce cas particulier, il suffirait cependant de mettre seulement le contexte constant $\widehat{\mathbf{0}}$, qui permet d'atteindre tous les autres par itération, puisque toutes les constructions syntaxiques sont présentes.

La proposition suivante permet de relier les deux notions ainsi obtenues. La définition standard ($\lfloor C_{ccs} \rfloor$) est plus directe en pratique : on a $p \ \lfloor C_{ccs} \rfloor(R) \ q$ si et seulement si p et q s'obtiennent en remplaçant par des processus mis en relation par R certaines occurrences de $\mathbf{0}$ d'un processus arbitraire. Mais notre définition ($\lfloor C_i \rfloor^\omega$) permet de raisonner plus facilement sur cette fonction, en se ramenant au cas des contextes initiaux.

Proposition 5.16. *On a* $\mathcal{C}_{ccs} \triangleq \lfloor C_i \rfloor^\omega = \lfloor C_{ccs} \rfloor$.

Démonstration (esquisse). On montre $\forall c \in C_{ccs},\ \lfloor c \rfloor \subseteq \mathcal{C}_{ccs}$ par induction structurelle sur le processus numéroté dont est issu c. Réciproquement, on montre $\forall n \in \mathbb{N},\ \lfloor C_i \rfloor^{(n)} \subseteq \lfloor C_{ccs} \rfloor$ par récurrence sur n. ■

Avant de passer à l'étude des techniques modulo contexte correspondant à ces contextes pour chacune des notions de bisimulation précédemment définies, nous donnons un lemme technique qui nous permettra de raisonner plus facilement sur la forme des transitions émergeant d'un processus répliqué :

Lemme 5.17. *Soient* $p, q \in \mathcal{P}$ *deux processus et* $\alpha \in \mathcal{L}$ *une étiquette. Si* $!p \xrightarrow{\alpha} q$*, alors :*

1. *soit* $q = !p \mid p^k \mid p' \mid p^{k'}$ *avec* $p \xrightarrow{\alpha} p'$ *(où* p^k *dénote la mise en parallèle de* k *copies de* p*) ;*
2. *soit* $\alpha = \tau$*, et* $q = !p \mid p^k \mid p_0 \mid p^{k'} \mid p_1 \mid p^{k''}$ *avec* $p \xrightarrow{a} p_0$*,* $p \xrightarrow{\overline{a}} p_1$ *pour une action visible* a*.*

Démonstration. Par induction structurelle sur la dérivation de $!p \xrightarrow{\alpha} q$. ■

5.3.1 Bisimilarité forte

Nous validons maintenant la technique modulo contexte pour la bisimilarité forte. Par les résultats du chapitre 3, il suffit de montrer que la clôture $\mathcal{C}_{ccs}$ est compatible avec la génératrice des simulations fortes (**s**). On applique pour cela la méthode établie par le corollaire 5.12 : il suffit de montrer que les fonctions associées aux contextes initiaux progressent via $\overset{\mathbf{s}}{\rightarrowtail}$ vers la fonction de clôture globale ($\mathcal{C}_{ccs}$).

On peut en profiter pour être plus précis, et indiquer dans chaque cas la fonction vers laquelle elles progressent « réellement ». Ainsi, le lemme suivant formalise par exemple l'idée que les opérateurs de mise en parallèle et de restriction, que Milner qualifie de « statiques » [Mil89], sont préservés le long des jeux de simulation forte : les fonctions qui leur sont associées progressent vers elles-mêmes. Au contraire, le préfixe et la somme, qui sont dits « dynamiques », disparaissent après une étape du jeu de simulation forte : leurs fonctions associées progressent vers l'identité. La fonction vers laquelle progresse la fonction associée à la réplication est moins évidente : elle indique que cet opérateur est conservé lors des jeux de simulation forte, et qu'il génère à chaque étape un nombre arbitraire de compositions parallèles. Dans tous les cas, comme on travaille ici avec la simulation forte, qui préserve strictement les transitions, on retrouve exactement au niveau des relations binaires le comportement initial de ces opérateurs, sur les processus.

Lemme 5.18. *On a :*

$$\lfloor \mathrm{id}_{\mathcal{P}} \rfloor = \mathrm{id}_{\mathcal{R}} \overset{\mathbf{s}}{\rightarrowtail} \mathrm{id}_{\mathcal{R}} \qquad \lfloor \widehat{p} \rfloor \overset{\mathbf{s}}{\rightarrowtail} \widehat{I}$$

$$\lfloor \alpha. \rfloor \overset{\mathbf{s}}{\rightarrowtail} \mathrm{id}_{\mathcal{R}} \qquad \lfloor (\nu a) \rfloor \overset{\mathbf{s}}{\rightarrowtail} \lfloor (\nu a) \rfloor$$

$$\lfloor | \rfloor \overset{\mathbf{s}}{\rightarrowtail} \lfloor | \rfloor \qquad \lfloor + \rfloor \overset{\mathbf{s}}{\rightarrowtail} \mathrm{id}_{\mathcal{R}}$$

$$\lfloor ! \rfloor \overset{\mathbf{s}}{\rightarrowtail} \lfloor | \rfloor^{\omega} \circ (\lfloor ! \rfloor \cup \mathrm{id}_{\mathcal{R}})$$

Démonstration. Le cas de l'identité ($\mathrm{id}_{\mathcal{R}}$) est immédiat ; dans chacun des autres cas, on suppose deux relations binaires R, S et trois processus $u, u', v \in \mathcal{P}$ tels que $R \rightsquigarrow_{\mathbf{s}} S$, $u \lfloor c \rfloor(R)\ v$ et $u \xrightarrow{\alpha} u'$, et l'on doit trouver un processus v' tel que $v \xrightarrow{\alpha} v'$ et $u' \lfloor c' \rfloor(S)\ v'$.

- $\lfloor \widehat{p} \rfloor$: $u = v = p$, $u' = p'$ avec $p \xrightarrow{\alpha} p'$. On prend $v' = p'$, on a bien $p'\ I\ p'$.
- $\lfloor \alpha. \rfloor$: $u = \alpha'.p \xrightarrow{\alpha} u'$, $v = \alpha'.q$ avec $p\ R\ q$. La seule règle de transition pour un processus préfixé est [Prf], d'où $\alpha = \alpha'$ et $u' = p$, et l'on a donc $v = \alpha.q \xrightarrow{\alpha} q$, avec $p\ \mathrm{id}_{\mathcal{R}}(S)\ q$, (rappelons que $R \rightsquigarrow_{\mathbf{s}} S$ implique $R \subseteq S$ par définition).
- $\lfloor (\nu a) \rfloor$: $u = (\nu a)p \xrightarrow{\alpha} u'$, $v = (\nu a)q$ avec $p\ R\ q$. La seule règle de transition pour un processus restreint est [Res], d'où $u' = (\nu a)p'$ où $p \xrightarrow{\alpha} p'$ et $\alpha \neq a, \overline{a}$. Comme $p\ R\ q$, on obtient q' tel que $q \xrightarrow{\alpha} q'$ et $p'\ S\ q'$, et l'on vérifie que $v \xrightarrow{\alpha} v' = (\nu a)q'$, avec $u' \lfloor (\nu a) \rfloor(S)\ v'$.

$\lfloor|\rfloor$: $u = p_1|p_2 \xrightarrow{\alpha} u'$, $v = q_1|q_2$ avec $p_1\ R\ q_1$ et $p_2\ R\ q_2$. Suivant les règles d'inférence dans le cas d'une composition parallèle, il y a trois cas :
- $[\text{PAR}_g]$: $u' = p'_1|p_2$ avec $p_1 \xrightarrow{\alpha} p'_1$. Comme $R \rightsquigarrow_{\mathbf{s}} S$, on a $q_1 \xrightarrow{\alpha} q'_1$ avec $p'_1\ S\ q'_1$. On vérifie alors que $v \xrightarrow{\alpha} v' = q'_1|q_2$ et $u'\ \lfloor|\rfloor(S)\ v'$ (on utilise là aussi le fait que $R \rightsquigarrow_{\mathbf{s}} S \Rightarrow R \subseteq S$, de telle sorte que l'on a aussi $p_2\ S\ q_2$).
- $[\text{PAR}_d]$: $u' = p_1|p'_2$ avec $p_2 \xrightarrow{\alpha} p'_2$, qui est identique au cas précédent.
- $[\text{COM}]$: $u' = p'_1|p'_2$ avec $p_1 \xrightarrow{a} p'_1$, $p_2 \xrightarrow{\overline{a}} p'_2$, et $\alpha = \tau$. On a $q_1 \xrightarrow{a} q'_1$ $q_2 \xrightarrow{\overline{a}} q'_2$ avec $p'_1\ S\ q'_1$ et $p'_2\ S\ q'_2$; de telle sorte que $v \xrightarrow{\tau} v' = q'_1|q'_2$ et $u'\ \lfloor|\rfloor(S)\ v'$.

$\lfloor+\rfloor$: $u = p_1 + p_2 \xrightarrow{\alpha} u'$, $v = q_1 + q_2$ avec $p_1\ R\ q_1$ et $p_2\ R\ q_2$. Suivant les règles d'inférence dans d'un somme, il y a deux cas :
- $[\text{CHX}_g]$: $u' = p'_1$ avec $p_1 \xrightarrow{\alpha} p'_1$. Comme $R \rightsquigarrow_{\mathbf{s}} S$, on a $q_1 \xrightarrow{\alpha} q'_1$ avec $p'_1\ S\ q'_1$. On vérifie alors que $v \xrightarrow{\alpha} v' = q'_1$ et $u'\ S\ v'$.
- $[\text{CHX}_d]$: $u' = p'_2$ avec $p_2 \xrightarrow{\alpha} p'_2$, qui est identique au cas précédent.

$\lfloor!\rfloor$: $u = !p \xrightarrow{\alpha} u'$, $v = !q$ avec $p\ R\ q$. Il y a deux cas possibles, par le lemme 5.17 :

1. $u' = !p|p^k|p'|p^{k'}$ avec $p \xrightarrow{\alpha} p'$. On en déduit $q \xrightarrow{\alpha} q'$, avec $p'\ S\ q'$. On vérifie ensuite que l'on a bien $v \xrightarrow{\alpha} v' = !q|q^k|q'|q^{k'}$, et $u'\ \lfloor|\rfloor^{k+k'+1} \circ (\lfloor!\rfloor \cup \mathrm{id}_{\mathcal{R}})(S)\ v'$.
2. $u' = !p|p^k|p_0|p^{k'}|p_1|p^{k''}$ avec $p \xrightarrow{a} p_0$, $p \xrightarrow{\overline{a}} p_1$ et $\alpha = \tau$. On en déduit $q \xrightarrow{a} q_0$ et $q \xrightarrow{\overline{a}} q_1$ avec $p_0\ S\ q_0$ et $p_1\ S\ q_1$. On vérifie alors que $v \xrightarrow{\tau} v' = !q|q^k|q_0|q^{k'}|q_1|q^{k''}$, où $u'\ \lfloor|\rfloor^{k+k'+k''+1} \circ (\lfloor!\rfloor \cup \mathrm{id}_{\mathcal{R}})(S)\ v'$. ■

On déduit de ce lemme le résultat annoncé :

Théorème 5.19. *La clôture $\mathcal{C}_{ccs}$ est compatible avec* **s**.

Démonstration. Dans le lemme précédent, toutes les fonctions vers lesquelles progressent les éléments de C_i sont contenues dans $\mathcal{C}_{ccs}$. On peut donc appliquer la méthode des contextes initiaux (corollaire 5.12). ■

La bisimilarité forte supporte donc la technique modulo contextes, pour tous les contextes de CCS ; on obtient de plus le corollaire suivant (où l'on appelle simplement *congruence* toute relation étant une congruence vis-à-vis de $\mathcal{C}_{ccs}$) :

Corollaire 5.20. *La bisimilarité forte est une congruence sur CCS.*

Si la preuve du lemme 5.18 est relativement longue, c'est parce que nous l'avons donnée dans les moindres détails : cette preuve pourrait être formalisée dans l'assistant de preuve COQ [CH84] sans aucun problème : nous ne raisonnons pas modulo congruence structurelle, et la façon dont on déduit la

forme syntaxique des processus après chaque transition correspond très précisément à la façon dont on détruit une hypothèse inductive dans un outil tel que Coq.

L'avantage de donner dans le lemme 5.18 les fonctions précises vers lesquelles progressent les différentes fonctions est que l'on peut ensuite obtenir la compatibilité des fonctions de clôture par certains contextes seulement. Par exemple, on a immédiatement que les fonctions $(\lfloor|\rfloor \cup \mathrm{id}_{\mathcal{R}})^\omega$ et $(\lfloor|\rfloor \cup \lfloor!\rfloor \cup \mathrm{id}_{\mathcal{R}})^\omega$ sont deux clôtures compatibles avec $\mathbf{s}$. Cela peut être utile lorsque l'on souhaite utiliser des techniques telles que le théorème 4.56, où une condition de congruence doit être vérifiée entre la clôture utilisée et la relation qui porte l'hypothèse de terminaison. Cela illustre aussi la modularité de la méthode : si l'on devait rajouter au calcul une nouvelle construction, il suffirait pour étendre la technique modulo contexte et la preuve de congruence de vérifier que la fonction associée à cette construction progresse par $\overset{\mathbf{s}}{\rightarrowtail}$ vers une fonction obtenue à partir d'elle-même et des fonctions associées aux autres contextes.

Comparaison avec l'approche de Sangiorgi

Nous donnons maintenant les grandes lignes de l'approche suivie par Sangiorgi [SW01], afin de la comparer à la nôtre. Notons C_m la restriction de l'ensemble des contextes de CCS (C_{ccs}) aux contextes monadiques et n'utilisant qu'une seule fois leur argument (i.e. ceux correspondant aux processus dont une seule occurrence de $\mathbf{0}$ est numérotée par 1, et toutes les autres par 0). L'intérêt de cette sous-classe de contextes est qu'elle permet d'atteindre la clôture $\mathcal{C}_{ccs}$, par transitivité : on a en effet les inclusions suivantes entre fonctions

$$\lfloor C_m \rfloor \subsetneq \lfloor C_{ccs} \rfloor \subsetneq \lfloor C_m \rfloor^\star$$

(la première inclusion provient de l'inclusion ensembliste $C_m \subseteq C_{ccs}$: tout contexte monadique est un contexte polyadique. Pour la seconde, on utilise la transitivité pour instancier une par une les occurrences numérotées du contexte polyadique). Sangiorgi utilise ensuite la proposition 5.6, qui permet de se ramener à prouver $\lfloor C_m \rfloor \overset{\mathbf{s}}{\rightarrowtail} \lfloor C_m \rfloor^\star$, et d'éviter ainsi de manipuler des contextes polyadiques. Il y a cependant deux inconvénients : bien que ce soit plus simple qu'avec des contextes polyadiques, on doit toujours faire une induction structurelle sur les contextes monadiques considérés ; mais surtout, cette méthode ne passe pas au cas faible, où la proposition 5.6 n'est plus valide (cette proposition repose sur la correction de modulo transitivité). Notre approche résout ces deux problèmes : si les contextes initiaux ne sont pas tous monadiques, ils restent bien plus simples que ces derniers et ne nécessitent pas d'induction structurelle pour être analysés : ils correspondent très précisément aux constructions syntaxiques du langage ; de plus comme on utilise l'itération plutôt que la fermeture transitive, cette méthode s'étend

sans problème au cas faible.

Notons par ailleurs que les contextes polyadiques ne peuvent pas être évités lorsque l'on considère la réplication : le contexte réplicatif (!), qui est à la fois initial et monadique, évolue par réduction en un contexte polyadique (non initial). C'est pour ce contexte particulier que l'on utilise l'itération dans notre preuve, ou la fermeture transitive dans celle de Sangiorgi, afin de se ramener aux contextes que l'on est entrain d'observer.

5.3.2 Bisimilarité faible

Nous passons maintenant au cas de la bisimilarité faible. Cette équivalence n'étant pas une congruence vis-à-vis de la somme (on a par exemple $a \approx \tau.a$, mais pas $a + b \approx \tau.a + b$, car le dernier processus peut faire une action silencieuse vers a, que ne sait pas mimer le processus $a + b$), tous les contextes ne peuvent pas être admis en tant que technique modulo. On commence pour simplifier par ignorer la somme, que l'on rajoutera seulement dans un second temps, à la section 5.3.4.

Définition 5.21 (Contextes initiaux de CCS, sans somme)**.** On note C_i^- l'ensemble C_i privé du contexte somme, et $\mathcal{C}_{ccs}^-$ la clôture associée :

$$C_i^- \triangleq \{\mathrm{id}_{\mathcal{P}}, |, !\} \cup \{\alpha. \mid \alpha \in \mathcal{L}\} \cup \{(\nu a) \mid a \in \mathcal{L}^{\mathrm{v}}\} \cup \{\widehat{p} \mid p \in \mathcal{P}\}$$
$$\mathcal{C}_{ccs}^- \triangleq \lfloor C_i^- \rfloor^{\omega} \ .$$

Cette définition nous donne naturellement l'ensemble des contextes de CCS qui n'utilisent jamais un argument dans un sous-terme d'une somme.

Contrairement à ce qui est annoncé dans [SW01, lemme. 2.4.52], la clôture $\mathcal{C}_{ccs}^-$ n'est pas compatible avec $\mathbf{w}$: prenons par exemple $R = \{\langle \tau.a, a\rangle\} \cup I$; bien que $R \rightsquigarrow_{\mathbf{w}} R$, on n'a pas $\mathcal{C}_{ccs}^-(R) \rightsquigarrow_{\mathbf{w}} \mathcal{C}_{ccs}^-(R)$: on ne peut pas répondre à l'offre ci-dessous dans $\mathcal{C}_{ccs}^-(R)$ puisque $!a$ ne peut pas bouger ; il nous faudrait pouvoir réécrire ce processus en $!a|a$:

$$\begin{array}{ccc} !\tau.a & \lfloor ! \rfloor(R) & !a \\ \downarrow \tau & & \\ !\tau.a|a & & ? \end{array}$$

C'est possible modulo déroulage des réplications : comme le montre le lemme suivant, cette opération est contenue dans la bisimilarité forte ($\sim$). Il faudrait donc corriger [SW01] en définissant la fonction de clôture par contextes modulo déroulage des réplications. Toutefois, la preuve, qui est déjà pénible du fait que l'on doive manipuler des contextes polyadiques, deviendrait encore plus pénible, et sujette à d'autres erreurs : il faut être capable de n'oublier aucune des situations résultant d'un déroulage de réplications et de l'application d'un contexte polyadique quelconque.

Le lemme suivant nous permettra de « dérouler » des réplications.

Lemme 5.22. *Pour tout processus p, on a $!p \mid p \sim !p$.*

Démonstration. Pour tout processus p, le singleton $\{\langle !p \mid p, !p\rangle\}$ est une bisimulation forte modulo réflexivité. ■

On peut alors montrer les progressions de compatibilité suivantes :

Lemme 5.23. *On a :*

$$\lfloor \mathrm{id}_{\mathcal{P}} \rfloor = \mathrm{id}_{\mathcal{R}} \overset{\mathbf{w}}{\rightarrowtail} \mathrm{id}_{\mathcal{R}} \qquad \lfloor \widehat{p} \rfloor \overset{\mathbf{w}}{\rightarrowtail} \widehat{I}$$
$$\lfloor \alpha. \rfloor \overset{\mathbf{w}}{\rightarrowtail} \mathrm{id}_{\mathcal{R}} \qquad \lfloor (\nu a) \rfloor \overset{\mathbf{w}}{\rightarrowtail} \lfloor (\nu a) \rfloor$$
$$\lfloor | \rfloor \overset{\mathbf{w}}{\rightarrowtail} \lfloor | \rfloor \qquad \lfloor ! \rfloor \overset{\mathbf{w}}{\rightarrowtail} \mathcal{C}^{-}_{ccs} \,\hat{}\, \widehat{\sim}$$

Démonstration. La preuve ne diffère de celle du lemme 5.18 que par le cas de la réplication, que nous détaillons. Soient R, S deux relations binaires telles que $R \rightsquigarrow_{\mathbf{w}} S$, il nous faut montrer $\lfloor ! \rfloor(R) \rightsquigarrow_{\mathbf{w}} \mathcal{C}^{-}_{ccs}(S) \cdot \sim$. Supposons que $p \; R \; q$ et $!p \xrightarrow{\alpha} p'$; d'après le lemme 5.17, il y a deux cas :

- $p' = !p|p^k|p_0|p^{k'}$ avec $p \xrightarrow{\alpha} p_0$. Comme $R \overset{\mathbf{w}}{\rightarrowtail} S$, on en déduit $q \overset{\widehat{\alpha}}{\Rightarrow} q_0$ avec $p_0 \; S \; q_0$; on distingue à nouveau deux cas :
  - $q \overset{\alpha}{\Rightarrow} q_0$, et on vérifie que $!q \overset{\alpha}{\Rightarrow} q' = !q|q^k|q_0|q^{k'}$, où $p' \; \mathcal{C}^{-}_{ccs}(S) \; q'$.
  - $q = q_0$ (et $\alpha = \tau$), dans ce cas, $!q$ ne peut pas bouger, c'est ici que l'on doit raisonner modulo dépliage de la réplication : $!q \sim q' = !q|q^{k+1+k'}$, et $p' \; \mathcal{C}^{-}_{ccs}(S) \; q' \sim !q$.
- $p' = !p|p^k|p_0|p^{k'}|p_1|p^{k''}$ avec $p \xrightarrow{a} p_0$ et $p \xrightarrow{\overline{a}} p_1$ ($\alpha = \tau$). Comme $R \overset{\mathbf{w}}{\rightarrowtail} S$, on en déduit $q \overset{a}{\Rightarrow} q_0$ et $q \overset{\overline{a}}{\Rightarrow} q_1$ avec $p_0 \; S \; q_0$ et $p_1 \; S \; q_1$. On vérifie alors que $!q \overset{\tau}{\Rightarrow} q' = !q|q^k|q_0|q^{k'}|q_1|q^{k''}$, où $p' \; \mathcal{C}^{-}_{ccs}(S) \; q'$. ■

Mais on est sorti du cadre du corollaire 5.12 : la fonction $\mathcal{C}^{-}_{ccs} \,\hat{}\, \widehat{\sim}$ n'est pas contenue dans la clôture $\mathcal{C}^{-}_{ccs}$. C'est ici que sert le corollaire 5.13 (qui provient du théorème 5.8) : on montre que $\lfloor C^{-}_{i} \rfloor$ est compatible avec $\mathbf{w}$ modulo itération et bisimilarité forte :

Théorème 5.24. *La fonction $R \mapsto \sim \cdot \mathcal{C}^{-}_{ccs}(R) \cdot \sim$ est une clôture compatible avec $\mathbf{w}$ et $\mathrm{id}^{\star}_{X}$.*

Démonstration. Soit $g : R \mapsto \sim \cdot R \cdot \sim$; g est une clôture, compatible avec $\mathbf{w}$ par le lemme 3.19 et la proposition 3.17(2). Pour tout contexte $c \in \mathcal{C}^{-}_{ccs}$, on a $\lfloor c \rfloor(\sim) \subseteq \mathcal{C}_{ccs}(\sim) = \sim$, dont découle la compatibilité de $\lfloor c \rfloor$ avec g.

On peut donc appliquer le corollaire 5.13 : dans le lemme précédent, toutes les fonctions vers lesquelles progressent les fonctions associées aux contextes de C^{-}_{i} sont contenues dans $g \circ \mathcal{C}^{-}_{ccs}$. La fonction $g \circ \mathcal{C}^{-}_{ccs}$ est donc une clôture compatible avec $\mathbf{w}$. On vérifie ensuite aisément qu'elle est compatible avec $\mathrm{id}^{\star}_{X}$. ■

Dans ce cas particulier, la fonction qu'il nous faut utiliser pour compléter la clôture est compatible ; si elle n'était que correcte, il faudrait utiliser la seconde partie du théorème 5.8.

$$R = \left\{ \langle a, (\nu b)(b.a|\overline{b}) \rangle, \langle b.a, b \rangle, \langle (\nu b)(b|\overline{b}), \mathbf{0} \rangle, \langle (\nu b)\mathbf{0}, \mathbf{0} \rangle \right\}$$
$$c : p \mapsto (\nu b)(p|\overline{b})$$

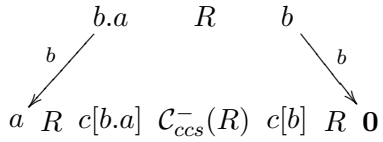


FIGURE 5.2 – La clôture $\mathcal{C}^{-}_{ccs}$ n'est pas $\mathbf{w}_{\mathrm{t}}$-correcte.

5.3.3 Un résultat négatif

Le théorème précédent implique en particulier la correction de la clôture $\mathcal{C}^{-}_{ccs}$ pour $\mathbf{w}$. De manière assez surprenante, cette clôture n'est pas correcte pour $\mathbf{w}_{\mathrm{t}}$, la génératrice des simulations faibles qui autorise modulo transitivité lors des offres visibles. Un contre-exemple [1] est donné sur la figure 5.3.3, où R n'est pas contenue dans la $\mathbf{w}_{\mathrm{t}}$-similarité ($\approx$) bien que R soit une $(\mathbf{w}_{\mathrm{t}} \circ \mathcal{C}^{-}_{ccs})$-simulation.

Lorsque l'on considère la preuve, le problème apparaît avec l'opérateur de mise en parallèle : on n'a pas $\lfloor | \rfloor \overset{\mathbf{w}_{\mathrm{t}}}{\longmapsto} \mathcal{C}^{-}_{ccs}$. Intuitivement, comme la composition parallèle est capable de « transformer » deux actions visibles en une action silencieuse, elle ramène la transitivité des offres visibles – où elle est autorisée par $\mathbf{w}_{\mathrm{t}}$ – vers les offres silencieuses – où elle ne l'est pas. Plus précisément, on se retrouve bloqué dans le cas d'une communication, lorsque l'on doit montrer que $R \rightsquigarrow_{\mathbf{w}_{\mathrm{t}}} S$ implique $\lfloor | \rfloor(R) \rightsquigarrow_{\mathbf{w}_{\mathrm{t}}} \mathcal{C}^{-}_{ccs}(S)$: en utilisant les notations de la preuve du lemme 5.18, on a $u' = p'_1|p'_2$ avec $p_1 \xrightarrow{a} p'_1$, $p_2 \xrightarrow{\overline{a}} p'_2$, et $\alpha = \tau$; cependant, l'hypothèse $R \rightsquigarrow_{\mathbf{w}_{\mathrm{t}}} S$ donne $q_1 \overset{a}{\Rightarrow} q'_1$ $q_2 \overset{\overline{a}}{\Rightarrow} q'_2$ avec $p'_1 \; S^{\star} \; q'_1$ et $p'_2 \; S^{\star} \; q'_2$ (plutôt que $p'_1 \; S \; q'_1$ et $p'_2 \; S \; q'_2$ avec la génératrice $\mathbf{w}$) ; on a donc $v \overset{\tau}{\Rightarrow} v' = q'_1|q'_2$ avec $u' \; \lfloor | \rfloor(S^{\star}) \; v'$; mais comme on répond à une offre silencieuse, il nous faudrait prouver que $u' \; \lfloor | \rfloor(S) \; v'$.

Ce contre-exemple montre qu'il existe un compromis entre la possibilité d'utiliser modulo contexte, et celle d'utiliser modulo transitivité lors des offres visibles. Plus généralement, comme annoncé à la section 2.2.2, on s'aperçoit à nouveau qu'à deux fonctions générant la même similarité ($\mathbf{w}$ et $\mathbf{w}_{\mathrm{t}}$ génèrent toutes deux $\approx$) peuvent correspondre des fonctions compatibles ou correctes complètement différentes ; de plus on retrouve le fait que la similarité soit close par une fonction ne suffit pas à ce que cette fonction soit correcte : la bisimilarité faible est close par $\mathcal{C}^{-}_{ccs}$.

Notons toutefois qu'une partie des techniques modulo contexte est com-

1. Ce contre-exemple provient d'un phénomène similaire découvert par Davide Sangiorgi, dans le cadre du λ-calcul [San06].

patible avec $\mathbf{w}_\mathrm{t}$; contrairement à ce que pourrait laisser penser le contre-exemple de la figure 5.3.3, modulo restriction, qui prend une importance particulière dans le cas du π-calcul, ne pose par exemple aucun problème : on a $\lfloor(\nu a)\rfloor \overset{\mathbf{w}_\mathrm{t}}{\rightarrowtail} \lfloor(\nu a)\rfloor$.

Ce contre-exemple montre aussi que du point de vue des techniques modulo, la bisimilarité faible diffère de la « bisimilarité forte sur le LTS faible » (remarque 3.14) : la relation R de la figure 5.3.3 satisfait aussi $\forall\alpha,\ \overset{\widehat{\alpha}}{\Longleftarrow} \cdot R \subseteq \mathcal{C}^-_{ccs}(R)^\star \cdot \overset{\widehat{\alpha}}{\Longleftarrow}$; ce qui en fait une simulation *forte* modulo contexte et transitivité, dans le LTS faible. Si l'on reprend la preuve de modulo contexte dans cette optique, le problème survient aussi avec la composition parallèle : on n'a plus $\lfloor|\rfloor \overset{\mathbf{s}}{\rightarrowtail} \lfloor|\rfloor$ lorsque l'on considère les transitions faibles, mais pas pour les mêmes raisons que précédemment. Supposons $R \overset{\mathbf{s}}{\rightarrowtail} S$, $p_1\ R\ q_1$, $p_2\ R\ q_2$, et $p_1 \mid p_2 \overset{\widehat{\alpha}}{\Rightarrow} p'$; il nous faut trouver q' tel que $q_1 \mid q_2 \overset{\widehat{\alpha}}{\Rightarrow} q'$ et $p'\ S\ q'$. Le problème est que la transition faible qu'effectue $p_1 \mid p_2$ peut provenir de deux séquences presque arbitraires de transitions de p_1 et p_2 (par exemple, si $p_1 \overset{a}{\Rightarrow} p'_1 \overset{b}{\Rightarrow} p''_1$ et $p_2 \overset{\overline{a}}{\Rightarrow} p'_2$, on a $p_1 \mid p_2 \overset{b}{\Rightarrow} p''_1 \mid p'_2$). Du coup, le fait de savoir $R \rightsquigarrow_\mathbf{s} S$ ne suffit pas pour pouvoir mimer ces séquences du coté droit : seules les premières transitions visibles (faibles) de p_1 et p_2 sont contrôlées (on obtient q'_1 et q'_2 tels que $q_1 \overset{a}{\Rightarrow} q'_1$, $q_2 \overset{\overline{a}}{\Rightarrow} q'_2$, $p'_1\ S\ q'_1$ et $p'_2\ S\ q'_2$, mais on n'a pas d'information pour obtenir q''_1 tel que $q'_1 \overset{b}{\Rightarrow} q''_1$ et $p''_2\ S\ q''_2$).

5.3.4 Ajout de la somme

Le fait que la bisimilarité faible ne soit pas close par l'opérateur de choix ($+$) n'est qu'une « petite irrégularité » : cette équivalence est « presque une congruence », et il est facile de contourner le problème, notamment pour ce qui est des techniques modulo contexte. Nous montrons rapidement comment s'y prendre en utilisant notre méthode.

Regardons tout d'abord pourquoi la preuve échoue : reprenons comme précédemment $R = \{\langle \tau.a, a\rangle\} \cup I$; bien que $R \rightsquigarrow_\mathbf{w} R$, on n'a pas $\lfloor+\rfloor(R) \rightsquigarrow_\mathbf{w} \mathcal{C}_{ccs}(R)$: l'offre ci-dessous n'admet pas de réponse puisque $a+b$ ne peut pas choisir silencieusement a :

$$\begin{array}{ccc} \tau.a+b & \lfloor+\rfloor(R) & a+b \\ \big\downarrow{\scriptstyle \tau} & & \\ a & & ? \end{array}$$

Comme précédemment pour la réplication, le problème est en quelque sorte dual de celui de la transitivité : c'est le fait que le processus de droite ne réponde pas suffisamment de transitions silencieuses qui est problématique.

Nous verrons en particulier que ce problème ne se pose plus avec la bisimulation progressive.

L'idée standard [SW01] consiste à d'interdire les contextes qui utilisent leur arguments en position *dégénérée*, c'est-à-dire directement en tant qu'opérande d'une somme. On appelle donc *contexte non dégénéré* tout processus numéroté dont toute occurrence de $\mathbf{0}$ apparaissant directement en tant qu'opérande d'une somme est étiquetée par 0. Pour les contextes initiaux, cette idée se traduit par la définition suivante :

Définition 5.25 (Contextes initiaux non dégénérés)**.** Pour tous $n \in \mathbb{N}$, $r \in \mathcal{P}$, et $\alpha_1, \ldots, \alpha_n \in \mathcal{L}$, on définit le *contexte initial non dégénéré* suivant :

$$r + \Sigma_i^n\ \alpha_i. : p_1, \ldots, p_n \mapsto r + (\alpha_1.p_1 + (\cdots + (\alpha_n.p_n)\ldots))\ .$$

On note C_i^{nd} l'ensemble C_i où l'on remplace le contexte somme par l'ensemble des contextes précédemment définis, et on appelle *clôture par les contextes non dégénérés de CCS* la fonction $\mathcal{C}_{ccs}^{nd} \triangleq \lfloor C_i^{nd} \rfloor^{\omega}$.

$$C_i^{nd} \triangleq C_i^{-} \cup \{r + \Sigma_i^n\ \alpha_i. \mid n \in \mathbb{N}\ , r \in \mathcal{P},\ \alpha_1, \ldots, \alpha_n \in \mathcal{L}\}\ .$$

Notons que l'on n'obtient pas tout à fait l'ensemble des contextes non dégénérés ($\lfloor C_i^{nd} \rfloor$) avec $\mathcal{C}_{ccs}^{nd}$: il faudrait définir les contextes initiaux non dégénérés modulo associativité et commutativité de la somme (par exemple, on ne capture pas le contexte $p \mapsto \tau.p + a$). Ce n'est cependant pas restrictif puisque l'on doit de toute façon composer cette clôture avec la fonction $R \mapsto \sim \cdot R \cdot \sim$ qui permet, entre autres, de raisonner modulo ces lois de congruence structurelle.

Théorème 5.26. *La fonction $R \mapsto \sim \cdot \mathcal{C}_{ccs}^{nd}(R) \cdot \sim$ est une clôture compatible avec* $\mathbf{w}$ *et* $\mathrm{id}_X^{\star}$.

Démonstration. On a $\lfloor r + \Sigma_i^n\ \alpha_i. \rfloor \overset{\mathbf{w}}{\rightarrowtail} \mathrm{id}_{\mathcal{R}}^{=}$; il suffit ensuite de reprendre la preuve du théorème 5.24. ■

5.3.5 Expansion, élaboration et bisimilarité progressive

Les techniques modulo contexte que nous venons d'étudier pour les bisimilarités fortes et faibles s'adaptent très naturellement à l'ensemble des préordres et équivalences intermédiaires. Le théorème suivant récapitule l'ensemble des résultats de compatibilité ainsi obtenus ; nous ne donnons pas les preuves détaillées, qui sont presque identiques aux précédentes ; nous soulignons simplement les particularités.

Théorème 5.27. *1. La clôture $\mathcal{C}_{ccs}$ est compatible avec* $\mathbf{s}$, $\mathbf{p}$ *et* $\mathrm{id}_X^{\star}$.

2. La clôture $R \mapsto \sim \cdot \mathcal{C}_{ccs}^{nd}(R) \cdot \sim$ est compatible avec $\mathbf{s}$, $\mathbf{e}$, $\mathbf{p}$, $\mathbf{w}$ *et* $\mathrm{id}_X^{\star}$.

Démonstration. La compatibilité avec $\mathrm{id}^\star_X$ s'obtient dans les deux cas via la proposition 5.14.

1. On a déjà démontré la compatibilité avec **s** (théorème 5.19), la preuve pour **p** est identique. Il n'y a notamment pas de problème avec la somme et la réplication : cette génératrice demande au processus de répondre toujours au moins une action, ce qui empêche les deux situations pathologiques.
2. On a déjà démontré la compatibilité avec **w** (théorème 5.26) ; la preuve pour l'expansion est identique (le fait de répondre au plus une action ne permet pas de se passer du raisonnement modulo $\sim$) ; le cas des deux autres génératrices ne découle pas du point précédent, mais leurs preuves reviennent au même. ■

En utilisant la proposition 2.56, on obtient donc des techniques modulo contexte pour chacun des préordres ou équivalences combinant un jeu de gauche à droite et un jeu de droite à gauche, ainsi que les propriétés de congruence suivantes :

Corollaire 5.28. *Dans CCS, la bisimilarité forte ($\sim$) et la bisimilarité progressive ($\simeq$) sont des congruences ; l'expansion ($\succsim$), la bi-expansion ($\asymp$), l'élaboration ($\succapprox$) et la bisimilarité faible ($\approx$) sont closes par les contextes non dégénérés.*

Ce dernier corollaire fournit un résultat annoncé au chapitre précédent : lorsque l'on souhaite utiliser l'élaboration en tant que relation contrôlée (corollaire 4.47), et que l'on veut la combiner à modulo contexte (corollaire 3.49), la condition entre la relation contrôlée et la clôture compatible est déjà satisfaite : on a

$$\sim \cdot \mathcal{C}^{nd}_{ccs}(\succapprox) \cdot \sim \subseteq \sim \cdot \succapprox \cdot \sim \subseteq \succapprox \cdot \succapprox \cdot \succapprox \subseteq \succapprox \ .$$

La congruence de la bisimilarité progressive mène à un résultat de Montanari et Sassone [MS92c] :

Corollaire 5.29. *Dans CCS, la bisimilarité progressive est la plus grande bisimulation faible qui soit en même temps une congruence.*

Démonstration. On vient d'établir que $\simeq$ est une congruence ; c'est une bisimulation faible par la proposition 3.12.

Il nous faut donc simplement montrer que toute bisimulation faible qui est une congruence est contenue dans $\simeq$; il suffit pour cela que toute simulation faible qui est une congruence soit une simulation progressive. Soit R une telle simulation faible. Le seul cas non trivial à traiter est celui d'une offre silencieuse, où les définitions de simulation faible et progressive diffèrent : soit p, p', q tels que $p \ R \ q$, $p \xrightarrow{\tau} p'$ et $p' \ R \ q'$, nous devons trouver q' tel que $p' \ R \ q'$ et $q \stackrel{\tau}{\Rightarrow} q'$. Soit pour cela $a \in \mathcal{L}^{\mathrm{v}}$ une étiquette quelconque. Comme R

est une congruence, on a $a+p\ R\ a+q$; comme c'est une simulation faible, on en déduit l'existence d'un processus q' tel que $a+q \stackrel{\widehat{\tau}}{\Rightarrow} q'$ et $p'\ R\ q'$. Comme $\tau \neq a$, on a nécessairement $q \stackrel{\tau}{\Rightarrow} q'$. ■

Notons pour l'anecdote que la preuve donnée dans [MS92c] de la première partie de ce résultat (la bisimulation progressive est une congruence) est fausse : elle s'appuie sur la technique « faible modulo transitivité » pour traiter le cas de la récursion. Le passage par les contextes initiaux et l'itération fonctionnelle nous évite ce faux pas.

5.4 Formats de règles

Nous concluons ce chapitre en citant quelques travaux qui pourraient nous permettre de développer un peu plus la méthode des contextes initiaux.

Plusieurs « formats » de règles ont été définis dans la littérature, qui permettent d'assurer que la bisimilarité forte correspondant aux LTS ainsi obtenus est une congruence : *de Simone* [dS85], *GSOS* [BIM95], *tyft/tyxt* [GV92], *panth* [Ver95], etc... Comme on vient de le voir, en déduire des techniques modulo contexte peut s'avérer plus délicat : avoir une congruence ne suffit pas pour avoir des techniques modulo contexte, cela dépend de la génératrice considérée, pas seulement de son plus grand point fixe.

Sangiorgi a montré que l'on obtenait automatiquement des techniques modulo contexte dans un cas particulier du format de Simone [San98]. En utilisant un résultat de Fokkink et Glabbeek [FvG96], qui permet de ramener l'étude du format tyft/tyxt à celle d'un format plus simple (appelé « arbre »), nous pourrions utiliser la méthode des contextes initiaux et montrer facilement que l'on obtient aussi des techniques modulo avec le format tyft/tyxt. Nous envisageons d'étudier si ce genre de résultats peut s'étendre au cas faible ; cependant, l'irrégularité que l'on a décelée avec l'opérateur de réplication, et qui nous force à raisonner modulo $\sim$, nous semble assez difficile a traiter de façon générique.

Chapitre 6

Etude d'un modèle d'exécution pour les calculs distribués

Nous illustrons dans ce chapitre l'utilisation des techniques modulo présentées au chapitres 3 et 4, en prouvant la correction de deux implantations possibles d'un modèle d'exécution pour des processus distribués. La première est relativement simple, de telle sorte que les techniques standard, qui reposent sur l'expansion, sont suffisantes ; au contraire, la seconde, qui est essentiellement une optimisation de la première, requiert des techniques plus poussées.

Cet exemple provient de notre travail de DEA [HPS05], où l'on a prouvé la correction d'une optimisation de la PAN, une machine abstraite définie par Sangiorgi et Valente [SV01], permettant d'exécuter de manière distribuée le calcul des *Safe Ambients* [LS00]. Ici, nous avons retiré tout ce qui était spécifique aux *ambients* : il ne reste qu'un modèle d'exécution, qui nous permettrait en particulier de programmer la PAN (ainsi, lorsque l'on optimise le modèle d'exécution, on optimise en particulier la PAN). Si l'on retire de la même manière tout ce qui a trait aux ambients dans l'implantation que nous avons écrite pour la GCPAN [Pou05a], on obtient une librairie OCAML permettant d'écrire des programmes distribués ; optimiser le modèle correspond à optimiser cette librairie.

Pour rendre compte de la possibilité de distribuer l'exécution sur plusieurs machines physiques, ce modèle définit une notion de *localité* (une entité logique indépendante, pouvant héberger des calculs), et procède uniquement par échange asynchrone de *messages* entre ces différentes localités. Les processus hébergés dans les différentes localités s'exécutent ainsi indépendamment les uns des autres, de façon asynchrone. L'une des particularités du calcul des Safe Ambients est la primitive, emblématique, de *migration* : un processus peut se déplacer de localité en localité. On retrouve donc naturellement cette primitive dans notre modèle d'exécution. Comme les localités (logiques), peuvent être arbitrairement distribuées sur diverses machines (lo-

calités physiques), cela nécessite l'introduction de *messagers* (« forwarders » en anglais), dont le rôle est de transmettre tout message atteignant une localité d'où un processus vient de migrer vers la localité où ce dernier a migré.

Le comportement naïf de ces messagers est inefficace : lorsqu'un processus migre plusieurs fois, il laisse derrière lui une chaîne de messagers, dont le comportement devient de moins en moins efficace. Plus le processus « s'éloigne », plus la longueur de la chaîne ralentit l'acheminement des messages, et plus elle génère de trafic sur le réseau. D'autre part, les messagers deviennent souvent inutiles, par exemple lorsque tous les processus se sont rendus compte de la migration du processus concerné, et qu'ils envoient leurs messages directement vers la bonne destination.

L'optimisation que l'on a définie pour la PAN a consisté d'une part à définir un algorithme distribué de contraction des chaînes de messagers (ressemblant à l'algorithme « union-find » de Tarjan), et d'autre part à installer un mécanisme de compteurs, permettant de retirer les messagers après qu'il sont devenus inutiles. La partie « nettoyage » de cette optimisation est très spécifique à la PAN : la façon dont on s'assure que les compteurs peuvent être tenus à jour de manière distribuée dépend fortement de l'algorithme utilisé dans la PAN pour interpréter les primitives des Safe Ambients. Au niveau du modèle d'exécution plus abstrait sur lequel on se focalise, nous nous contentons donc de définir et valider l'algorithme de contraction des chaînes de messagers.

Bien que l'on présente ici ces travaux pour illustrer l'utilisation de nos nouvelles techniques modulo, la définition et l'étude de tels modèles d'exécution est relativement importante en soi : de nombreux calculs incluant des primitives de distribution et de mobilité on été proposés ces dernières années (Join [FFMS02], Dπ [Hen07], Nomadic pict [US01], Kells [SS04], Mobile ambients [CG98], Klaim [NFP98], Seals [CN02], etc), dans l'idée d'établir un fondement théorique pour des langages de programmation plus généraux. Mais l'expressivité fournie par les primitives qui sous-tendent ces modèles pose réellement la question de leur implantabilité dans les systèmes distribués (physiquement). Le modèle que nous présentons ici est réaliste de ce point de vue : nous avons implanté de façon distribuée la PAN et son optimisation (la GCPAN), en OCAML [LW85], ce qui nous a permis de comparer de façon empirique les deux versions de la machine [Pou05a].

Nous commençons par spécifier le modèle d'exécution à l'aide de la notion de LTS (section 6.1) ; nous définissons et validons ensuite une première implantation (section 6.2), puis nous étudions son optimisation à la section 6.3.

6.1 Les LTS comme outil de modélisation

Nous décrivons maintenant un peu plus précisément la méthodologie que l'on va employer pour définir le modèle d'exécution. Nous voulons représenter

l'exécution d'une collection de *processus*, distribués selon un ensemble de *localités*, où chaque processus peut :

- évoluer par lui-même, indépendamment des processus hébergés par les autres localités ;
- envoyer des messages à chacun des autres processus, identifiés par leur localité (une localité a intuitivement le rôle d'une adresse) ;
- recevoir ces messages ;
- démarrer de nouveaux processus, dans de nouvelles localités ;
- migrer (déménager) vers une localité existante, et y recevoir les messages envoyés à l'ancienne adresse.

On abstrait sur la structure des processus en les représentant par les états d'un LTS quelconque, dont les étiquettes correspondent aux primitives précédentes. L'ensemble de ces processus, avec l'ensemble des messages qu'ils peuvent échanger, sont ainsi des paramètres du modèle d'exécution. Nous ne faisons aucune hypothèse sur le comportement des processus ni sur la nature des messages qu'ils échangent. Les messages peuvent notamment contenir des localités, des noms ou des processus, de telle sorte que l'ordre supérieur et la mobilité des liens (telle qu'elle est présente dans le π-calcul) sont implicitement autorisés. De même les processus peuvent être séquentiels ou concurrents, selon les possibilités fournies par les localités considérées.

En définissant des étiquettes complémentaires à celles des processus, on obtient une famille de LTS, que nous appelons *modèles*, dont les états sont appelés *réseaux* (le terme « modèle » est très générique, nous l'entendons implicitement comme dans « modèle d'exécution » ou « modèle de réseaux »). Ces étiquettes correspondent aux réactions du réseau vis-à-vis des requêtes des processus. D'un point de vue « programmation », elles forment l'interface d'une librairie, à laquelle les processus peuvent être « liés ». Avant de définir le « corps » de telles librairies, nous montrons comment y lier les processus, en définissant une opération de composition de deux LTS, l'un étant le modèle et l'autre l'ensemble des processus. Intuitivement, le *LTS composite* que l'on obtient exécute les processus de manière concurrente, et réagit aux requêtes de ces derniers : transmettre un message, créer une nouvelle localité , etc... Cette approche nous permet de séparer complètement l'étude des processus de celle des réseaux : la bisimilarité faible, comme les autres équivalences comportementales, est préservée par cette construction (proposition 6.4).

On peut ensuite étudier plusieurs implantations possibles : il suffit de définir des LTS concrets, qui décrivent réellement le comportement du réseau, et pas seulement son interface. Nous commençons par définir une spécification, en donnant un *modèle de référence*, qui décrit de façon relativement directe le comportement attendu d'un réseau mais ne peut pas être implanté directement de façon distribuée. Nous définissons ensuite un *modèle simple*, qui correspond à une implantation distribuée relativement simpliste et que nous raffinons ensuite en un *modèle optimisé* où les chaînes de messagers sont

contractées. Nous montrons que les réseaux des trois modèles sont faiblement bisimilaires, ce qui valide l'implantation et son optimisation.

On aura besoin de pouvoir parler facilement de différents LTS dont les étiquettes ne sont pas toujours les mêmes. On introduit pour cela la définition suivante, qui correspond bien à la définition 3.8 dans le cas d'une algèbre relationnelle propre :

Définition 6.1 (Système de $\mathcal{L}$-transitions ($\mathcal{L}$-TS))**.** Soit $\mathcal{L}$ un ensemble d'étiquettes. Un *système de $\mathcal{L}$-transitions ($\mathcal{L}$-TS)* est un couple $\langle \mathcal{P}, \hookrightarrow \rangle$ où $\mathcal{P}$ est un ensemble arbitraire d'états, appelé *domaine*, et $\hookrightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ un ensemble de transitions étiquetées par $\mathcal{L}$.

On notera $p \overset{\alpha}{\hookrightarrow} p'$ lorsque $\langle p, \alpha, p' \rangle \in \hookrightarrow$.

6.1.1 Interface du modèle d'exécution

Pour décrire le modèle d'exécution, il nous faut tout d'abord nommer les « primitives » que l'on souhaite fournir aux processus à exécuter. On se donne pour cela :

- un ensemble $\mathcal{H}$ de *localités*, dont les éléments seront notés $h, k \ldots$;
- un ensemble de *messages élémentaires*, dont les multi-ensembles finis forment des *messages* $m, n \ldots$, appartenant à un ensemble noté $\mathcal{M}$;
- un ensemble d'*actions locales* $\mathcal{L}$ (contenant l'action distinguée τ), notées $\alpha, \beta \ldots$ comme aux chapitres précédents ;
- un ensemble $\mathcal{P}$ de *processus* $p, q \ldots$; ceux que l'on souhaite exécuter.

Puis on définit l'ensemble $\mathcal{L}_p$ des *actions des processus* par la grammaire suivante :

$$
\begin{array}{ll}
\delta ::= \ \alpha & \text{(action locale)} \\
\mid h\langle m \rangle & \text{(émission d'un message)} \\
\mid (m) & \text{(réception d'un message)} \\
\mid \ \triangleright h & \text{(migration)} \\
\mid \nu h[p] & \text{(création d'une localité)}
\end{array}
$$

On fixe alors un $\mathcal{L}_p$-TS de domaine $\mathcal{P}$, qui définit les transitions des processus : $\langle \mathcal{P}, \hookrightarrow \rangle$. Les actions de ces processus correspondent aux primitives que l'on a présentées plus haut : α correspond à une transition locale et indépendante d'un processus, $h\langle m \rangle$ à l'émission d'un message m vers la localité h, (m) à la réception d'un message m, $\triangleright h$ à la migration vers une localité h, et $\nu h[p]$ au déploiement d'un processus p dans une nouvelle localité h.

Un *processus distribué* est une fonction de $\mathcal{H}$ dans $\mathcal{P}$, à support fini, que l'on notera de la façon générique suivante :

$$h_1 : p_1, \ldots, h_n : p_n \ .$$

$$[\textsc{Loc}_\otimes]\ \frac{p \xrightarrow{\alpha} p'}{U \otimes D, h : p \overset{\alpha}{\mapsto} U \otimes D, h : p'} \qquad [\textsc{Int}_\otimes]\ \frac{U \xrightarrow{\tau} U'}{U \otimes D \overset{\tau}{\mapsto} U' \otimes D}$$

$$[\textsc{Snd}_\otimes]\ \frac{U \xrightarrow{k\langle m\rangle} U' \qquad p \xrightarrow{k\langle m\rangle} p'}{U \otimes D, h : p \overset{\tau}{\mapsto} U' \otimes D, h : p'}$$

$$[\textsc{Rcv}_\otimes]\ \frac{U \xrightarrow{h(m)} U' \qquad p \xrightarrow{(m)} p'}{U \otimes D, h : p \overset{\tau}{\mapsto} U' \otimes D, h : p'}$$

$$[\textsc{New}_\otimes]\ \frac{U \xrightarrow{\nu k} U' \qquad p \xrightarrow{\nu k[q]} p'}{U \otimes D, h : p \overset{\tau}{\mapsto} U' \otimes D, h : p', k : q}$$

$$[\textsc{Mig}_\otimes]\ \frac{U \xrightarrow{h \triangleright k} U' \qquad p \xrightarrow{\triangleright k} p'}{U \otimes D, h : p \overset{\tau}{\mapsto} U' \otimes D}$$

FIGURE 6.1 – Transitions du LTS composite.

Dans ce cas, on dira que le processus p_i *est hébergé par la localité* h_i (ou *en* h_i). L'ensemble des processus distribués, dont les éléments sont notés D, est noté $\mathcal{DP}$. Afin de représenter l'exécution d'un processus distribué, on définit une notion de *réseau*, et on montre comment composer un tel réseau avec un processus distribué.

Définition 6.2 (Modèle, réseau)**.** Un *modèle* est un $\mathcal{L}_r$-TS, où $\mathcal{L}_r$ est l'ensemble des *actions du réseau*, définies ci-dessous ; un *réseau* est un état d'un modèle. Les réseaux seront dénotés par $U, V \ldots$

$$\begin{array}{ll}
\mu ::= \ \tau & \text{(action silencieuse, interne)} \\
\mid h\langle m\rangle & \text{(émission d'un message)} \\
\mid h(m) & \text{(réception d'un message)} \\
\mid h \triangleright k & \text{(migration)} \\
\mid \nu h & \text{(création d'une localité)}
\end{array}$$

Les actions du réseau correspondent étroitement aux actions des processus : ces actions sont appariées afin de définir le LTS composite ci-dessous, qui décrit l'exécution d'un processus distribué dans un réseau arbitraire.

Définition 6.3 (Composition d'un réseau avec un processus distribué)**.** Soit $\langle \mathcal{U}, \rightarrow \rangle$ un modèle. On appelle *$\mathcal{L}$-TS composite* le $\mathcal{L}$-TS $\langle \mathcal{U} \times \mathcal{DP}, \mapsto \rangle$ où la relation $\mapsto$ est définie par les règles d'inférence de la figure 6.1 (les éléments du domaine sont notés $U \otimes D$).

Par la règle [$\text{Loc}_\otimes$], un processus peut évoluer indépendamment des autres ; de manière similaire, le réseau peut effectuer des transitions silencieuses grâce à la règle [$\text{Int}_\otimes$]. Un processus hébergé en h peut envoyer un message vers la localité k par la règle [$\text{Snd}_\otimes$], et recevoir des messages par la règle [$\text{Rcv}_\otimes$]. La règle [$\text{New}_\otimes$] permet à un processus hébergé en h de lancer l'exécution d'un processus q dans une nouvelle localité, k. Enfin, par la règle [$\text{Mig}_\otimes$], un processus hébergé en h peut « migrer » vers une localité k. Cette dernière règle peut surprendre, puisque la « continuation » (p') du processus qui migre est perdue, au lieu d'être relancée dans la nouvelle localité (k). Ce n'est pas restrictif, puisque cette continuation peut être envoyée par un message avant la migration, et cela nous permet de ne pas avoir à supposer une opération de composition sur les processus (sans cela, il faudrait pouvoir faire cohabiter la continuation et le processus déjà hébergé en k). De même, le fait de ne pouvoir migrer que vers des localités existantes n'est pas gênant : un processus peut toujours créer une nouvelle localité avant d'y migrer.

Les seules transitions visibles du LTS composite proviennent des processus, par la règle [$\text{Loc}_\otimes$]. Le réseau n'interagit donc pas avec « l'extérieur », et ne peut être observé qu'indirectement, par son interaction avec le processus distribué.

Remarquons que la communication est effectuée par unification (la règle [Rcv_s] demande que le réseau et le processus s'accordent sur un même message, m) ; ceci impose implicitement une sémantique *précoce* (« early » en anglais [SW01]) aux processus. Adapter cette règle afin de supporter une sémantique *retardée* (« late ») des processus est immédiat, mais cela suppose de définir une notion de substitution sur les processus. Remarquons aussi que bien que l'on échange des multi-ensembles de messages (cela simplifie les notations dans la suite), le fait d'avoir une sémantique précoce permet à un processus de décider de ne recevoir que des messages élémentaires.

Comme l'exprime la proposition suivante, l'avantage principal de cette approche, où les processus sont complètement séparés du modèle, est que l'on peut étudier les deux entités indépendamment. On pourra donc dans la suite ne s'intéresser qu'aux modèles.

Proposition 6.4. *Soient U, V deux réseaux, p, q deux processus et D un processus distribué.*

- *Si $U \approx V$, alors $U \otimes D \approx V \otimes D$.*
- *Si $p \approx q$, alors $U \otimes D, h : p \approx U \otimes D, h : q$.*

Démonstration. On démontre d'abord que pour tous réseaux U, U', processus p, p', processus distribué D, et localité h, on a :

- $U \xrightarrow{\tau}{}^\star U'$ implique $U \otimes D \mapsto^{\tau\star} U' \otimes D$, et
- $p \hookrightarrow^{\tau\star} p'$ implique $U \otimes D, h : p \mapsto^{\tau\star} U \otimes D, h : p'$.

On vérifie ensuite facilement que les deux relations ci-dessous sont des bisimulations faibles :

$$R_1 \triangleq \{\langle U \otimes D,\ V \otimes D\rangle \mid \forall U, V,\ U \approx V\} \quad ,$$
$$R_2 \triangleq \{\langle U \otimes D, h : p,\ U \otimes D, h : q\rangle \mid \forall U, p, q,\ p \approx q\} \cup I \quad . \qquad \blacksquare$$

Notons que cette proposition reste vraie pour les préordres comportementaux plus forts (expansion, élaboration, bisimilarité forte, etc. . .)

Remarque 6.5 (Russel). Un point important de la proposition précédente est que les deux réseaux U et V n'ont pas besoin d'appartenir au même modèle : un réseau est un couple état-modèle, bien que l'on ait pris soin de ne pas rendre ceci explicite dans la définition 6.2. Mais un problème se cache derrière cet abus de notation, qui nécessite quelques explications. On mentionne en effet dans la proposition 6.4 la bisimilarité faible, en tant que relation binaire entre les réseaux. Or la bisimilarité, telle qu'on l'a définie au chapitre 3, est une relation binaire entre les états d'un même LTS.

Pour résoudre ce problème on pourrait être tenté de définir un modèle « universel », capable de représenter tous les modèles à l'aide d'un seul espace d'états. En effet, en considérant la règle d'inférence suivante, la collection des réseaux peut naturellement être équipée de transitions étiquetées : cette règle permet d'injecter les transitions entre les états de tout modèle vers des transitions entre réseaux :

$$\frac{U \overset{\mu}{\hookrightarrow} U'}{\langle U, \langle \mathcal{P}, \hookrightarrow\rangle\rangle \overset{\mu}{\rightarrow} \langle U', \langle \mathcal{P}, \hookrightarrow\rangle\rangle}$$

Mais on a ici un problème de taille : un réseau (un couple état-modèle) étant à peu près un ensemble quelconque, la collection des réseaux ne peut pas être un ensemble (la règle précédente montre qu'il s'appartiendrait lui-même).

Il nous faut donc être moins ambitieux, et considérer ici la bisimilarité dans le modèle obtenu comme l'union disjointe des cinq modèles que nous définirons par la suite. Le champ d'application de la proposition 6.4 est donc implicitement restreint à ces cinq modèles.

6.1.2 Spécification du comportement attendu

La notion de modèle ne spécifie pas le comportement des réseaux, mais seulement leur interface. Afin de spécifier le comportement attendu, on définit un *modèle de référence*, qui nous permettra ensuite de valider les deux implantations, en montrant qu'elles lui sont faiblement bisimilaires.

Trois catégories de réseaux sont définis sur la figure 6.2 ; dans un premier temps, on ne considère que les *réseaux de référence* : les deux autres extensions de cette syntaxe correspondent aux deux implantations et seront étudiées plus tard. On raisonne implicitement modulo la commutativité

Réseaux :			
de référence	*simples*	*optimisés*	
$U ::= \mathbf{0}$	$U ::= \mathbf{0}$	$U ::= \mathbf{0}$	(réseau vide)
$\mid U \mid U$	$\mid U \mid U$	$\mid U \mid U$	(composition parallèle)
$\mid h[m]$	$\mid h[m]$	$\mid h[m]$	(localité réelle)
$\mid h \triangleright k$	$\mid h \triangleright k$	$\mid h \triangleright k$	(messager)
	$\mid h\langle m \rangle$	$\mid h\langle m \rangle_{\widetilde{k}}$	(message en attente)
		$\mid h \not\triangleright$	(messager bloqué)
		$\mid h\langle \triangleright k \rangle$	(message de relocalisation)

FIGURE 6.2 – Syntaxe des réseaux de référence, simples et optimisés.

et l'associativité de la composition parallèle, qui admet $\mathbf{0}$ comme élément neutre :

$$U \mid V = V \mid U \qquad U \mid (V \mid W) = (U \mid V) \mid W \qquad \mathbf{0} \mid U = U$$

Un réseau de référence est donc plus simplement un multi-ensemble fini dont chaque élément est

- soit une *localité réelle* ($h[m]$) contenant un message m (rappelons qu'un message est en fait une collection de messages élémentaires) – intuitivement, dans le $\mathcal{L}$-TS composite (définition 6.3), les processus s'exécutent dans de telles localités ;
- soit un *messager* ($h \triangleright k$), dont le rôle est de rediriger tout message arrivant en h vers k.

Les localités représentent intuitivement des entités indépendantes, hébergeant des *agents*, qu'il nous faut être capable d'identifier de manière unique. C'est ce qui est exprimé par la propriété de bonne formation suivante :

Définition 6.6 (Localité définie, bonne formation)**.** Soit U un réseau de référence.

- Une localité $h \in \mathcal{H}$ est *définie par* U lorsqu'il existe m, V, k tels que $U = h[m] \mid V$ ou $U = h \triangleright k \mid V$. On note $l(U)$ l'ensemble des localités définies par U.
- U est *bien formé* si toute localité est définie au plus une fois (i.e., si elle y apparaît au plus une fois, comme une localité réelle ou comme la source d'un messager), et si pour tout messager $h \triangleright k$ qu'il contient, la localité k est définie.

Un réseau bien formé pourrait donc être défini comme une fonction associant un agent à chacun des éléments d'une partie finie de $\mathcal{H}$. On préfère

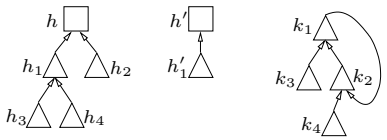


FIGURE 6.3 – Un réseau de référence générique.

cependant notre syntaxe explicitement concurrente, qui est plus intuitive, et plus facile à manipuler.

Remarquons que notre définition de bonne formation n'interdit pas les *cycles de messagers* (nous expliquons plus bas pourquoi) ; la figure 6.3 représente ainsi un réseau bien formé, qui contient deux localités réelles (h et h'), vers lesquelles pointent deux forêts de messagers (les h_i et h'_1), ainsi qu'un cycle de messagers (k_1, k_2) vers lequel pointe une autre forêt de messagers (k_3 et k_4). Selon la terminologie introduite ci-dessous, les localités h_i admettent h comme destination, et les localités k_j sont perdues. On dénote dans la suite par $\Pi_{i\in I}\ U_i$ la composition parallèle d'une famille de réseaux $(U_i)_{i\in I}$.

Définition 6.7 (Destination, localité perdue)**.** Soit U un réseau de référence bien formé. La *destination d'une localité h dans U* est la localité k s'il existe l, m, V et $(h_i)_{i\leq l}$ avec $h = h_0$, $k = h_l$ tels que :

$$U = V \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[m]\ ;$$

elle est indéfinie sinon. Une localité *perdue dans U* est une localité définie par U dont la destination est indéfinie. On note $lp(U)$ l'ensemble des localités perdues de U.

Notons que la bonne formation assure l'unicité des destinations. Intuitivement une localité est perdue lorsqu'elle fait partie d'un cycle de messagers, ou lorsqu'elle pointe vers un tel cycle. Dans la suite, nous dénotons par $m; n$ la concaténation de deux messages m et n (ce sont des multi-ensembles de messages élémentaires, on en prend simplement la somme). On peut maintenant définir les transitions des réseaux de référence ; nous les décrivons plus bas.

Définition 6.8 (Modèle de référence)**.** Le *modèle de référence* est $\langle \mathcal{U}_\mathrm{r}, \rightarrow_\mathrm{r} \rangle$, où $\mathcal{U}_\mathrm{r}$ est l'ensemble des réseaux de référence bien formés, et $\rightarrow_\mathrm{r}$ est définie par les règles d'inférence de la figure 6.4.

Le lemme suivant valide la définition, et nous permet de ne considérer que des réseaux bien formés dans la suite.

Lemme 6.9. *La bonne formation* est préservée par $\rightarrow_r$ *, i.e., si* $U \xrightarrow{\mu}_r U'$ *et si U est bien formé, alors U' est bien formé.* $\langle \mathcal{U}_r, \rightarrow_r \rangle$ *est donc un* $\mathcal{L}_r$*-TS.*

$$[\textsc{Rcv}]\ \frac{}{h[m;n] \mid U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} h[n] \mid U} \qquad [\textsc{Snd}]\ \frac{}{h[n] \mid U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} h[m;n] \mid U}$$

$$[\textsc{Kil}]\ \frac{}{U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} U}\ h \in lp(U) \qquad [\textsc{Fwd}]\ \frac{h \triangleright k \mid U \xrightarrow{k\langle m\rangle}_{\mathrm{r}} U'}{h \triangleright k \mid U \xrightarrow{h\langle m\rangle}_{\mathrm{r}} U'}$$

$$[\textsc{New}]\ \frac{}{U \xrightarrow{\nu h}_{\mathrm{r}} h[\emptyset] \mid U}\ h \notin l(U) \qquad [\textsc{Mig}]\ \frac{h \triangleright k \mid U \xrightarrow{k\langle m\rangle}_{\mathrm{r}} U'}{h[m] \mid U \xrightarrow{h \triangleright k}_{\mathrm{r}} U'}$$

FIGURE 6.4 – Transitions des réseaux de référence.

On décrit maintenant les transitions des réseaux de référence (figure 6.4) : lorsqu'une localité réelle contient un message m, il peut être reçu par le processus qu'elle héberge par la règle [Rcv]. Quand un message est envoyé vers une localité réelle il est ajouté aux messages de cette dernière (règle [Snd]) ; par la règle [Kil], tout message envoyé vers une localité perdue disparaît simplement (il est pris dans un cycle de messagers). Enfin lorsqu'un message atteint une localité hébergeant un messager $h \triangleright k$, il est redirigé vers k par la règle [Fwd]. La règle [New] permet d'ajouter une localité réelle, pourvu qu'elle ne soit pas déjà définie. Finalement, une localité réelle $h[m]$ peut migrer vers une localité définie (k) par la règle [Mig] ; dans ce cas, le message m est tout d'abord transmis à k (m contient les messages élémentaires qui ont atteint h, mais que le processus hébergé en h n'a pas encore consommés), puis la localité réelle est remplacée par un messager $h \triangleright k$.

Remarquons que du fait de la bonne formation, les transitions sont déterministes : si $U \xrightarrow{\mu}_{\mathrm{r}} U_1$ et $U \xrightarrow{\mu}_{\mathrm{r}} U_2$, alors $U_1 = U_2$, et que ce modèle de référence est plutôt « haut-niveau » : il n'utilise pas de transitions silencieuses, et la transmission des messages par les messagers est réalisée de manière atomique, par la règle [Fwd]. Ce dernier point, ainsi que la condition de la règle [Kil], qui nécessite de détecter les cycles de messagers, rend une implantation directe et distribuée de ce modèle difficilement réalisable.

Remarquons aussi qu'un réseau de référence n'accepte d'émettre un message que vers les localités définies (mais potentiellement perdues), et qu'il n'accepte de même que les migrations vers des localités définies. Bien que cela complique légèrement la présentation de la spécification, cela nous permettra une présentation plus simple des deux implantations, et cette contrainte est réaliste en pratique : il est aisé de s'assurer que les processus n'inventent pas des localités. Par contre, rien n'empêche une migration de former un cycle de messagers (on a par exemple $h[m] \xrightarrow{h \triangleright h}_{\mathrm{r}} h \triangleright h$). De telles situations correspondent a priori à des erreurs, mais en les acceptant du côté du

modèle, on laisse le choix au concepteur des processus distribués de décider lui-même de ce qui est une erreur, selon la possibilité qu'il a de prouver que de telles configurations ne sont jamais atteintes : le modèle fixe une sémantique même dans ces situations problématiques (vérifier que les migrations que demandées par un ensemble de processus ne génèrent pas de cycle n'est pas forcément facile).

Le lemme suivant donne une intuition plus précise du comportement d'un réseau de référence lors de l'émission d'un message :

Lemme 6.10. *Soient U un réseau de référence, h, k deux localités, et m un message.*

1. *Si $U = h \triangleright k \mid V$, alors soit les localités h et k sont perdues, soit elles ont la même destination.*
2. *La destination de h dans U est k et $U = k[n] \mid V$ si et seulement si $U \xrightarrow{h\langle m\rangle}_r k[m; n] \mid V$.*
3. *La localité h est perdue dans U si et seulement si $U \xrightarrow{h\langle m\rangle}_r U$.*

Remarque 6.11 (Expressivité du modèle). Les localités expriment seulement la distribution *logique* des processus. La condition de bonne formation n'interdit donc pas de placer plusieurs localités sur la même unité *physique*. D'autre part, à la différence de [SV01, HPS05], les processus ne doivent pas être distribués (même logiquement) selon une hiérarchie arborescente. Il n'y a en particulier aucune contrainte sur la topologie des liens de communication (sur la figure 6.3, seuls les messagers forment des arbres : les localités réelles ne sont pas ordonnées).

Remarquons aussi que la migration est *subjective* : les processus décident eux-même de migrer. Des mécanismes de migration *objective* (comme la « passivation », disponible dans le calcul des Kells [SS04]) peuvent être simulés, en utilisant des messages pour déclencher les migrations.

6.2 Définition et correction d'un modèle d'exécution simple

On définit maintenant un second modèle, plus réaliste du point de vue d'une implantation distribuée. On valide cette implantation en prouvant que ses états (les réseaux *simples*) sont faiblement bisimilaires aux réseaux de référence.

6.2.1 Réseaux simples

On considère les *réseaux simples*, dont la syntaxe est donnée sur la figure 6.2 ; elle étend celle des réseaux de référence en ajoutant des *messages*

$$[\textsc{Rcv}_\text{s}]\ \frac{}{h[m;n] \mid U \xrightarrow{h(m)}_\text{s} h[n] \mid U} \qquad [\textsc{Snd}_\text{s}]\ \frac{}{U \xrightarrow{h\langle m\rangle}_\text{s} h\langle m\rangle \mid U}\ h \in l(U)$$

$$[\textsc{Fwd}_\text{s}]\ \frac{}{h\langle m\rangle \mid h \triangleright k \mid U \xrightarrow{\tau}_\text{s} h \triangleright k \mid k\langle m\rangle \mid U}$$

$$[\textsc{Dst}_\text{s}]\ \frac{}{h\langle m\rangle \mid h[n] \mid U \xrightarrow{\tau}_\text{s} h[m;n] \mid U} \qquad [\textsc{New}_\text{s}]\ \frac{}{U \xrightarrow{\nu h}_\text{s} h[\emptyset] \mid U}\ h \notin l(U)$$

$$[\textsc{Mig}_\text{s}]\ \frac{}{h[m] \mid U \xrightarrow{h \triangleright k}_\text{s} h \triangleright k \mid k\langle m\rangle \mid U'}\ k \in h, l(U)$$

FIGURE 6.5 – Transitions des réseaux simples.

en attente. Ces derniers seront utilisés pour représenter les états intermédiaires correspondant à l'acheminent des messages par les messagers.

On étend aux réseaux simples les notions de destination et de localité définie ou perdue (définitions 6.6 et 6.7) en ignorant les messages en attente. La bonne formation est étendue ci-dessous ; par rapport à la définition 6.6, on demande simplement que tous les messages en attente le soient sur des localités définies.

Définition 6.12 (Bonne formation des réseaux simples, message perdu)**.** Un réseau simple U est *bien formé* si toute localité y est définie au plus une fois, et si pour tout messager $h \triangleright k$ ou message en attente $k\langle m\rangle$ apparaissant dans U, la localité k est définie par U. Un message en attente $h\langle m\rangle$ dans un réseau simple U est dit *perdu* lorsque h est perdue dans U.

On définit ensuite les transitions des réseaux simples :

Définition 6.13 (Modèle simple)**.** Le *modèle simple* est $\langle \mathcal{U}_\text{s}, \rightarrow_\text{s}\rangle$, où $\mathcal{U}_\text{s}$ est l'ensemble des réseaux simples bien formés, et où la relation $\rightarrow_\text{s}$ est définie par les règles de la figure 6.5.

Lemme 6.14. *La bonne formation est préservée par $\rightarrow_s$; $\langle \mathcal{U}_s, \rightarrow_s\rangle$ est un $\mathcal{L}_r$-TS.*

Les règles [$\textsc{Rcv}_\text{s}$] et [$\textsc{New}_\text{s}$] restent inchangées, la règle [$\textsc{Snd}_\text{s}$] est simplifiée : on ajoute simplement le message au réseau, sous la forme d'un message en attente sur la localité h, à condition que cette localité soit définie. Les deux règles « silencieuses » se chargent de l'acheminement des messages en attente : la règle [$\textsc{Fwd}_\text{s}$] définit le comportement des messagers, ils redirigent les messages ; la règle [$\textsc{Dst}_\text{s}$] s'occupe de la réception finale d'un message, par une localité réelle. Notons que la règle [$\textsc{Mig}_\text{s}$], qui pourrait sembler différente de [$\textsc{Mig}$], reste en fait inchangée, du fait de la nouvelle définition

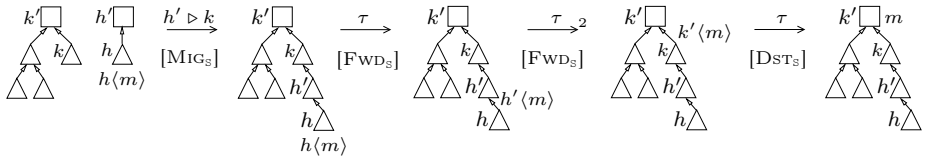


FIGURE 6.6 – Migration et acheminement des messages en attente.

de $\xrightarrow{h\langle m\rangle}_S$. Enfin, il n'y a pas de règle correspondant à [KIL] : les messages perdus restent indéfiniment dans le réseau, et tournent le long des cycles de messagers.

Une séquence de transitions est illustrée sur la figure 6.6 : la localité réelle h' migre vers k et le message m, en attente sur h, est acheminé en k', sa destination finale.

Remarquons que les règles définissant les transitions du modèle simple n'ont pas de prémisses, chacune d'elle ne fait intervenir qu'une localité, et le besoin de détecter les localités perdues a disparu avec la règle [KIL]. Ces trois propriétés rendent ce modèle adapté pour une implantation distribuée : les programmes hébergés par chacune des localités peuvent s'exécuter de façon asynchrone.

6.2.2 Correction du modèle simple

Les deux différences principales entre le modèle de référence et le modèle simple sont l'introduction de transitions silencieuses pour transmettre les messages, et le fait que des messages perdus persistent dans le modèle simple, bien qu'ils soient indétectables. Nous commençons par montrer que les messages perdus peuvent être tout simplement retirés des réseaux simples ; on peut ainsi se ramener à l'étude d'un modèle dit *propre*, où les messages perdus sont retirés dès qu'une transition les introduit. Le modèle ainsi obtenu est réactif (définition 4.45), puisqu'il n y a plus de messages perdus, et nous montrons qu'il est τ-inerte (définition 3.34) : l'acheminement des messages par les transitions silencieuses préserve la classe d'équivalence du réseau. Cela nous permet de prouver la correction vis-à-vis de la spécification en considérant un candidat *normalisé*, où tous les messages qui apparaissent lors du jeu de bisimulation sont systématiquement acheminés vers leur destination.

La simplicité du modèle nous permet d'utiliser seulement l'expansion ($\succsim$) dans cette section : le modèle simple « expanse » le modèle propre, qui expanse à son tour le modèle de référence. Le seul endroit où cela est réellement utile est dans la preuve de τ-inertie : cela nous permet d'utiliser la technique « expansion modulo transitivité » (théorème 3.53), et de garder ainsi une preuve très locale.

Du modèle simple au modèle propre

Nous commençons par prouver que l'on peut retirer les messages perdus :

Lemme 6.15. *Soit h une localité perdue dans un réseau simple U. Pour tout message m on a :*

$$U \mid h\langle m\rangle \succsim U \quad .$$

Démonstration. On vérifie que la relation suivante est une expansion :

$$R \triangleq \{\langle U \mid h\langle m\rangle, U\rangle \mid U \in \mathcal{U}_{\mathrm{s}},\ h \in lp(U)\} \quad .$$

Pour la partie de droite à gauche, on a immédiatement que $U \xrightarrow{\mu}_{\mathrm{s}} U'$ entraîne $U \mid h\langle m\rangle \xrightarrow{\mu}_{\mathrm{s}} U' \mid h\langle m\rangle$, et on vérifie que h est toujours perdu dans U'.

Dans l'autre sens, le seul cas non-trivial apparaît lorsque $U = V \mid h \triangleright k$, $U \mid h\langle m\rangle \xrightarrow{\tau}_{\mathrm{s}} U \mid k\langle m\rangle$. Dans ce cas, par le lemme 6.10(1), k est perdu dans V, d'où $U \mid k\langle m\rangle \ R\ U$: le réseau de droite ne bouge pas. ■

Définition 6.16 (Réseau propre)**.** Un réseau U est *propre* s'il ne contient aucun message perdu. On note $\lfloor \mathcal{U}_{\mathrm{s}} \rfloor$ l'ensemble des réseaux simples propres, et $\lfloor U \rfloor$ le réseau propre obtenu à partir d'un réseau simple U, en en retirant tous les messages perdus.

Corollaire 6.17. *Pour tout réseau simple U, on a $U \succsim \lfloor U \rfloor$.*

L'ensemble des réseaux propres n'est cependant pas préservé par $\rightarrow_{\mathrm{s}}$: la règle [$\textsc{Snd}_{\mathrm{s}}$] permet de rajouter des messages perdus, et la règle [$\textsc{Mig}_{\mathrm{s}}$] permet de former des cycles de messagers, et de perdre ainsi des localités sur lesquelles attendaient potentiellement des messages. Pour obtenir un modèle propre, il faut donc retirer les messages perdus après chaque transition (en fait, seulement les émissions et les migrations) :

Définition 6.18 (Modèle propre)**.** Le *modèle propre* est $\langle \lfloor \mathcal{U}_{\mathrm{s}} \rfloor, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$, où $\rightarrow_{\lfloor \mathrm{s} \rfloor}$ est définie par la règle suivante :

$$\frac{U \xrightarrow{\mu}_{\mathrm{s}} U'}{U \xrightarrow{\mu}_{\lfloor \mathrm{s} \rfloor} \lfloor U' \rfloor}$$

Ce modèle n'est pas réaliste du point de vue d'une implantation distribué, puisqu'il doit détecter les localités perdues, c'est-à-dire les cycles de messagers. Mais ce n'est pas gênant : nous n'utiliserons ce modèle que pour raisonner sur le modèle simple. Comme le montre la proposition suivante ces deux modèles sont en effet équivalents (la notation $\langle U, \rightarrow_{\mathrm{s}} \rangle$ permet d'indiquer dans quel $\mathcal{L}_r$-TS est considéré le réseau : cette proposition indique que le réseau simple U, considéré comme un élément du modèle simple, expanse sa forme nettoyée, $\lfloor U \rfloor$, considérée comme un élément du modèle propre ; dans le corollaire précédent les deux réseaux étaient implicitement considérés comme faisant partie du modèle simple, bien que le second soit propre).

Proposition 6.19. *Pour tout réseau simple U, on a :*

$$\langle U, \rightarrow_s \rangle \gtrsim \langle \lfloor U \rfloor, \rightarrow_{\lfloor s \rfloor} \rangle \ .$$

Démonstration. Par le corollaire 6.17, il suffit par transitivité de montrer que $\langle U, \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$, pour tout réseau propre U. On montre pour cela que la relation ci-dessous est une expansion modulo expansion (i.e., que $R \rightsquigarrow_{\mathbf{e} \wedge \overline{\mathbf{w}}} \gtrsim \cdot R$, un cas particulier du théorème 3.53).

$$R \triangleq \left\{ \langle \langle U, \rightarrow_{\mathrm{s}} \rangle, \ \langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \rangle \mid U \in \lfloor \mathcal{U}_{\mathrm{s}} \rfloor \right\} \ .$$

- Si $\langle U, \rightarrow_{\mathrm{s}} \rangle \xrightarrow{\mu} \langle U', \rightarrow_{\mathrm{s}} \rangle$, alors $\langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \xrightarrow{\mu} \langle \lfloor U' \rfloor, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$ et on vérifie que $\langle U', \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle \lfloor U' \rfloor, \rightarrow_{\mathrm{s}} \rangle \ R \ \langle \lfloor U' \rfloor, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$.
- Si $\langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \xrightarrow{\mu} \langle U', \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$, alors $\langle U, \rightarrow_{\mathrm{s}} \rangle \xrightarrow{\mu} \langle V, \rightarrow_{\mathrm{s}} \rangle$ avec $\lfloor V \rfloor = U'$, et on vérifie que $\langle V, \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle U', \rightarrow_{\mathrm{s}} \rangle \ R \ \langle U', \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle$. ■

Notons que l'on utilise dans la preuve la technique « expansion modulo expansion », mais que l'on pourrait en fait utiliser « bisimulation faible modulo $\approx$ lors des offres visibles », si le retrait des messages perdus était seulement contenu dans la bisimilarité faible. On n'utilise en effet la technique que lors des offres visibles correspondant à une migration ou une émission.

Réactivité et τ-inertie du modèle propre

On a maintenant un modèle propre, où tout message en attente admet une destination ; nous allons montrer qu'il est réactif et τ-inerte. On montre dans un premier temps que les transitions silencieuses sont convergentes :

Lemme 6.20. *La relation $\xrightarrow{\tau}_{\lfloor s \rfloor}$ est confluente et normalise fortement.*

Démonstration. On définit le poids d'un message en attente $h\langle m \rangle$ dans un réseau propre U, comme l'entier naturel $l + 1$, où l est donné par la définition 6.7 (comme U est propre, h admet une destination dans U). La taille de U est ensuite définie comme la somme des poids de ses messages en attente. Ce poids diminue strictement lors de chaque transition silencieuse, $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$.

De plus, grâce à la bonne formation, $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$ n'admet aucune paire critique, d'où la confluence. ■

Notons que la relation $\xrightarrow{\tau}_{\mathrm{s}}$ ne termine pas, à cause de la présence potentielle de messages en attente dans des cycles de messagers. On dénote par $U_{\downarrow}$ la forme normale d'un réseau propre U par rapport à $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$. Remarquons que pour tout réseau simple U, $U = \lfloor U \rfloor_{\downarrow}$ si et seulement si U est un réseau de référence.

On prouve en réalité un résultat plus fort que la τ-inertie : les transitions silencieuses sont contenues dans l'expansion. Comme expliqué plus haut, cela nous permet d'utiliser la technique expansion modulo transitivité, et de donner une preuve bien plus simple.

Lemme 6.21. *La relation $\xrightarrow{\tau}_{\lfloor s \rfloor}$ est contenue dans l'expansion.*

Démonstration. On montre que la relation $R \triangleq \xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} \cup R'$ est une expansion modulo transitivité (théorème 3.53), où

$$R' \triangleq \{\langle h\langle m\rangle \mid h\langle n\rangle \mid U,\ h\langle m;n\rangle \mid U\rangle \mid U \in \lfloor \mathcal{U}_{\mathrm{s}} \rfloor,\ h \notin lp(U)\} \quad .$$

Les deux cas où l'on a besoin de la transitivité sont les offres suivantes du réseau de gauche :

- $U = h\langle m\rangle \mid h[n] \mid U_0 \xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} h[m;n] \mid U_0 = V$ par la règle $[\mathrm{DST_s}]$, et $U \xrightarrow{h \triangleright k}_{\lfloor \mathrm{s} \rfloor} U' = h\langle m\rangle \mid h \triangleright k \mid k\langle n\rangle \mid U_0$ par la règle $[\mathrm{MIG_s}]$. On a :

$$\begin{array}{lll} V \xrightarrow{h \triangleright k}_{\lfloor \mathrm{s} \rfloor} V' = h \triangleright k \mid k\langle m;n\rangle \mid U_0 & & [\mathrm{MIG_s}] \\ U' \quad R \quad h \triangleright k \mid k\langle m\rangle \mid k\langle n\rangle \mid U_0 \quad R \quad V' \ ; & & [\mathrm{FWD_s}],(R') \end{array}$$

- $U = h\langle m\rangle \mid h\langle n\rangle \mid h \triangleright k \mid U_0 \quad R' \quad h\langle m;n\rangle \mid h \triangleright k \mid U_0 = V$ et $U \xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} U' = h\langle m\rangle \mid h \triangleright k \mid k\langle n\rangle \mid U_0$ par la règle $[\mathrm{FWD_s}]$. Dans ce cas, on a :

$$\begin{array}{lll} V \xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} V' = h \triangleright k \mid k\langle m;n\rangle \mid U_0 & & [\mathrm{FWD_s}] \\ U' \quad R \quad h \triangleright k \mid k\langle m\rangle \mid k\langle n\rangle \mid U_0 \quad R \quad V' \ . & & [\mathrm{FWD_s}],(R') \end{array}$$

■

Corollaire 6.22. *Pour tout réseau propre U on a :*

$$\langle U, \rightarrow_{\lfloor s \rfloor}\rangle \gtrsim \langle U_{\downarrow}, \rightarrow_{\lfloor s \rfloor}\rangle \ .$$

Un phénomène qui apparaît dans la preuve du lemme précédent nous permet de justifier l'utilité de la technique « modulo transitivité » par rapport à la technique « modulo expansion », plus standard. Lorsque l'on commence la preuve avec la relation $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$ (qui est un candidat plus naturel), on se retrouve bloqué dans le premier cas détaillé dans la preuve, et l'on souhaite raisonner modulo les couples de R'. On pourrait utiliser pour cela la technique « modulo expansion », si l'on pouvait prouver au préalable que cette relation est contenue dans l'expansion. Or, lorsque l'on essaie de montrer que R' est une expansion, on se retrouve à nouveau bloqué, dans le deuxième cas détaillé dans la preuve, où l'on souhaite raisonner modulo $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor} \ldots$ Pour sortir de ce cercle vicieux, la solution consiste à prouver conjointement que $\xrightarrow{\tau}_{\lfloor \mathrm{s} \rfloor}$ et R' sont contenues dans l'expansion, et cela nécessite la technique « modulo transitivité ».

Notons que l'on pourrait prouver le lemme précédent sans technique modulo, mais en utilisant un candidat très global : l'ensemble des couples de réseaux qui ont les mêmes arbres de messagers, et dont les multi-ensembles de messages « destinés » à chaque localité réelle coïncident. La définition formelle de ce candidat, et la preuve qu'il s'agit bien d'une bisimulation faible seraient vraiment laborieuses (ce serait pire encore pour définir une relation d'expansion).

Du modèle propre au modèle de référence

On peut finalement prouver la correction du modèle propre vis-à-vis du modèle de référence. On commence par établir la correction de l'algorithme d'acheminement des messages : étant donné un réseau de référence U, si l'acheminement d'un message par le réseau de référence mène à un réseau V, alors le réseau propre peut faire de même, en utilisant quelques transitions silencieuses. Réciproquement, si un message est ajouté à U par une transition propre menant à V, la normalisation de V aboutit exactement au réseau de référence obtenu par le $\mathcal{L}_r$-TS de référence.

Lemme 6.23. *Soit U un réseau de référence.*

1. *Si $U \xrightarrow{h\langle m\rangle}_r V$, alors $U \xrightarrow{h\langle m\rangle}_{\lfloor s\rfloor} \cdot \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor s\rfloor} V$.*
2. *Si $U \xrightarrow{h\langle m\rangle}_{\lfloor s\rfloor} V$, alors $U \xrightarrow{h\langle m\rangle}_r V_{\downarrow}$.*

Démonstration. 1. Si h est perdu dans U, alors par le lemme 6.10(3) $U = V$, et on a $U \xrightarrow{h\langle m\rangle}_{\lfloor s\rfloor} \lfloor h\langle m\rangle \mid U\rfloor = \lfloor U\rfloor = U$. Sinon, h admet une destination dans U, d'où

$$\begin{aligned} U &= U' \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[n] && (\text{où } h = h_0)\\ V &= U' \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[m;n]\ , && (\text{lemme 6.10(2)}) \end{aligned}$$

on vérifie alors que $U \xrightarrow{h\langle m\rangle}_{\lfloor s\rfloor} h\langle m\rangle \mid U \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor s\rfloor} V$, par récurrence sur l.

2. Si h est perdu dans U, alors $V = \lfloor h\langle m\rangle \mid U\rfloor = U$, et $U \xrightarrow{h\langle m\rangle}_r U$ par le lemme 6.10(3). Sinon,

$$\begin{aligned} &U = U' \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[n] && (\text{où } h = h_0)\\ &U \xrightarrow{h\langle m\rangle}_{\lfloor s\rfloor} V = h\langle m\rangle \mid U \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor s\rfloor} U' \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[m;n] = V'. \end{aligned}$$

où U', et donc V', sont nécessairement des réseaux de référence. Par le lemme 6.10(2), $U \xrightarrow{h\langle m\rangle}_r V'$, et puisque V' est en forme normale, $V_{\downarrow} = V'$. ■

Grâce à la technique expansion modulo expansion, on peut ensuite travailler avec un candidat très simple : l'identité syntaxique entre les réseaux de référence, équipée des transitions simples et propres d'un côté, et des transitions de référence de l'autre. Lorsqu'une offre du côté simple « dé-normalise » le réseau en introduisant un message en attente, on renormalise modulo expansion en utilisant le corollaire 6.22 (on applique en fait un cas relativement dégénéré de la technique décrite page 156).

Lemme 6.24. *Pour tout réseau de référence U, on a :*

$$\langle U, \rightarrow_{\lfloor s\rfloor}\rangle \gtrsim \langle U, \rightarrow_r\rangle$$

Démonstration. On montre que la relation suivante est une expansion modulo expansion :

$$R \triangleq \left\{ \langle \langle U, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle,\ \langle U, \rightarrow_{\mathrm{r}} \rangle \rangle \mid U \in \mathcal{U}_{\mathrm{r}} \right\} \ .$$

On considère les différentes offres selon une action μ :

- μ ne peut pas être τ, puisque les réseaux de référence sont en forme normale ;
- si $\mu = \nu h$ ou $\mu = h(m)$, on a $U \xrightarrow{\mu}_{\lfloor \mathrm{s} \rfloor} V$ si et seulement si $U \xrightarrow{\mu}_{\mathrm{r}} V$;
- si $\mu = h\langle m \rangle$, on applique le lemme 6.23 :
  - si $U \xrightarrow{h\langle m \rangle}_{\lfloor \mathrm{s} \rfloor} V$, alors on a $U \xrightarrow{h\langle m \rangle}_{\mathrm{r}} V_{\downarrow}$, puis, par le corollaire 6.22, on obtient $\langle V, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \gtrsim \langle V_{\downarrow}, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \ R \ \langle V_{\downarrow}, \rightarrow_{\mathrm{r}} \rangle$ ($V_{\downarrow}$ est un réseau de référence) ;
  - si $U \xrightarrow{h\langle m \rangle}_{\mathrm{r}} V$, alors on a $U \xRightarrow{\widehat{h\langle m \rangle}}_{\lfloor \mathrm{s} \rfloor} V$ et $\langle V, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \ R \ \langle V, \rightarrow_{\mathrm{r}} \rangle$.
- Le cas $\mu = h \triangleright k$ s'appuie sur le précédent. ■

Le résultat final de correction de l'implantation simple en découle :

Théorème 6.25. *Pour tout réseau simple U, on a :*

$$\langle U, \rightarrow_s \rangle \gtrsim \langle \lfloor U \rfloor_{\downarrow}, \rightarrow_r \rangle \ .$$

Démonstration. Par la proposition 6.19, le corollaire 6.22 et le lemme 6.24,

$$\langle U, \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle \lfloor U \rfloor, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \gtrsim \langle \lfloor U \rfloor_{\downarrow}, \rightarrow_{\lfloor \mathrm{s} \rfloor} \rangle \gtrsim \langle \lfloor U \rfloor_{\downarrow}, \rightarrow_{\mathrm{r}} \rangle$$

(rappelons que $\lfloor U \rfloor_{\downarrow}$ est un réseau de référence). ■

En particulier, si U est un réseau de référence, alors $\langle U, \rightarrow_{\mathrm{s}} \rangle \gtrsim \langle U, \rightarrow_{\mathrm{r}} \rangle$.

6.3 Définition et correction d'une optimisation

Les chaînes de messagers qui sont générées lors de l'évolution d'un réseau rendent l'implantation précédente inefficace. Par exemple, le message m de la figure 6.6 doit passer par trois localités avant d'atteindre sa destination finale, il en est de même pour tout message qui sera potentiellement émis vers h par la suite. Dans cette section, on définit un modèle optimisé qui contracte ces chaînes de messagers, puis on prouve la correction de cette optimisation, en montrant que les réseaux simples sont faiblement bisimilaires aux réseaux optimisés.

6.3.1 Réseaux optimisés

Nous noterons dans la suite $\widetilde{h}, \widetilde{k} \ldots$ les listes de localités. Comme l'ensemble vide et le multi-ensemble vide, la liste vide sera noté $\emptyset$, et on dénotera par $h;\widetilde{k}$ (resp. $\widetilde{h};\widetilde{k}$) l'addition d'une localité h (resp. d'une liste de localités $\widetilde{h}$) à une liste $\widetilde{k}$. La syntaxe des *réseaux optimisés* est donnée figure 6.2 ; elle étend celle des réseaux simples :

- en annotant les messages en attente avec une liste de localités : $h\langle m\rangle_{\widetilde{k}}$;
- en introduisant des *messagers bloqués* : $h\not\triangleright$;
- et en ajoutant une seconde sorte de messages, les *messages de relocalisation* : $h\langle\triangleright k\rangle$.

Intuitivement, la liste qui décore un message en attente contient l'ensemble des messagers par lesquels il est passé. Les messages émis par les processus sous-jacents auront donc une liste vide. Les messages de relocalisation sont uniquement destinés aux messagers bloqués ; leur effet consistera à débloquer et rediriger ces messagers vers une localité plus proche de la localité vers laquelle ils pointaient auparavant.

On commence par étendre la propriété de bonne formation aux réseaux optimisés.

Définition 6.26 (Localité définie, bonne formation des réseaux optimisés)**.** Une localité $h \in \mathcal{H}$ est *définie* par un réseau optimisé U lorsqu'il existe m, V, k tels que $U = h[m] \mid V$, $U = h \triangleright k \mid V$ ou $U = h\not\triangleright \mid V$. On note $l(U)$ l'ensemble des localités définies par U.

Un réseau optimisé U est *bien formé* si :

1. toute localité $h \in \mathcal{H}$ apparaît au plus une fois dans U en tant que source d'une localité réelle ($h[m]$), d'un messager ($h \triangleright k$), ou d'un messager bloqué ($h\not\triangleright$) ; et si pour tout messager $h \triangleright k$, message en attente $k\langle m\rangle$ ou message de relocalisation $h\langle\triangleright k\rangle$ que U contient, la localité k est définie.
2. Pour tout messager bloqué $h\not\triangleright$ de U, h apparaît exactement une fois dans la liste de localités décorant un (unique) message en attente ($k\langle m\rangle_{\widetilde{h}}$, avec $h \in \widetilde{h}$), ou comme la cible d'un (unique) message de relocalisation ($h\langle\triangleright k\rangle$).
3. Toute localité enregistrée dans la liste d'un message en attente ou apparaissant en tant que cible d'un message de relocalisation héberge un messager bloqué.

On peut alors définir le modèle :

Définition 6.27 (Modèle optimisé)**.** Le *modèle optimisé* est $\langle\mathcal{O}, \rightarrow_{\mathrm{o}}\rangle$, où $\mathcal{O}$ est l'ensemble des réseaux optimisés bien formés, et $\rightarrow_{\mathrm{o}}$ est définie sur la figure 6.7.

$$[\textsc{Rcv}_{\text{o}}]\ \frac{}{h[m;n] \mid U \xrightarrow{h(m)}_{\text{o}} h[n] \mid U} \qquad [\textsc{Snd}_{\text{o}}]\ \frac{}{U \xrightarrow{h\langle m\rangle}_{\text{o}} h\langle m\rangle_{\emptyset} \mid U}\ h \in l(U)$$

$$[\textsc{New}_{\text{o}}]\ \frac{}{U \xrightarrow{\nu h}_{\text{o}} h[\emptyset] \mid U}\ h \notin l(U)$$

$$[\textsc{Mig}_{\text{o}}]\ \frac{}{h[m] \mid U \xrightarrow{h \triangleright k}_{\text{o}} h \triangleright k \mid k\langle m\rangle \mid U'}\ k \in h, l(U)$$

$$[\textsc{Fwd}_{\text{o}}]\ \frac{}{h\langle m\rangle_{\widetilde{k}} \mid h \triangleright k \mid U \xrightarrow{\tau}_{\text{o}} h\not\triangleright \mid k\langle m\rangle_{h;\widetilde{k}} \mid U}$$

$$[\textsc{Dst}_{\text{o}}]\ \frac{}{h\langle m\rangle_{\widetilde{k}} \mid h[n] \mid U \xrightarrow{\tau}_{\text{o}} h[m;n] \mid \widetilde{k}\langle \triangleright h\rangle \mid U}$$

$$[\textsc{Upd}_{\text{o}}]\ \frac{}{h\langle \triangleright k\rangle \mid h\not\triangleright \mid U \xrightarrow{\tau}_{\text{o}} h \triangleright k \mid U}$$

FIGURE 6.7 – Transitions des réseaux optimisés.

Proposition 6.28. *Tout réseau simple bien formé (au sens de la définition 6.6) est bien formé au sens de la définition 6.26. La bonne formation d'un réseau optimisé est préservée par $\rightarrow_o$; $\langle \mathcal{O}, \rightarrow_o \rangle$ est un $\mathcal{L}_r$-TS.*

Démonstration. Essentiellement bureaucratique ■

Nous décrivons les règles ci-dessous ; par rapport au modèle simple, les règles « visibles » [Rcv], [Snd], [Mig] et [New] restent inchangées, les deux règles « silencieuses » [Fwd] et [Dst] sont modifiées, et la règle [$\textsc{Upd}_{\text{o}}$] est nouvelle. Dans la règle [$\textsc{Dst}_{\text{o}}$], on note $\widetilde{h}\langle \triangleright k\rangle$ la composition parallèle $\Pi_{h\in\widetilde{h}}\ h\langle \triangleright k\rangle$. Dans la suite, nous noterons respectivement $\widetilde{h} \triangleright k$ et $\widetilde{h}\not\triangleright$ les réseaux $\Pi_{h\in\widetilde{h}}\ h \triangleright k$ et $\Pi_{h\in\widetilde{h}}\ h\not\triangleright$.

Lorsqu'un messager transmet un message (règle [$\textsc{Fwd}_{\text{o}}$]), il enregistre sa localité et bascule dans l'état *bloqué*, de telle sorte qu'il ne transmet temporairement plus les messages potentiels. Lors de la réception finale du message par une localité réelle (règle [$\textsc{Dst}_{\text{o}}$]), un message de relocalisation est diffusé vers chacun des messagers enregistrés dans la liste du message. Les messagers se trouvant à ces localités, qui sont bloqués par bonne formation, se débloqueront lors de la réception de ce message, en mettant à jour leur destination (règle [$\textsc{Upd}_{\text{o}}$]). Ce comportement est illustré sur la figure 6.8, où les triangles grisés représentent les messagers bloqués. Notons que le mécanisme de blocage des messagers est nécessaire : sans cela, il faudrait mettre en place un système de dates de validité (« timestamps »), afin qu'un messager puisse choisir intelligemment entre deux messages de relocalisation différents : nous sommes dans un système asynchrone, et l'ordre de réception des messages

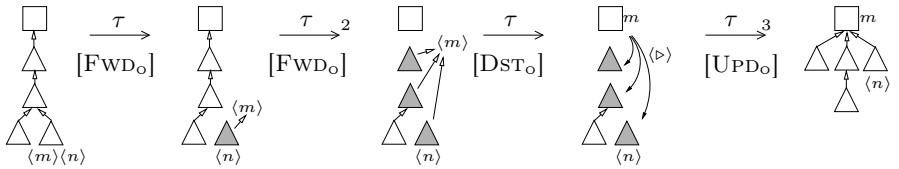


FIGURE 6.8 – Comportement des messagers optimisés.

ne correspond pas nécessairement à l'ordre d'émission.

Remarque 6.29 (D'autres optimisations sont possibles). On pourrait aller plus loin en termes d'optimisation : par exemple, les messages en attente derrière un messager bloqué pourraient être rassemblés en un unique message, qui serait ensuite envoyé en une seule fois, lorsque le messager se débloque :

$$[\text{PCK}_\text{o}]\ \frac{}{h\langle m\rangle_{\widetilde{h}} \mid h\langle n\rangle_{\widetilde{k}} \mid h\not\triangleright \mid U \xrightarrow{\tau}_\text{o} h\langle m;n\rangle_{\widetilde{h};\widetilde{k}} \mid h\not\triangleright \mid U}$$

On pourrait aussi décomposer la règle $[\text{DST}_\text{o}]$ afin de se rapprocher encore plus d'une implantation réelle, en rendant la diffusion du message de relocalisation progressive :

$$[\text{DST}_\text{o}]\ \frac{}{h\langle m\rangle_{k;\widetilde{k}} \mid h[n] \mid U \xrightarrow{\tau}_\text{o} h\langle m\rangle_{\widetilde{k}} \mid h[n] \mid k\langle\triangleright h\rangle \mid U}$$

$$[\text{DST'}_\text{o}]\ \frac{}{h\langle m\rangle_{\emptyset} \mid h[n] \mid U \xrightarrow{\tau}_\text{o} h[m;n] \mid U}$$

Néanmoins, nous nous intéressons ici surtout à la méthodologie et aux techniques de preuve utilisables pour la définition de tels systèmes distribués, plutôt qu'à la recherche de la meilleure optimisation possible. Nous préférons donc en rester à la première optimisation, afin que notre discours ne se perde pas au milieu de détails trop techniques.

6.3.2 Correction de l'optimisation

Bien qu'elle soit plus délicate, la preuve de correction de l'optimisation suit les grandes lignes de celle du modèle simple : on montre que les messages perdus peuvent être retirés, ce qui permet de définir un modèle propre et optimisé, irréaliste d'un point de vue distribué, mais bien plus pratique pour raisonner. On montre ensuite que ce modèle est réactif et τ-inerte, ce qui permet de donner une preuve de bisimilarité faible entre les modèles simple et optimisé, en utilisant un candidat normalisé.

Cycles de messagers, passage à un modèle propre

Il nous faut tout d'abord reprendre la notion de destination, afin de prendre en compte les messagers bloqués :

Définition 6.30 (Destinations dans un réseau optimisé)**.** Soit U un réseau optimisé (bien formé). La *destination d'une localité h dans U* est k s'il existe l, m, V et $(h_i)_{i\leq l}$ avec $h = h_0$, $k = h_l$ tels que :

$$U = \Pi_{i<n}\ F_i \mid h_l[m] \mid V \ ,$$

où pour tout $i < l$, F_i est :
- soit un messager, $h_i \triangleright h_{i+1}$;
- soit un messager bloqué, avec un message de relocalisation, $h_i \not\triangleright \mid h_i\langle \triangleright h_{i+1}\rangle$;
- soit un messager bloqué, dont la localité est enregistrée dans un message bloquant d'autres messagers : $h_i \not\triangleright \mid \widetilde{k} \not\triangleright \mid h_{i+1}\langle n\rangle_{h_i;\widetilde{k}}$.

Les notions de localité perdue et de réseau propre sont naturellement étendues aux réseaux optimisés en utilisant cette nouvelle notion de destination. On a les propriétés suivantes :

Lemme 6.31. *1. Les destinations sont uniques dans tout réseau optimisé bien formé.*

2. Dans chacun des réseaux suivants, soit les localités h et k sont perdues, soit elles ont la même destination :

$$(a)\ h \triangleright k \mid U \ , \qquad (b)\ h \not\triangleright \mid h\langle \triangleright k\rangle \mid U \ , \qquad (c)\ h\langle m\rangle_{k,\widetilde{k}} \mid U \ .$$

Du fait de l'optimisation, on ne peut plus retirer les messages perdus directement : pour conserver un réseau bien formé, il faut tenir compte des messagers que ces messages peuvent potentiellement bloquer. On introduit pour cela la relation suivante :

Définition 6.32 (Relation de nettoyage)**.** On appelle *relation de nettoyage* la relation suivante, sur $\mathcal{O}$:

$$\mathcal{E} \triangleq \left\{ \langle U \mid \widetilde{h} \not\triangleright \mid h\langle m\rangle_{\widetilde{h}}\ ,\ U \mid \widetilde{h} \triangleright h\rangle \ \middle|\ h \in lp(U \mid \widetilde{h} \not\triangleright \mid h\langle m\rangle_{\widetilde{h}}) \right\} \ .$$

Notons que V est bien formé dès que $U\ \mathcal{E}\ V$ et U est bien formé, et que $\mathcal{E}$ préserve les destinations et les localités perdues. On démontre que cette relation est contenue dans l'expansion en utilisant la technique « bi-expansion modulo transitivité » (théorème 3.54) :

Lemme 6.33. *La relation de nettoyage est contenue dans l'expansion.*

Démonstration. Soit $\mathcal{E}'$ la fermeture symétrique de la relation suivante :

$$\left\{ \langle U,\ U_0 \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h\rangle \ \middle|\ U = U_0 \mid \widetilde{h} \not\triangleright \mid h\langle m\rangle_{\widetilde{h}} \text{ et } h, (k_h)_{h\in\widetilde{h}} \in lp(U) \right\}$$

$\mathcal{E}$ est clairement contenue dans $\mathcal{E}'$, et $\mathcal{E}'$ préserve la bonne formation, les destinations, et les localités perdues. On montre que $\mathcal{E}'$ est une candidate pour

le théorème 3.54. Comme $\mathcal{E}'$ est symétrique, il suffit de démontrer que c'est une « pré-expansion modulo transitivité ». La condition de bonne formation nous permet de rejeter les cas pathologiques, les cas intéressants qui restent sont les suivants :

- $U = U_0 \mid \widetilde{h}\not\triangleright \mid h\langle m\rangle_{\widetilde{h}} \mid h \triangleright k \quad \mathcal{E}' \quad V = U_0 \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h \mid h \triangleright k$ et $U \xrightarrow{\tau}_{\mathrm{o}} U' = U_0 \mid (h;\widetilde{h})\not\triangleright \mid k\langle m\rangle_{h;\widetilde{h}}$ par la règle [FWD$_{\mathrm{o}}$]. On vérifie simplement que $U'\ \mathcal{E}'\ V$: le réseau de droite ne bouge pas.
- $U = U_0 \mid \widetilde{k}\not\triangleright \mid k\langle n\rangle_{\widetilde{k}} \mid (k;\widetilde{h})\not\triangleright \mid h\langle m\rangle_{k;\widetilde{h}}$
  $\mathcal{E}' \quad V = U_0 \mid \widetilde{k}\not\triangleright \mid k\langle n\rangle_{\widetilde{k}} \mid k \triangleright k' \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h$
  et $V \xrightarrow{\tau}_{\mathrm{o}} V' = U_0 \mid (k;\widetilde{k})\not\triangleright \mid k'\langle n\rangle_{k;\widetilde{k}} \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h$ par la règle [FWD$_{\mathrm{o}}$]. Dans ce cas, on a $V\ \mathcal{E}'^2\ V'$:

  $$V \quad \mathcal{E} \quad U_0 \mid (k;\widetilde{k}) \triangleright k' \mid \Pi_{h\in\widetilde{h}} h \triangleright k_h \quad \mathcal{E}' \quad V' \ .$$

  Le réseau de gauche (U) ne bouge pas, et on applique $\mathcal{E}'$ trois fois, afin de mettre U et V' en relation : $U\ \mathcal{E}'\ V\ \mathcal{E}'^2\ V'$. ■

Lemme 6.34. *La relation $\mathcal{E}$ est confluente et normalise fortement.*

On note $\lfloor\mathcal{O}\rfloor$ l'ensemble des réseaux optimisés propres, et $\lfloor U\rfloor$ la forme normale d'un réseau optimisé U vis-à-vis de $\mathcal{E}$ (sur les réseaux simples, cette notion coïncide avec celle définie dans la section 6.2.2) ; $\lfloor U\rfloor$ est un réseau propre, et l'on a :

Corollaire 6.35. *Pour tout réseau optimisé U, on a :*

$$\langle U, \to_o\rangle \gtrsim \langle \lfloor U\rfloor, \to_o\rangle \ .$$

Comme à la section 6.2.2, on utilise cette relation de nettoyage pour définir un modèle ne manipulant que des réseaux optimisés propres, modèle qui est équivalent au modèle optimisé :

Définition 6.36 (Modèle propre et optimisé)**.** Le *modèle propre et optimisé* est $\langle \lfloor\mathcal{O}\rfloor, \to_{\lfloor \mathrm{o}\rfloor}\rangle$, où la relation $\to_{\lfloor \mathrm{o}\rfloor}$ est définie par la règle suivante :

$$\frac{U \xrightarrow{\mu}_{\mathrm{o}} U'}{U \xrightarrow{\mu}_{\lfloor \mathrm{o}\rfloor} \lfloor U'\rfloor}$$

Proposition 6.37. *Pour tout réseau optimisé U, on a :*

$$\langle U, \to_o\rangle \gtrsim \langle \lfloor U\rfloor, \to_{\lfloor o\rfloor}\rangle \ .$$

Démonstration. Identique à la preuve de la proposition 6.19. ■

On peut maintenant travailler avec des réseaux propres et optimisés et valider l'algorithme d'acheminement des messages en attente, sans avoir à s'occuper des messages perdus.

Réactivité et τ-inertie

Comme dans le cas des réseaux simples, la relation $\xrightarrow{\tau}_{\lfloor o \rfloor}$ n'est pas une bisimulation. Il nous faut donc une technique de preuve afin de pouvoir travailler avec un candidat raisonnablement petit et local. On ne peut cependant pas utiliser les techniques s'appuyant sur l'expansion comme dans la section 6.2.2 : les transitions silencieuses optimisées ne sont pas contenues dans l'expansion.

Cela est dû aux « courses » introduites par le mécanisme de blocage des messagers ; ce phénomène est illustré sur la figure 6.8 : le message n, qui s'est fait doubler par m, doit attendre l'arrivée de ce dernier avant de pouvoir à son tour atteindre sa destination finale. La nature très contraignante de l'expansion – le processus de droite doit être aussi rapide que celui de gauche, à chaque instant – ne lui permet pas de rendre compte du fait que n est finalement plus proche de sa destination, puisque le messager qui le bloque est relocalisé juste sous la destination finale.

Plus formellement, si $\xrightarrow{\tau}_{\lfloor o \rfloor}$ était contenue dans l'expansion, on aurait

$$
\begin{aligned}
& U \triangleq h\langle n\rangle_\emptyset \mid h\langle m\rangle_\emptyset \mid h \triangleright k \mid k[\emptyset] \\
& \quad \succsim h\langle n\rangle_\emptyset \mid h\not\triangleright \mid k\langle m\rangle_{\{h\}} \mid k[\emptyset] \triangleq V \ . && [\text{FWD}_o] \\
\text{Or on a :} \quad & U \xrightarrow{\tau}_{\lfloor o \rfloor} h\langle m\rangle_\emptyset \mid h\not\triangleright \mid k\langle n\rangle_{\{h\}} \mid k[\emptyset] && [\text{FWD}_o] \\
& \xrightarrow{\tau}_{\lfloor o \rfloor} h\langle m\rangle_\emptyset \mid h\not\triangleright \mid h\langle \triangleright k\rangle \mid k[n] && [\text{DST}_o] \\
& \xrightarrow{k(n)}_{\lfloor o \rfloor} h\langle m\rangle_\emptyset \mid h\not\triangleright \mid h\langle \triangleright k\rangle \mid k[\emptyset] \ , && [\text{RCV}_o]
\end{aligned}
$$

et V ne peut pas effectuer une telle transition visible après deux transitions silencieuses au plus (à condition que le message m ne contienne pas le message n) : comme le message m est déjà passé il faut à V quatre transitions silencieuses pour débloquer le messager hébergé en h, et acheminer n.

Comme nous aurons besoin de raisonner modulo transitivité et que l'on sort du cadre de l'expansion, il nous faudra utiliser l'une des techniques du chapitre 4, reposant sur des hypothèses de terminaison. Une première idée serait d'utiliser le corollaire 4.53, mais nous allons voir que cela ne suffit pas, et qu'il faut utiliser le corollaire 4.55. Essayons toutefois d'utiliser le premier : cela va nous mener naturellement au second. Le diagramme silencieux requis par le corollaire 4.53 est satisfait : nous commençons par montrer que la relation $\xrightarrow{\tau}_{\lfloor o \rfloor}$ est localement confluente.

Comme on travaille avec des réseaux propres, tout message en attente a une destination ; on démontre tout d'abord un lemme permettant d'acheminer les messages vers leur destination. Alors que ce lemme était à peu près trivial dans le cas du modèle simple (lemme 6.23(1)), ici, comme illustré par la figure 6.9, cela nécessite de finir d'acheminer tous les messages qui bloquent des messagers entre le message considéré et sa destination. Il y a

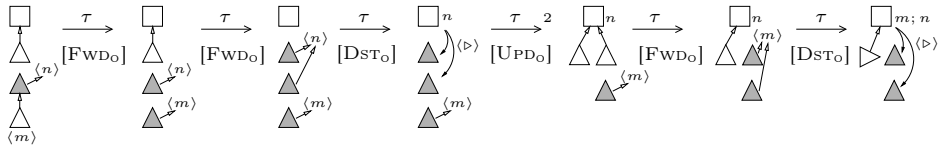


FIGURE 6.9 – Débloquer un messager pour acheminer un message donné.

un effet secondaire : les arbres de messagers sont reconfigurés par ces étapes de réductions.

Lemme 6.38. *Soit $U = V \mid h_0\langle m_0\rangle_{\widetilde{h}}$ un réseau optimisé propre. Par la définition 6.30, $U = V' \mid h_0\langle m_0\rangle_{\widetilde{h}} \mid \Pi_{i<l}F_i \mid h_n[n]$, et on a :*

$$U \overset{\tau}{\Rightarrow}_{\lfloor o\rfloor} V' \mid \widetilde{h}\langle \triangleright h_l\rangle \mid \Pi_{i<l}h_i \triangleright h_l \mid h_l[m_0; m; n] \mid \widetilde{k} \triangleright h_l$$

où n et $\widetilde{k}$ sont les messages et localités collectés dans $\Pi_{i<l}F_i$.

Démonstration. On procède par récurrence sur l : si $l = 0$, on applique simplement la règle [DSTO] ; sinon, on raisonne par cas selon la forme de F_0 :

- si c'est un simple messager $(h_0 \triangleright h_1)$, on transmet le message par la règle [FWDO], on applique l'hypothèse d'induction (HI), et on relocalise le messager en utilisant la règle [UPDO] :

$$\begin{aligned} U &\xrightarrow{\tau}_{\lfloor o\rfloor} V' \mid h_0\not\triangleright \mid h_1\langle m_0\rangle_{h_0,\widetilde{h}} \mid \Pi_{0<i<l}F_i \mid h_l[n] && [\text{FWD}_o]\\ &\overset{\tau}{\Rightarrow}_{\lfloor o\rfloor} V' \mid h_0\not\triangleright \mid (h_0,\widetilde{h})\langle \triangleright h_l\rangle \mid \Pi_{0<i<l}h_i \triangleright h_l \mid h_l[m_0; m; n] \mid \widetilde{k} \triangleright h_l && (\text{HI})\\ &\xrightarrow{\tau}_{\lfloor o\rfloor} V' \mid \widetilde{h} \triangleright h_l \mid h_0 \triangleright h_l \mid \Pi_{i<l}h_i \triangleright h_l \mid h_l[m_0; m; n] \mid \widetilde{k} \triangleright h_l && [\text{UPD}_o] \end{aligned}$$

- Si c'est un messager bloqué, avec son message de relocalisation $(h_0\not\triangleright \mid h_0\langle \triangleright h_1\rangle)$, on relocalise le messager par la règle [UPDO], ce qui nous ramène au cas précédent.
- Dans le dernier cas $((h_0; \widetilde{k}_1)\not\triangleright \mid h_1\langle m_1\rangle_{h_0,\widetilde{k}_1})$, on applique l'hypothèse d'induction au message sur h_1, puis on transmet le message initial via le messager qui vient d'être relocalisé :

$$\begin{aligned} U = V' &\mid h_0\langle m_0\rangle_{\widetilde{h}} \mid (h_0; \widetilde{k}_1)\not\triangleright \mid h_1\langle m_1\rangle_{(h_0;\widetilde{k}_1)} \mid \Pi_{0<i<l}F_i \mid h_l[n]\\ &\overset{\tau}{\Rightarrow}_{\lfloor o\rfloor} V' \mid h_0\langle m_0\rangle_{\widetilde{h}} \mid (h_0; \widetilde{k}_1)\not\triangleright \mid (h_0; \widetilde{k}_1)\langle \triangleright h_l\rangle \mid \Pi_{0<i<l}h_i \triangleright h_l\\ &\qquad \mid h_l[m_1; m; n] \mid \widetilde{k} \triangleright h_l && (\text{HI})\\ &\overset{\tau}{\Rightarrow}_{\lfloor o\rfloor} V' \mid h_0\langle m_0\rangle_{\widetilde{h}} \mid (h_0; \widetilde{k}_1) \triangleright h_l \mid \Pi_{0<i<l}h_i \triangleright h_l \mid h_l[m_1; m; n] \mid \widetilde{k} \triangleright h_l && [\text{UPD}_o]\\ &\xrightarrow{\tau\,3}_{\lfloor o\rfloor} V' \mid \widetilde{h}\langle \triangleright h_l\rangle \mid \Pi_{i<l}h_i \triangleright h_l \mid h_l[m_0; m_1; m; n] \mid (h_0; \widetilde{k}_1; \widetilde{k}) \triangleright h_l && [\text{FWD}_o, \text{DST}_o, \text{UPD}_o] \end{aligned}$$

■

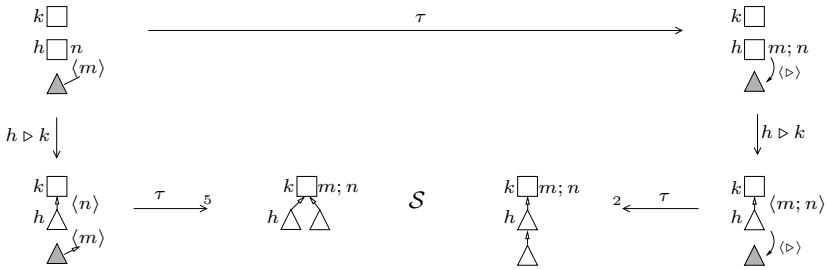


FIGURE 6.10 – Commutation modulo réorganisation des arbres de messagers.

Malgré la reconfiguration des arbres de messagers qu'entraîne l'acheminement des messages, l'algorithme est convergent (nous montrerons sa terminaison plus loin) :

Lemme 6.39. *La relation* $\xrightarrow{\tau}_{\lfloor o \rfloor}$ *est localement confluente.*

Démonstration. Par bonne formation, la seule paire critique est la suivante :

$$U = V \mid h\langle m\rangle_{\widetilde{h}} \mid h\langle n\rangle_{\widetilde{k}} \mid h \triangleright k$$
$$U \xrightarrow{\tau}_{\lfloor o \rfloor} V \mid h\langle m\rangle_{\widetilde{h}} \mid h\not\triangleright \mid k\langle n\rangle_{h,\widetilde{k}} = U_1 \qquad [\text{FWD}_o]$$
$$U \xrightarrow{\tau}_{\lfloor o \rfloor} V \mid h\langle n\rangle_{\widetilde{k}} \mid h\not\triangleright \mid k\langle m\rangle_{h,\widetilde{h}} = U_2 \qquad [\text{FWD}_o]$$

En utilisant le lemme 6.38 sur U_1, on peut acheminer le message n jusqu'à une localité k' :

$$\begin{aligned} U_1 &\overset{\widehat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} V' \mid h\langle m\rangle_{\widetilde{h}} \mid h\not\triangleright \mid (h,\widetilde{k})\langle \triangleright k'\rangle \mid k'[n;m'] && \text{(Lemme 6.38)} \\ &\xrightarrow{\tau}_{\lfloor o \rfloor} V' \mid h\langle m\rangle_{\widetilde{h}} \mid h \triangleright k' \mid \widetilde{k}\langle \triangleright k'\rangle \mid k'[n;m'] && [\text{UPD}_o] \\ &\xrightarrow{\tau\ 3}_{\lfloor o \rfloor} V' \mid (\widetilde{k},\widetilde{h})\langle \triangleright k'\rangle \mid h \triangleright k' \mid k'[m;n;m'] = U' && [\text{FWD}_o,\text{DST}_o,\text{UPD}_o] \end{aligned}$$

Le même raisonnement sur U_2 mène à $U_2 \overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} U'$. ■

Il nous faut maintenant vérifier les diagrammes visibles requis par le corollaire 4.53. Le mécanisme de relocalisation empêche cependant les transitions silencieuses de commuter avec les transitions visibles (les migrations, plus précisément) : comme l'indique la figure 6.10, il y a commutation, mais seulement modulo réorganisation des arbres de messagers. On introduit donc la relation suivante, qui permet justement de réorganiser pas à pas la structure des arbres de messagers : c'est intuitivement la relation E_1 de l'introduction, page 19 ; elle permet de remonter ou de redescendre d'un cran un messager faisant partie d'un arbre :

Définition 6.40 (Relation d'échange)**.** On note $\mathcal{S}$ la *relation d'échange*, définie comme la fermeture symétrique de la relation suivante, sur $\lfloor \mathcal{O} \rfloor$:

$$\left\{ \langle h \triangleright h' \mid h' \triangleright k \mid U,\ h \triangleright k \mid h' \triangleright k \mid U\rangle \mid \forall h, h', k, U \right\} .$$

Remarquons que $\mathcal{S}$ préserve les destinations (et les localités perdues) ; cette relation nous permet d'obtenir le lemme suivant : $\xrightarrow{\tau}_{\lfloor o \rfloor}$ commute avec les transitions visibles modulo $\mathcal{S}^\star$.

Lemme 6.41. *Si* $U \xrightarrow{\tau}_{\lfloor o \rfloor} V$ *et* $U \xrightarrow{\mu}_{\lfloor o \rfloor} U'$, *alors* $U' \stackrel{\widehat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} \cdot \mathcal{S}^\star \cdot \stackrel{\widehat{\mu}}{\Leftarrow}_{\lfloor o \rfloor} V$.

Démonstration. Le cas $\mu = \tau$ est donné par la confluence locale de $\xrightarrow{\tau}_{\lfloor o \rfloor}$ (lemme 6.39) ; sinon, on a $U' \xrightarrow{\tau}_{\lfloor o \rfloor} \cdot \xleftarrow{\mu}_{\lfloor o \rfloor} V$, sauf dans le cas suivant (qui nécessite $\mathcal{S}$) :

$$U = W \mid h\langle m \rangle_{\widetilde{h}} \mid h[n] \quad \xrightarrow{\tau}_{\lfloor o \rfloor} \quad W \mid h[m;n] \mid \widetilde{h}\langle \triangleright h \rangle = V \qquad [\textsc{Dst}_o]$$

$$U \xrightarrow{h \triangleright k}_{\lfloor o \rfloor} W \mid h\langle m \rangle_{\widetilde{h}} \mid h \triangleright k \mid k\langle n \rangle = U' \qquad [\textsc{Mig}_o]$$

où l'on a

$$V \xrightarrow{h \triangleright k}_{\lfloor o \rfloor} W \mid h \triangleright k \mid k\langle m;n \rangle \mid \widetilde{h}\langle \triangleright h \rangle = V' \ . \qquad [\textsc{Mig}_o]$$

On raisonne par cas, selon l'agent hébergé en k :

- si c'est une localité réelle ($k[n']$), en acheminant vers k les messages exhibés dans U' et V', on obtient :

  $$U' \stackrel{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid (h, \widetilde{h}) \triangleright k \mid k[m;n;n']$$
  $$V' \stackrel{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid \widetilde{h} \triangleright h \mid h \triangleright k \mid k[m;n;n']$$

  (le seul message à acheminer dans V' est presque arrivé à destination, et le message de relocalisation a déjà été diffusé aux messagers bloqués ($\widetilde{h}$), qui se relocalisent donc en h plutôt qu'en k).
  Finalement, on relocalise ces messagers sous k par l applications de la relation d'échange, l étant la taille de $\widetilde{h}$: $U' \stackrel{\widehat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} \cdot \mathcal{S}^l \cdot \stackrel{\widehat{\tau}}{\Leftarrow}_{\lfloor o \rfloor} V'$.
- S'il s'agit d'un messager ($k \triangleright k'$) : comme dans le cas précédent, on commence par acheminer les messages vers leur destination, notée k'' :

  $$U' \stackrel{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid (h, k, \widetilde{h}) \triangleright k'' \mid k''[m;n;n'] \ ,$$
  $$V' \stackrel{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid \widetilde{h} \triangleright h \mid h \triangleright k \mid k \triangleright k'' \mid k''[m;n;n'] \ .$$

  On a ici besoin d'appliquer la relation d'échange une première fois, pour relocaliser h sous k'', avant de pouvoir relocaliser les messagers de $\widetilde{h}$:

  $$U' \stackrel{\widehat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} \cdot \mathcal{S}^{l+1} \cdot \stackrel{\widehat{\tau}}{\Leftarrow}_{\lfloor o \rfloor} V' \ .$$
- Enfin, s'il s'agit d'un messager bloqué ($k \not\triangleright$), on raisonne comme dans le cas précédent, en commençant par acheminer le message qui bloque ce messager vers sa destination. ■

Si l'on pouvait maintenant prouver que $\mathcal{S}$ est contenue dans la bisimilarité faible, on pourrait obtenir la τ-inertie par le corollaire 4.53 : les lemmes 6.39 et 6.39 nous assureraient que le diagramme requis par ce corollaire est satisfait, et le lemme 6.43 que l'on prouve plus bas établit en particulier la réactivité du modèle propre et optimisé.

Cependant, comme pour le lemme 6.21 dans le cas simple, on a besoin de raisonner modulo $\xrightarrow{\tau}_{\lfloor o \rfloor}$ (entre autres) pour prouver que cette relation auxiliaire est contenue dans la bisimilarité faible : on a seulement le lemme de commutation suivant.

Lemme 6.42. *Si $U\ \mathcal{S}\ V$ et $U \xrightarrow{\mu}_{\lfloor o \rfloor} U'$ alors $U' \overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} \cdot\, \mathcal{S}^\star \cdot \overset{\widehat{\mu}}{\Leftarrow}_{\lfloor o \rfloor} V$.*

Démonstration. C'est immédiat lorsque $\mu \neq \tau$: on a $U'\ \mathcal{S} \cdot \overset{\mu}{\Leftarrow}_{\lfloor o \rfloor} V$. Lorsque $\mu = \tau$, les cas intéressants sont ceux où la transition silencieuse $U \xrightarrow{\tau}_{\lfloor o \rfloor} U'$ correspond à la transmission d'un message par l'un des messagers qui sont échangés :

– $U = W \mid h\langle m \rangle_{\widetilde{h}} \mid h \triangleright h' \mid h' \triangleright k \xrightarrow{\tau}_{\lfloor o \rfloor} W \mid h \not\triangleright \mid h'\langle m \rangle_{h,\widetilde{h}} \mid h' \triangleright k = U'$.
En acheminant les messages, on obtient :

$$\begin{aligned} U' &\overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid h \triangleright k' \mid h' \triangleright k' \mid k'[m;n] = U'' \ , \\ V &\overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid h \triangleright k' \mid h' \triangleright k \mid k'[m;n] = V' \ . \end{aligned}$$

Si $k = k'$, c'est immédiat : $U'' = V'$. Sinon, il y a un messager $k \triangleright k'$ dans W', et il faut une application de la relation d'échange pour relocaliser le messager hébergé en h' dans V'.

– $U = h \triangleright h' \mid h'\langle m \rangle_{\widetilde{h}} \mid h' \triangleright k \xrightarrow{\tau}_{\lfloor o \rfloor} h \triangleright h' \mid h' \not\triangleright \mid k\langle m \rangle_{h',\widetilde{h}} = U'$
En acheminant les messages, on obtient :

$$\begin{aligned} U' &\overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid h \triangleright h' \mid h' \triangleright k' \mid k'[m;n] = U'' \ , \\ V &\overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} W' \mid h \triangleright k \mid h' \triangleright k' \mid k'[m;n] = V' \ . \end{aligned}$$

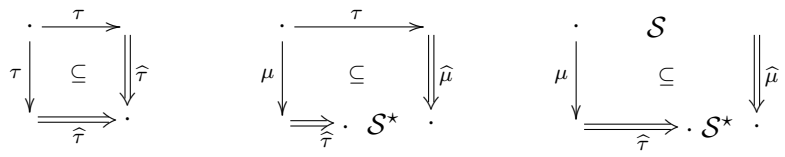
Si $k = k'$, c'est là aussi immédiat : $U'' = V'$. Sinon, on a $W' = W'' \mid k \triangleright k'$, et il faut deux échanges pour relocaliser dans les deux réseaux le messager hébergé en h :

$$\begin{aligned} U'' = W'' &\mid h \triangleright h' \mid k \triangleright k' \mid h' \triangleright k' \mid k'[m;n] \\ \mathcal{S}\ W'' &\mid h \triangleright k' \mid k \triangleright k' \mid h' \triangleright k' \mid k'[m;n] \\ \mathcal{S}\ W'' &\mid h \triangleright k \mid k \triangleright k' \mid h' \triangleright k' \mid k'[m;n] = V' \end{aligned}$$

Cette analyse s'applique aussi dans les cas symétriques où la transition silencieuse est jouée par le réseau dont les messagers échangés sont « à plat ». ■

En résumé, nous avons vérifié les trois diagrammes ci-dessous :

(lemme 6.39) (lemme 6.41) (lemme 6.42)



On ne peut donc pas appliquer le corollaire 4.53, mais il nous suffit d'utiliser le corollaire 4.55, qui permet de prouver conjointement que $\xrightarrow{\tau}_{\lfloor o \rfloor}$ et $\mathcal{S}$ sont contenues dans la bisimilarité faible. Il reste pour cela à prouver la terminaison de $\xrightarrow{\tau}_{\lfloor o \rfloor}$ modulo la relation d'échange :

Lemme 6.43. *$\xrightarrow{\tau}_{\lfloor o \rfloor} \cdot \mathcal{S}^\star$ est fortement normalisante.*

Démonstration. Appelons *taille* d'un réseau U le triplet $s(U) \triangleq \langle n, r, l \rangle$, où n est le nombre de messages en attente, r est le nombre de messages de relocalisation, et l est le nombre de messagers qui ne sont pas bloqués. Ces triplets sont ordonnés lexicographiquement. On vérifie alors que $U \mathcal{S} V$ implique $s(U) = s(V)$, et que cette taille diminue strictement le long des transitions silencieuses. ■

Théorème 6.44. *Le modèle propre et optimisé est réactif et τ-inerte ; pour tous $U, V \in \lfloor \mathcal{O} \rfloor$, si $U \mathcal{S} V$ ou $U \xrightarrow{\tau}_{\lfloor o \rfloor} V$, alors*

$$\langle U, \rightarrow_{\lfloor o \rfloor} \rangle \approx \langle V, \rightarrow_{\lfloor o \rfloor} \rangle \ .$$

Démonstration. La réactivité découle du lemme 6.43 ; pour l'inertie, il suffit d'appliquer le corollaire 4.55 en prenant $z = S$; nous venons d'en vérifier toutes les hypothèses. ■

Remarquons que l'usage du corollaire 4.55 est crucial pour la preuve précédente : il permet de se concentrer sur les relations « locales » $\mathcal{S}$ et $\xrightarrow{\tau}_{\lfloor o \rfloor}$, qui ne mettent en relation que des réseaux qui diffèrent très légèrement. Cela nous a permis de raisonner en considérant leurs (petites) différences syntaxiques. Si nous avions dû prouver le théorème précédent de manière directe, il nous aurait fallu montrer que $(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor})^\star$ est une simulation, ce qui est vraiment difficile : cette relation met l'un en face de l'autre des réseaux complètement différents (les arbres de messagers sont notamment réorganisés de façon arbitraire).

Correction vis-à-vis du modèle simple

Nous pouvons maintenant montrer que les modèles simples et optimisés sont équivalents, ce qui mène à la correction de l'optimisation vis-à-vis de la spécification, l'implantation simple étant correcte.

Les lemmes 6.43 et 6.39 impliquent que la relation $\xrightarrow{\tau}_{\lfloor o \rfloor}$ définit une unique forme normale pour tout réseau optimisé et propre U ; on note $U_{\downarrow o}$ cette forme normale. Notons que $U_{\downarrow o}$ est toujours un réseau de référence : il ne contient ni messagers bloqués, ni messages en attente.

La normalisation d'un réseau simple propre par $\xrightarrow{\tau}_{\lfloor s \rfloor}$ et $\xrightarrow{\tau}_{\lfloor o \rfloor}$ ne mène cependant pas forcément au même réseau de référence : $U_{\downarrow} \neq U_{\downarrow o}$. Cependant ces réseaux ne diffèrent que par l'organisation de leurs arbres de messagers : ils sont en relation par $\mathcal{S}^\star$.

Lemme 6.45. *Pour tout réseau simple propre U, on a :*

$$U_{\downarrow}\ \mathcal{S}^\star\ U_{\downarrow o}\ .$$

Démonstration. On procède par induction bien fondée sur U, en utilisant la terminaison de la relation $\xrightarrow{\tau}_{\lfloor s \rfloor}$:

- Si U est un réseau de référence, on a $U_{\downarrow} = U = U_{\downarrow o}$.
- Sinon, on a $U \xrightarrow{\tau}_{\lfloor s \rfloor} U'$; et comme U est propre et simple, on obtient :

$$\begin{aligned}
&U = V \mid h_0\langle m\rangle \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[n] \\
&U \xrightarrow{\tau}_{\lfloor s \rfloor} U' \overset{\tau}{\Rightarrow}_{\lfloor s \rfloor} V \mid \Pi_{i<l}\ h_i \triangleright h_{i+1} \mid h_l[m;n] = U_1 && [\textsc{Fwd}_s, \textsc{Dst}_s] \\
&U \overset{\tau}{\Rightarrow}_{\lfloor o \rfloor} V \mid \Pi_{i<l}\ h_i \triangleright h_l \mid h_l[m;n] = U_2 && \text{(lemme 6.38)}
\end{aligned}$$

On vérifie que $U_1\ \mathcal{S}^l\ U_2$, et on a $U_1 \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} U_{1\downarrow o}$ de telle sorte que, par le théorème 6.44, $U_2 \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} U_2'$ avec $U_{1\downarrow o}(\mathcal{S} \cup \xrightarrow{\tau}_{\lfloor o \rfloor})^\star U_2'$.
De plus, comme $\mathcal{S}$ préserve les formes normales, on a $U_{1\downarrow o}\ \mathcal{S}^\star\ U_2'$, et $U_2' = U_{2\downarrow o}$. Finalement, on obtient par induction $U_{1\downarrow}\ \mathcal{S}^\star\ U_{1\downarrow o}$, puis $U_{\downarrow} = U_{1\downarrow}\ \mathcal{S}^\star\ U_{1\downarrow o}\ \mathcal{S}^\star\ U_{2\downarrow o} = U_{\downarrow o}$. ■

Il s'en suit que dans le modèle propre et optimisé, tout réseau propre et simple est bisimilaire à sa forme normale vis-à-vis de $\xrightarrow{\tau}_{\lfloor s \rfloor}$ (cela correspond au corollaire 6.22 dans le modèle propre et simple) :

Corollaire 6.46. *Pour tout réseau simple et propre U, on a :*

$$\langle U, \rightarrow_{\lfloor o \rfloor}\rangle \approx \langle U_{\downarrow o}, \rightarrow_{\lfloor o \rfloor}\rangle \approx \langle U_{\downarrow}, \rightarrow_{\lfloor o \rfloor}\rangle\ .$$

Démonstration. Par le lemme 6.45, $U \overset{\widehat{\tau}}{\Rightarrow}_{\lfloor o \rfloor} U_{\downarrow o}\ \mathcal{S}^\star\ U_{\downarrow}$; on conclut par le théorème 6.44. ■

On peut finalement conclure en utilisant la technique « bisimulation faible modulo $\approx$ lors des offres visibles » (corollaire 3.43) : grâce à cette technique, on peut ne considérer que des réseaux de référence, ne contenant aucun message en attente, et normaliser ces réseaux dès qu'ils effectuent une transition (visible) qui introduit des messages en attente. Comme les réseaux de référence ne donnent pas lieu à des transitions silencieuses, la contrainte imposée par la technique modulo n'est en fait pas gênante.

Lemme 6.47. *Pour tout réseau de référence U, on a :*

$$\langle U, \to_{\lfloor o \rfloor} \rangle \approx \langle U, \to_{\lfloor s \rfloor} \rangle \ .$$

Démonstration. On montre que la relation suivante est une bisimulation faible modulo $\approx$ lors des offres visibles (corollaire 3.43) :

$$R \triangleq \left\{ \langle \langle U, \to_{\lfloor \mathrm{o} \rfloor} \rangle, \ \langle U, \to_{\lfloor \mathrm{s} \rfloor} \rangle \rangle \mid U \in \mathcal{U}_{\mathrm{r}} \right\} \ .$$

Pour tout réseau de référence U, comme U ne contient pas de messages en attente, on a $U \xrightarrow{\mu}_{\lfloor \mathrm{o} \rfloor} V$ si et seulement si $U \xrightarrow{\mu}_{\lfloor \mathrm{s} \rfloor} V$. Par contre, bien que V soit un réseau simple et propre, ce n'est plus un réseau de référence, on n'a donc pas $\langle V, \to_{\lfloor \mathrm{o} \rfloor} \rangle \ R \ \langle V, \to_{\lfloor \mathrm{s} \rfloor} \rangle$. Néanmoins, $V_{\downarrow}$ est un réseau de référence, et par les corollaires 6.46 et 6.22, on a :

$$\langle V, \to_{\lfloor \mathrm{o} \rfloor} \rangle \approx \langle V_{\downarrow}, \to_{\lfloor \mathrm{o} \rfloor} \rangle \ R \ \langle V_{\downarrow}, \to_{\lfloor \mathrm{s} \rfloor} \rangle \precsim \langle V, \to_{\lfloor \mathrm{s} \rfloor} \rangle \ .$$

On utilise donc la bisimilarité faible pour réécrire le réseau de gauche ; c'est permis par la technique modulo : nous répondons à une offre visible. ■

La preuve du théorème final de correction se résume à assembler les résultats précédents, selon la figure 6.11 détaillée plus bas.

Théorème 6.48. *Pour tout réseau optimisé U, on a :*

$$\langle U, \to_o \rangle \approx \langle \lfloor U \rfloor_{\downarrow}, \to_r \rangle \ .$$

Démonstration. $\lfloor U \rfloor$ est propre et optimisé, et $\lfloor U \rfloor_{\downarrow}$ est un réseau de référence, on a donc :

$$\begin{aligned}
\langle U, \to_{\mathrm{o}} \rangle &\succsim \langle \lfloor U \rfloor, \to_{\lfloor \mathrm{o} \rfloor} \rangle && \text{(proposition 6.37)} \\
&\approx \langle \lfloor U \rfloor_{\downarrow}, \to_{\lfloor \mathrm{o} \rfloor} \rangle && \text{(corollaire 6.46)} \\
&\approx \langle \lfloor U \rfloor_{\downarrow}, \to_{\lfloor \mathrm{s} \rfloor} \rangle && \text{(lemme 6.47)} \\
&\precsim \langle \lfloor U \rfloor_{\downarrow}, \to_{\mathrm{r}} \rangle \ . && \text{(théoreme 6.25)}
\end{aligned}$$

■

En particulier, si U est un réseau de référence, alors $\langle U, \to_{\mathrm{o}} \rangle \approx \langle U, \to_{\mathrm{r}} \rangle$.

La figure 6.11 récapitule les grandes étapes de la preuve de correction ; nous concluons ce chapitre en comparant les techniques modulo utilisées pour le modèle simple et le modèle optimisé.

Alors qu'une preuve d'expansion directe suffisait à retirer les messages perdus dans le cas simple, nous avons utilisé la technique bi-expansion modulo transitivité dans le cas optimisé. Le passage au modèle propre correspondant s'est fait dans les deux cas par expansion modulo expansion. Pour

$$\begin{array}{ccccc} & & \text{simple} & & \text{optimisé} \\ & & \wr\Upsilon & & \wr\Upsilon \\ \text{référence} & \precsim & \text{propre, simple} & \approx & \text{propre, optimisé} \end{array}$$

FIGURE 6.11 – Grandes étapes de la preuve de correction.

montrer la τ-inertie du modèle simple, nous avons pu raisonner par expansion modulo transitivité ; dans le modèle optimisé, où les transitions silencieuses ne sont contenues que dans la bisimilarité faible, il nous a fallu utiliser le corollaire 4.55, qui s'appuie sur une hypothèse de terminaison pour autoriser la transitivité dans le cas de la bisimulation faible. Enfin, pour la preuve de correction finale, nous avons raisonné modulo expansion pour le modèle simple, et modulo bisimilarité faible lors des offres visibles pour le modèle optimisé.

Dans les deux cas, les techniques modulo nous ont permis de ne raisonner que de façon locale : les seuls diagrammes que nous ayons effectivement fermés sont des diagrammes locaux, où les offres portent sur de « petites » relations.

Chapitre 7

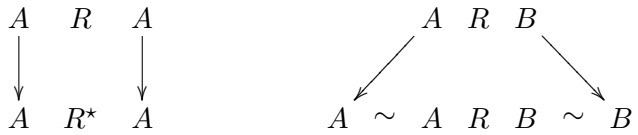
Conclusions et pistes de recherche

En résumé, nous avons tout d'abord développé une théorie relativement abstraite des techniques modulo pour la co-induction, que nous avons ensuite appliquée pour retrouver les techniques modulo standard pour les différentes notions de bisimulation. Ce faisant, il est apparu que ces techniques gagnaient à être étudiées dans le cadre abstrait des algèbres relationnelles. Certaines de ces techniques sont problématiques dans le cas faible ; nous avons montré que ces problèmes proviennent essentiellement de problèmes similaires apparaissant en théorie de la réécriture. Cela nous a permis, en utilisant les outils de la réécriture, de trouver de nouvelles techniques pour la bisimulation faible, s'appuyant sur des hypothèses de terminaison ; de l'un de ces nouveaux résultats de commutation découle un lemme de confluence a priori nouveau, qui pourrait être utile en réécriture. En utilisant notre théorie des techniques modulo pour la co-induction, nous avons ensuite montré comment combiner ces nouvelles techniques aux techniques plus standard, puis nous avons étudié diverses conséquences de ces nouveaux résultats, notamment dans le cas des systèmes dits réactifs. Toujours en s'appuyant sur la théorie générale des techniques modulo pour la co-induction, nous avons défini une nouvelle méthode, permettant de prouver facilement la validité des techniques modulo contexte, que nous avons illustrée en l'appliquant dans le cas de CCS. Enfin, nous avons démontré l'utilité de l'une des nouvelles techniques modulo obtenues, en l'utilisant afin de donner une preuve de correction « locale » et modulaire pour une optimisation d'un modèle d'exécution pour des processus distribués. La figure 7.1 établit les dépendances entre ces différents résultats.

Nous indiquons finalement quelques directions dans lesquelles nous aimerions poursuivre notre travail.

Catégories, allégories

Afin de donner une présentation uniforme des résultats pour la bisimulation, nous ne l'avons définie qu'au sein d'un unique LTS, arbitraire. Plus généralement, toutes les relations binaires que nous avons manipulées ne relient que des éléments du même type, ce qui se traduit au niveau abstrait des algèbres relationnelles par le fait que la composition relationnelle soit toujours définie. L'avantage principal est que nous n'avons pas à « typer » les objets manipulés, et que des opérations telles que la fermeture transitive sont toujours définies. Ainsi, la technique modulo transitivité, qui ne fait sens qu'au sein d'un unique LTS, permet de capturer la technique modulo $\sim$, bien qu'elle s'étende au cas de systèmes différents : si l'on type les diagrammes leur correspondant, ces deux techniques peuvent être représentées ainsi ($A\ R\ B$ signifiant que R est une relation entre les éléments de A et ceux de B) :



Par contre, l'inconvénient de notre approche, où nous supposons qu'il n'y a qu'un type ($A = B$), est qu'il nous faut utiliser des unions disjointes lorsque nous souhaitons, comme au chapitre 6, comparer les états de plusieurs LTS (cf. remarque 6.5).

La théorie des catégories, en nous évitant d'avoir à fixer un « univers », pourrait sûrement aider à surmonter ce problème. Les allégories [FS90, Joh02], qui sont le pendant catégorique des algèbres relationnelles, semblent notamment prometteuses.

Modèles

Il y a deux avantages principaux à se placer à un niveau d'abstraction élevé (les treillis complets éventuellement monoïdaux pour la co-induction ; les algèbres relationnelles pour la bisimulation) : on gagne en clarté, puisque les objets manipulés sont plus simples ; et en généralité, puisque l'on ne s'appuie pas sur les propriétés inhérentes à un modèle particulier. On a ainsi pu appliquer la théorie des techniques modulo, initialement conçue pour le treillis complet monoïdal des relations binaires, dans un treillis complet monoïdal de fonctions, afin d'obtenir des techniques modulo pour caractériser certaines propriétés de ces fonctions.

Comme indiqué à la remarque 3.6, nous aimerions savoir s'il existe des modèles non-triviaux d'algèbres relationnelles, dans lesquels la notion de semi-commutation jouerait un rôle suffisamment intéressant pour que nos techniques puissent y être utiles. Pour l'instant, nous manquons cruellement de culture à ce sujet.

Preuves assistées

Nous projetons de formaliser dans l'assistant de preuve Coq [CH84] les chapitres 2, 3, 4 et 5 de cette thèse. Nous l'avons fait pour une version préliminaire de certains résultats présentés ici (l'article [Pou05b] est entièrement formalisé, les développements sont accessibles via [Pou05c]) ; mais nous espérons cette fois pouvoir aller plus loin qu'une nouvelle preuve formelle, en définissant deux outils qui ne serviraient pas qu'à vérifier que nous ne nous sommes pas trompés...

Le premier outil serait un ensemble de tactiques permettant de raisonner de manière calculatoire lorsque l'on ferme des diagrammes de semi-commutation ; tout comme nous l'avons fait à la section 4.2, en suivant la méthode de Doornbos et al. [DBvdW97] pour les inductions bien fondées. Cet outil nous permettrait entre autres de prouver plus facilement nos résultats de commutation, mais il pourrait surtout servir pour rechercher de nouvelles techniques, en permettant d'effectuer automatiquement une partie des calculs auxquelles elles se ramènent. A priori, la seule difficulté, technique, consiste à définir des tactiques permettant de raisonner modulo l'associativité du monoïde et modulo le respect de l'ordre partiel par le monoïde : alors qu'il ne nous faut qu'une étape de raisonnement pour déduire l'inégalité $x \cdot y \cdot z \sqsubseteq y \cdot x \cdot z$ à partir de l'hypothèse (H) : $x \cdot y \sqsubseteq y \cdot x$, une preuve formelle doit passer par l'arbre de dérivation suivant :

$$\dfrac{\dfrac{\dfrac{}{x \cdot y \sqsubseteq y \cdot x}\,(\mathrm{H})}{(x \cdot y) \cdot z \sqsubseteq (y \cdot x) \cdot z}\,(\textsc{Respect})}{(x \cdot y) \cdot z \sqsubseteq y \cdot (x \cdot z)}\,(\textsc{Assoc.})}{x \cdot (y \cdot z) \sqsubseteq y \cdot (x \cdot z)}\,(\textsc{Assoc.})$$

Les tactiques à définir devraient par exemple nous permettre d'appliquer une hypothèse sur un sous-terme arbitraire.

La seconde idée consiste à développer, à partir des outils précédents et de la formalisation de la bisimilarité et de nos résultats, un autre jeu de tactiques permettant de faire facilement des preuves par bisimulation (modulo). Cela pourrait se résumer à des tactiques permettant de répondre automatiquement à certaines offres inintéressantes lors des jeux de bisimulation, où à appliquer automatiquement les techniques modulo contexte, pour simplifier les processus obtenus. Mais cela pourrait aussi aller plus loin, en donnant par exemple la possibilité de définir le candidat de bisimulation de façon paresseuse, en l'étendant au fur et à mesure, selon les besoins (tout comme on construit des bisimulations « à la volée » dans certains outils de vérification automatique [FMJJ92]).

Ce second outil pourrait par ailleurs être adapté, afin de pouvoir faciliter des preuves formelles de confluence.

Systèmes partiellement réactifs, bisimulation arborescente

Nous avons étudié à la section 4.5 les conséquences de notre nouveau résultat dans le cas des systèmes réactifs, où les transitions silencieuses doivent terminer : l'élaboration ($\gtrapprox$) est un préordre contrôlé, et on a accès à des techniques puissantes pour montrer la τ-inertie.

Cette hypothèse de terminaison n'est cependant pas toujours satisfaite : certains algorithmes distribués reposent sur une introduction volontaire de boucles silencieuses. Bien que la technique principale que l'on a obtenue (théorème 4.30, dont découle le corollaire 4.37) ne demande pas cette hypothèse, et puisse donc a priori être utilisé directement, les résultats de la section 4.5 nous semblent suffisamment attrayants pour que nous souhaitions les adapter au cas non-réactif.

Suivant l'approche de Groote et Sellink [GS96], nous avons proposé dans le cas de l'élaboration [Pou06b] de distinguer une partie des actions silencieuses, dites progressives, et de demander leur terminaison relativement aux autres. On permet ainsi l'étude de systèmes partiellement réactifs, et on aboutit à une notion d'élaboration qui peut être utilisée lors des jeux de bisimulation faible, mais seulement lors des offres progressives.

Si la version symétrique du théorème 4.28 que l'on propose page 118 s'avère correcte, elle pourrait permettre d'aller dans cette direction de manière plus générale, et notamment d'obtenir de nouveaux résultats pour prouver la τ-inertie de systèmes partiellement réactifs. Une autre idée, que nous souhaitons explorer plus en détails, consiste à étudier ce genre de techniques en utilisant la *bisimilarité arborescente* de Rob van Glabbeek et Peter Weijland [vGW96] : cette équivalence comportementale, qui est plus fine que la bisimilarité faible, permet en effet de distinguer sémantiquement différentes sortes d'actions silencieuses. Les techniques modulo développées dans cette thèse s'adaptent de façon relativement directe à cette notion de bisimilarité, et il n'est pas exclu que les propriétés spécifiques de cette bisimilarité nous permettent en fait d'aller plus loin.

Congruences et équivalences barbelées

Dans certains calculs (le π-calcul asynchrone par exemple), la bisimilarité étiquetée est souvent trop fine : les étiquettes permettent de distinguer des comportements que les contextes ne distinguent pas. On préfère souvent dans ce cas considérer l'*équivalence barbelée* [MS92b, FG05] : la plus grande congruence contenue dans la bisimilarité *barbelée*. Notre approche abstraite de la bisimilarité nous paraît relativement bien adaptée pour l'analyse de telles équivalences : étant donné une clôture $\mathcal{C}$, représentant la propriété de congruence à satisfaire, nous pouvons définir son *adjointe* [Ore44] comme étant la fonction $\mathcal{C}^\circ : x \mapsto \bigvee \{y \mid \mathcal{C}(y) \sqsubseteq x\}$. Dès que la clôture distribue sur les bornes supérieures (comme c'est le cas des clôtures par les contextes

monadiques) on a $\mathcal{C} \circ \mathcal{C}^\circ = \mathcal{C}^\circ$; de telle sorte que la fonction $\mathcal{C}^\circ$ associe à tout élément x la plus grande congruence majorée par x. Par exemple, si $\mathbf{w}_\mathrm{b}$ est une adaptation de $\mathbf{w}$ qui prend en compte les barbes, $\mathcal{C}^\circ(\boldsymbol{\nu}\overleftrightarrow{\mathbf{w}_\mathrm{b}})$ est la plus grande congruence contenue dans la bisimilarité barbelée (i.e. l'équivalence barbelée). Une autre approche standard consiste à clore la relation par les contextes, à chaque étape du jeu de bisimulation ; c'est ainsi que l'on définit par exemple de la *congruence barbelée* [HY95, FG05], qui est à la fois une congruence et une bisimulation barbelée. Dans notre cadre, nous pouvons capturer cette approche en considérant la génératrice $\overleftrightarrow{\mathbf{w}_\mathrm{b}} \wedge \mathcal{C}^\circ$.

Il serait donc utile d'étudier si des techniques modulo peuvent être définies, qui permettent de réduire le nombre de contextes à considérer dans de telles situations. Plus généralement, il faudrait réussir à mieux comprendre les interactions possibles entre une « génératrice de jeu », telle que $\mathbf{w}_\mathrm{b}$, et une « génératrice de congruence », telle que $\mathcal{C}^\circ$. Notre méthode des contextes initiaux va dans ce sens, il nous faudra continuer à chercher dans cette direction.

Ordre supérieur, environnements

La définition d'une théorie « comportementale » pour des calculs d'ordre supérieur est un problème difficile [Pit07, GP98] : Samson Abramski a défini la *bisimilarité applicative* pour le λ-calcul [Abr90], mais l'extension de cette notion à des calculs incluant des aspects impératifs (références) pose problème, notamment pour ce qui est des propriétés de congruence et des techniques modulo contexte [How96, Las98, Las05, KW06].

L'une des difficultés provient du fait que les étiquettes que l'on définit dans de tels langages contiennent généralement des processus (d'où l'ordre supérieur), et doivent être traitées de manière plus subtile lors des jeux de bisimulation : l'égalité syntaxique usuellement requise entre l'étiquette de la question et celle de la réponse n'est plus satisfaisante.

Davide Sangiorgi, Naoki Kobayashi et Eijiro Sumii ont récemment proposé deux solutions [SKS07b, SKS07a] ; la première utilise le candidat de bisimulation lui-même pour contraindre les étiquettes, mais d'une façon qui rend la génératrice non-croissante, ce qui rend improbable l'utilisation de nos techniques dans un tel cadre. La seconde généralise les idées de la première, en utilisant un *environnement* pour quantifier le jeu sur les étiquettes. Bien que ce ne soit pas immédiat, il nous semble que cette dernière méthode peut être définie dans notre cadre, à condition de bien choisir l'algèbre relationnelle et le LTS dans lequel on se place (les processus eux-mêmes doivent être annotés par l'environnement) : nous devrions pouvoir se ramener à des jeux de bisimulation standard, où les étiquettes sont de premier ordre, et où l'environnement permet d'imposer indépendamment les contraintes nécessaires. Si tel est réellement le cas, nous pourrons appliquer directement les techniques développées dans cette thèse, et la méthode des contextes initiaux

devrait grandement simplifier l'étude des techniques modulo contexte pour ce genre de calculs (avec un peu de chance, cela pourrait fournir un exemple d'application de la seconde partie du théorème 5.8, qui permet de prouver la compatibilité d'une fonction « à travers » la correction d'une technique arbitraire).

Travaux non détaillés

D'autres travaux ont été menés durant cette thèse, qui ne sont pas détaillés dans ce document ; nous les indiquons rapidement ci-dessous.

Machines abstraites et calculs distribués. Comme indiqué au début du chapitre 6, le modèle d'exécution présenté dans ce dernier chapitre provient de l'étude d'une optimisation d'une machine abstraite pour les *Safe Ambients* [LS03]. Nous avons implanté cette machine ainsi que son optimisation de façon réellement distribuée, en utilisant le langage OCaml. L'optimisation est décrite dans [HPS05], et l'implantation est accessible via [Pou05a].

Dans ce domaine des machines abstraites distribuées, nous avons aussi travaillé avec Tom Hirschowitz, Daniel Hirschkoff et l'équipe SARDES de l'INRIA Rhône Alpes sur des calculs dérivés des *Kells* [SS04] : nous avons défini une extension du π-calcul comprenant une notion de localité et une primitive de *passivation*, qui permettent de raisonner sur des propriétés de modularité dynamique. Nous avons défini une machine abstraite distribuée pour ce calcul que nous avons ensuite implantée, toujours en OCaml.

Congruence et axiomatisations. Indépendamment, nous avons prouvé avec Daniel Hirschkoff que la bisimilarité est une congruence dans le π-calcul sans somme ni réplication [HP07], problème réputé difficile [SW01, chapitre 5] et connu depuis [BS98]. La preuve passe par une axiomatisation de la bisimilarité sur un fragment encore plus restreint de CCS (sans somme, ni réplication, ni restriction), puis exploite un résultat de transfert qui permet d'obtenir la clôture par substitution de la bisimilarité dans le π-calcul à partir de propriétés établies dans CCS à l'aide de l'axiomatisation.

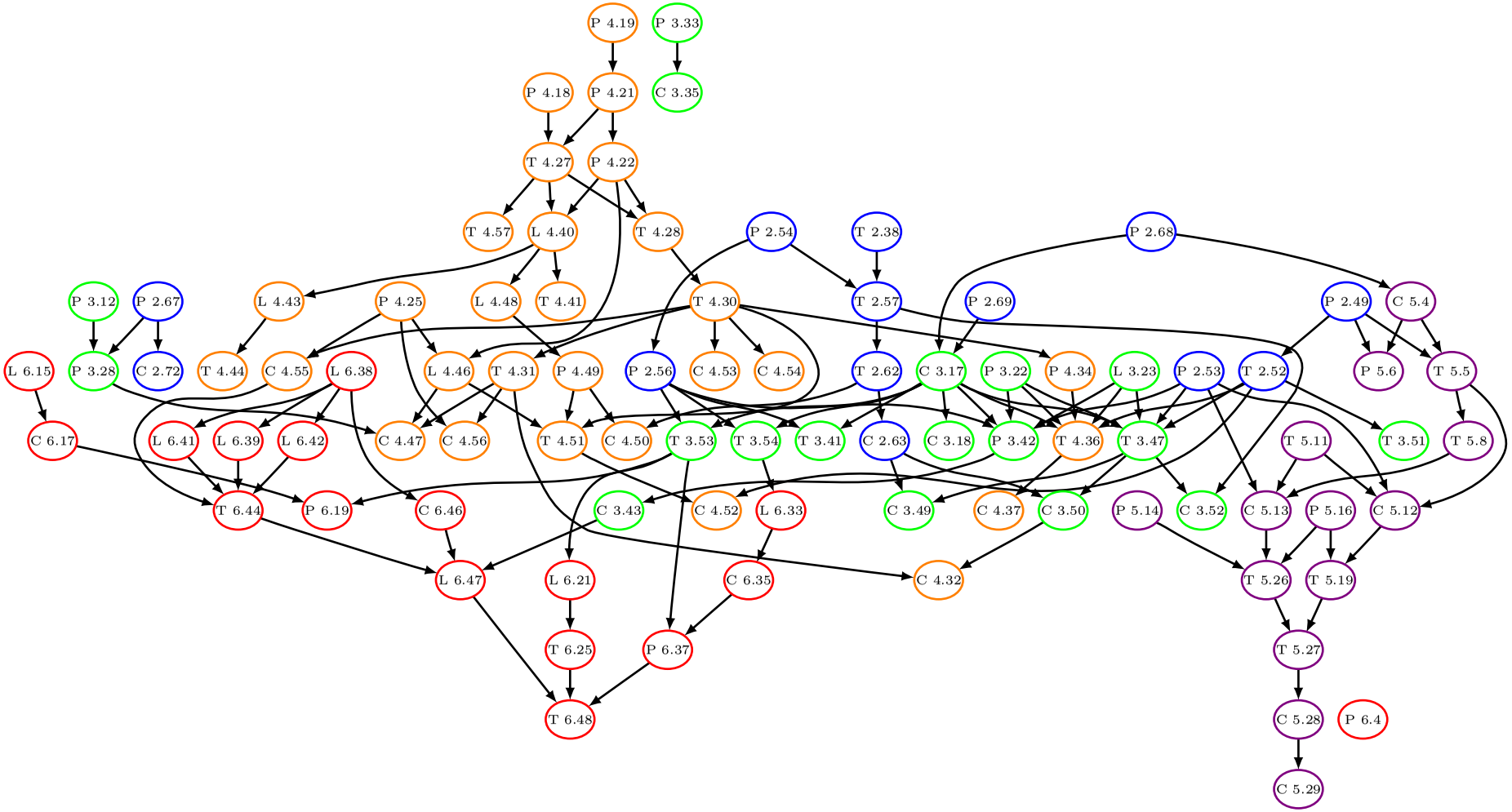


FIGURE 7.1 – Dépendances entre les principaux résultats.

Liste des définitions

Liste des figures

Liste des notations

Notations générales

$\mathcal{P} \times \mathcal{Q}$	produit cartésien de deux ensembles $\mathcal{P}$ et $\mathcal{Q}$
$\mathscr{P}(\mathcal{P})$	ensemble des parties d'un ensemble $\mathcal{P}$
$\mathcal{P} \setminus \mathcal{Q}$	différence ensembliste de $\mathcal{P}$ et $\mathcal{Q}$
$\emptyset$	ensemble vide, multi-ensemble vide, liste vide
$\subseteq$	inclusion ensembliste
$\bigcap$ ($\cap$)	intersection ensembliste (binaire)
$\bigcup$ ($\cup$)	union ensembliste (binaire)
$\mathcal{P} \setminus \mathcal{Q}$	différence ensembliste
$\langle x, y \rangle$	couple
$\{x_1, \ldots, x_n\}$	ensemble donné par énumération
$\{x \mid P(x)\}$	ensemble donné par compréhension
$\mathbb{N}$	ensemble des entiers naturels
$\overline{\mathbb{N}}$	ensemble des entiers naturels, complété par un élément maximal
$\leq_{\mathbb{N}}, (<_{\mathbb{N}})$	ordre (strict) sur les entiers naturels
$[m; n]$	ensemble des entiers compris entre m et n
$\triangleq$	égalité par définition
$P \Rightarrow Q$	implication logique
$\neg P$	négation logique
$\forall$, $\exists$	quantifications universelles et existentielles

Structures

X	domaine d'une structure (treillis complet, monoïde, algèbre relationnelle)
$x, y, z, t \ldots$	éléments de X (relations abstraites)

$Y, Z, \ldots$	parties de X
$\sqsubseteq$	ordre partiel 31
$\sqsupseteq$	converse de $\sqsubseteq$
$\bigvee Y$ $(x \vee y)$	borne supérieure (binaire) 32
$\bigwedge Y$ $(x \wedge y)$	borne inférieure (binaire) 32
$\top$, $(\bot)$	élément maximal (minimal) du treillis complet 32
Y^l	ensemble des éléments majorés par par tous ceux de Y .. 32
$x \cdot y$	produit de deux éléments 34
1	élément neutre d'un monoïde 34
$Y \cdot Y'$	produit des parties Y et Y' 34
x^n	itéré n fois de x 34
$x^=$	fermeture réflexive de x 35
x^+	fermeture transitive de x 35
$x^\star$	fermeture réflexive transitive de x 35
$\overline{x}$	converse d'un élément 55
$A, B \ldots$	monotypes (éléments majorés par 1) 125
$/$, $\backslash$	facteur à gauche, à droite 124
$\not/$, $\not\backslash$	facteur monotype à gauche, à droite 126

Fonctions

$X^X, X \to X$	ensemble des fonctions (totales) de X dans X 38
$X^{\langle X \rangle}$	ensemble des fonctions croissantes de X dans X 38
$X^{[X]}$	ensemble des fonctions continues de X dans X 42
$f, g, h \ldots$	fonctions
$f(Y)$	image par f de la partie Y 38
id_X	fonction identité sur X 38
$\circ$	composition fonctionnelle 38
$\widehat{x}$	fonction constante égale à x 38
$\hat{\cdot}$	extension point à point du produit aux fonctions 38
$f \sqsubseteq g$	extension point à point de l'ordre partiel aux fonctions .. 38
$\bigvee F$, $(\bigwedge F)$	extension point à point des bornes supérieures (inférieures) aux fonctions 38
f^n	: $x \mapsto f(x)^n$ 41
$f^=$	: $x \mapsto f(x)^=$ 41
f^+	: $x \mapsto f(x)^+$ 41
$f^\star$	: $x \mapsto f(x)^\star$ 41
$\mathrm{id}_X^\star$	fonction de fermeture réflexive transitive ($f^\star = \mathrm{id}_X^\star \circ i$) .. 41
$f^{(n)}$	itérée n fois de f $(\underbrace{f \circ \ldots \circ f}_{n \text{ fois}})$ 41

f^ω	fermeture par itération de f ($\bigvee_{n\in\mathbb{N}} f^{(n)}$)	41
$\varphi, \ldots$	fonctionnelles (fonctions des fonctions vers les fonctions)	161
$\hat{\circ}$	extension point-à-point de $(\circ)$ aux fonctionnelles	162
ω	fonctionnelle de fermeture par itération ($\omega : f \mapsto f^\omega$)	162
i $(\dot{-})$	fonction de conversion (involution croissante)	55,70
$\overline{f}$	converse d'une fonction ($\overline{f} \triangleq i \circ f \circ i$)	55
$\overleftrightarrow{f}$	symétrisée d'une fonction ($\overleftrightarrow{f} \triangleq f \wedge \overline{f}$)	55

Co-induction

s	une génératrice (une fonction croissante)	45
X_s	ensemble des s-simulations ($X_s \triangleq \{x \in X \mid x \sqsubseteq s(x)\}$)	46
$\boldsymbol{\nu} s$	s-similarité ($\boldsymbol{\nu} s \triangleq \bigvee X_s$)	46
$\rightsquigarrow$	progression	62
$\boldsymbol{\nu}_{\rightsquigarrow}$	$\rightsquigarrow$-similarité	62
$\rightsquigarrow_s$	progression associée à la génératrice s	64
$s_{\rightsquigarrow}$	génératrice associée à la progression $\rightsquigarrow$	65
$\overset{s}{\rightarrowtail}$	progression de compatibilité avec s	161

Bisimulation

$\mathcal{P}$	ensemble générique de processus	
$p, q, r, u, \ldots$	processus (ou états)	
$\mathcal{R}$	ensemble des relations binaires sur $\mathcal{P}$	71
$R, S, \ldots$	relations binaires	71
I	relation identité ($\{\langle p, p\rangle \mid \forall p \in \mathcal{P}\}$)	71
$R \cdot S$	composition relationnelle de R et S	71
$\overline{R}$	converse de la relation R	71
$\mathcal{L}$	ensemble des étiquettes	74
$\alpha, \beta, \ldots$	étiquettes (ou actions)	74
τ	étiquette silencieuse (ou interne)	74
$\mathcal{L}^{\mathrm{v}}$	ensemble des étiquettes visibles ($\mathcal{L}^{\mathrm{v}} \triangleq \mathcal{L} \setminus \{\tau\}$)	74
$a, b, \ldots$	étiquettes visibles	74
$\xrightarrow{\alpha}$	transition selon l'étiquette α	74
$\xrightarrow{\widehat{\alpha}}$	$\xrightarrow{\widehat{\tau}} \triangleq \xrightarrow{\tau}{}^{=}$, $\xrightarrow{\widehat{a}} \triangleq \xrightarrow{a}$	74
$\overset{\alpha}{\Rightarrow}$	$\xrightarrow{\tau}{}^{\star} \cdot \xrightarrow{\alpha} \cdot \xrightarrow{\tau}{}^{\star}$ ($\overset{\tau}{\Rightarrow} = \xrightarrow{\tau}{}^{+}$)	74
$\overset{\widehat{\alpha}}{\Rightarrow}$	transition faible selon α ($\overset{\widehat{a}}{\Rightarrow} = \overset{a}{\Rightarrow}$, $\overset{\widehat{\tau}}{\Rightarrow} = \xrightarrow{\tau}{}^{\star}$)	74

$\overleftarrow{\stackrel{\alpha}{\longrightarrow}}$	converse de $\stackrel{\alpha}{\longrightarrow}$ (idem pour toute relation représentée par une flèche) 75
$\mathbf{s}$	génératrice des simulations fortes 75
$\mathbf{w}$	génératrice des simulations faibles 75
$\mathbf{e}$	génératrice des pré-expansions 75
$\mathbf{p}$	génératrice des simulations progressives 75
$\mathbf{w}_\mathrm{t}$	génératrice des simulations faibles, avec transitivité lors des offres visibles 75
$\rightsquigarrow_{\mathbf{s}} \ldots$	progressions des simulations fortes, etc... 75
$\sim$	bisimilarité forte : $\boldsymbol{\nu}\overleftrightarrow{\mathbf{s}}$ 87
$\approx$	bisimilarité faible : $\boldsymbol{\nu}\overleftrightarrow{\mathbf{w}}$ 87
$\asymp$	expansion : $\boldsymbol{\nu}\,(\mathbf{e} \wedge \overline{\mathbf{w}})$ 87
$\lessapprox$	élaboration : $\boldsymbol{\nu}\,(\mathbf{w} \wedge \overline{\mathbf{p}})$ 87
$\precsim$	bi-expansion : $\boldsymbol{\nu}\overleftrightarrow{\mathbf{e}}$ 87
$\trianglerighteq$	bisimilarité progressive : $\boldsymbol{\nu}\overleftrightarrow{\mathbf{p}}$ 87
$\trianglerighteq$, $\blacktriangleright$	préordres contrôlés 99
$\succ$	relation satisfaisant une hypothèse de terminaison
$\precsim, \gtrapprox, \blacktriangleleft, \prec$	converses des relations $\succsim, \gtrapprox, \blacktriangleright, \succ$

Modulo Contexte

$c, d \ldots$	contextes 165
$C, \ldots$	ensembles de contextes 165
$\mathcal{C}, \ldots$	clôtures 95
$\lfloor c \rfloor$	fonction associée à un contexte c 165
$\lfloor C \rfloor$	fonction associée à un ensemble de contextes C 165
C_{ccs}	contextes de CCS 169
C_m	contextes monadiques de CCS 173
C_i	contextes initiaux de CCS 169
C_i^{nd}	contextes initiaux non dégénérés de CCS 178
$\mathcal{C}_{ccs}$	clôture par les contextes de CCS 169
$\mathcal{C}_{ccs}^{nd}$	clôture par les contextes non dégénérés de CCS 178
$p, q, r ::=$	processus CCS 169
$\mid \mathbf{0}$	processus vide
$\mid p \mid q$	mise en parallèle de p et q
$\mid p + q$	choix non déterministe entre p et q
$\mid \alpha.p$	préfixe séquentiel : α puis p
$\mid\, !p$	réplication de p
$p \parallel q$	entrelacement de p et q 10

Modèle d'exécution pour les processus distribués

$\mathcal{L}$-TS	système de transitions étiquetées par $\mathcal{L}$	184
$\mathcal{H}$	ensemble des localités	184
$h, k, \ldots$	localités	184
$\widetilde{h}$	liste de localités	199
$h; \widetilde{k}$	ajout de la localité h à la liste $\widetilde{k}$	199
$\widetilde{h}; \widetilde{k}$	ajout des localités $\widetilde{h}$ à la liste $\widetilde{k}$	199
$\mathcal{M}$	ensemble des messages	184
$m, n, \ldots$	messages	184
$m; n$	union de deux messages	189
$\mathcal{L}_p$	ensemble des actions des processus	184
$\delta ::=$	actions des processus	184
$\mid \quad h\langle m\rangle$	émission d'un message m vers h	
$\mid \quad (m)$	réception d'un message m	
$\mid \quad \triangleright h$	migration vers h	
$\mid \quad \nu h[p]$	déploiement du processus q dans une nouvelle localité h	
$\overset{\delta}{\hookrightarrow}$	transitions étiquetées des processus	
$\mathcal{DP}$	ensemble des processus distribués	185
D	processus distribués	185
$U \otimes D$	état du $\mathcal{L}$-TS-composite	185
$\overset{\alpha}{\mapsto}$	transitions du $\mathcal{L}$-TS-composite	185
$\mathcal{L}_r$	ensemble des actions du réseau	185
$\mu ::=$	actions du réseau	185
$\mid \quad h\langle m\rangle$	émission d'un message m vers h	
$\mid \quad h(m)$	réception d'un message m en h	
$\mid \quad h \triangleright k$	migration de h vers k	
$\mid \quad \nu h$	création d'une nouvelle localité h	
$U, V ::=$	réseaux	188
$\mid \quad \mathbf{0}$	réseau vide	
$\mid \quad U \mid V$	composition parallèle	
$\mid \quad h[m]$	localité réelle contenant un message m	
$\mid \quad h \triangleright k$	messager de h vers k	
$\mid \quad h\not\triangleright$	messager bloqué en h	
$\mid \quad h\langle m\rangle$	message en attente sur h	
$\mid \quad h\langle m\rangle_{\widetilde{k}}$	message annoté (modèle optimisé)	
$\mid \quad h\langle \triangleright k\rangle$	message de relocalisation	

Bibliographie

[Abr90] Samson Abramsky. The Lazy Lambda Calculus. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison Wesley, 1990.

[AD07] Roberto Amadio and Mehdi Dogguy. Determinacy in a synchronous pi-calculus. Technical report, CNRS: UMR7126, 2007.

[AKH92] S. Arun-Kumar and Matthew Hennessy. An Efficiency Preorder for Processes. *Acta Informatica*, 29(9):737–760, 1992.

[AKN95] S. Arun-Kumar and V. Natarajan. Conformance: A Precongruence Close to Bisimilarity. In *Proc. Struct. in Concurrency Theory*, pages 55–68. Springer Verlag, 1995.

[Bar84] Hengt Barendregt. *The Lambda Calculus, its Syntax and Semantics*. North Holland, 1984.

[BD86] Leo Bachmair and Nachum Dershowitz. Commutation, transformation, and termination. In *Proc. of 8th International Conference on Automated Deduction,*, volume 230 of *LNCS*, pages 5–20. Springer Verlag, 1986.

[BdS96] Frédéric Boussinot and Robert de Simone. The SL synchronous language. *IEEE Transactions on Software Engineering*, 22(4):256–266, 1996.

[BG92] Gerard Berry and Georges Gonthier. The ESTEREL synchronous programming language: design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, November 1992.

[BIM95] Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *Journal of ACM*, 42(1):232–268, 1995.

[Bir40] Garrett Birkhoff. *Lattice Theory*, volume 25 of *Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, 3rd (1967) edition, 1940.

[BK84] Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984.

[BK85] Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[BKvO98] Marc Bezem, Jan Willem Klop, and Vincent van Oostrom. Diagram Techniques for Confluence. *Information and Computation*, 141(2):172–204, 1998.

[BL87] Françoise Bellegarde and Pierre Lescanne. Transformation ordering. In *TAPSOFT, Vol.1*, volume 249 of *LNCS*, pages 69–80. Springer Verlag, 1987.

[BL90] Françoise Bellegarde and Pierre Lescanne. Termination by completion. *Appl. Algebra Eng. Commun. Comput.*, 1:79–96, 1990.

[BS98] Michele Boreale and Davide Sangiorgi. Some Congruence Properties for π-calculus Bisimilarities. *Theoretical Computer Science*, 198:159–176, 1998.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile Ambients. In *Proc. FOSSACS '98*, volume 1378 of *LNCS*, pages 140–155. Springer Verlag, 1998.

[CH84] Thierry Coquand and Gérard P. Huet (projet Logical, INRIA). The Coq proof assistant, 1984. http://coq.inria.fr/.

[CN02] Giuseppe Castagna and Frank Zappa Nardelli. The Seal Calculus Revisited. In *Proc. of FSTTCS '02*, volume 2556 of *LNCS*, pages 85–96. Springer Verlag, 2002.

[Coh00] Ernie Cohen. Separation and reduction. In *in Proc. MPC*, volume 1837 of *LNCS*, pages 45–59. Springer Verlag, 2000.

[CT51] Louise H. Chin and Alfred Tarski. Distributive and modular laws in the arithmetic of relation algebras. *University of California Publications in Mathematics*, 1(9):341–384, 1951.

[DBvdW97] Henk Doornbos, Roland Backhouse, and Jaap van der Woude. A calculational approach to mathematical induction. *Theoretical Computer Science*, 179(1-2):103–135, 1997. Fundamental Study.

[Dil39] Robert P. Dilworth. Non-commutative residuated lattices. *Transactions of the American Mathematical Society*, 46:426–444, 1939.

[DP90] Brian Davey and Hilary Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2nd (2002) edition, 1990.

[dS85] Robert de Simone. Higher-level synchronising devices in meije-sccs. *Theoretical Computer Science*, 37:245–267, 1985.

[FFMS02] Cédric Fournet, Fabrice Le Fessant, Luc Maranget, and Alan Schmitt. JoCaml: A Language for Concurrent Distributed and

Mobile Programming. In *Proc. of Advanced Functional Programming 2002*, volume 2638 of *LNCS*, pages 129–158. Springer Verlag, 2002.

[FG05] Cédric Fournet and Georges Gonthier. A hierarchy of equivalences for asynchronous calculi. *Journal of Algebraic and Logic Programming*, 63(1):131–173, 2005.

[FMJJ92] Jean-Claude Fernandez, Laurent Mounier, Claude Jard, and Thierry Jéron. On-the-fly verification of finite transition systems. *Formal Methods in System Design*, 1(2/3):251–273, 1992.

[Fok00] Wan Fokkink. *Introduction to Process Algebra*. Springer, 2000.

[Fou98] Cédric Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming*. PhD thesis, Ecole Polytechnique, 1998.

[FS90] Peter Freyd and Andre Scedrov. *Categories, Allegories*. North Holland, 1990.

[FvG96] Wan Fokkink and Rob J. van Glabbeek. Ntyft/ntyxt rules reduce to ntree rules. *Information and Computation*, 126(1):1–10, 1996.

[Ges90] Alfons Geser. *Relative Termination*. PhD thesis, Universität Passau, Germany, 1990.

[Giv07] Steven Givant. The calculus of relations as a foundation for mathematics. *Journal of Automated Reasoning*, 37(4):277–322, 2007.

[GP98] Andrew D. Gordon and Andrew M. Pitts. *Higher Order Operational Techniques in Semantics*. Cambridge University Press, 1998.

[GS96] Jan Friso Groote and M. P. A. Sellink. Confluence for process verification. *Theoretical Computer Science*, 170(1-2):47–81, 1996.

[GSV06] Paola Giannini, Davide Sangiorgi, and Andrea Valente. Safe ambients: Abstract machine and distributed implementation. *Science of Computer Programming*, 59(3):209–249, 2006.

[GV92] Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, 1992.

[Hen88] Matthew Hennessy. *Algebraic Theory of Processes*. MIT Press, 1988.

[Hen07] Matthew Hennessy. *A Distributed Pi-Calculus*. Cambridge University Press, 2007.

[Hin64] Roger Hindley. *The Church–Rosser Property and a Result in Combinatory Logic*. PhD thesis, University of Newcastle-upon-Tyne, 1964.

[Hoa69] Charles A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, 1969.

[Hoa78] Charles A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

[Hoa85] Charles A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[How96] Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124:103–112, 1996.

[HP07] Daniel Hirschkoff and Damien Pous. A Distribution Law for CCS and a New Congruence Result for the π-calculus. In *Proc. FOSSACS '07*, volume 4423 of *LNCS*, pages 228–242. Springer Verlag, 2007.

[HPS05] Daniel Hirschkoff, Damien Pous, and Davide Sangiorgi. A Correct Abstract Machine for Safe Ambients. In *Proc. COORD '05*, volume 3454 of *LNCS*, pages 17–32. Springer Verlag, 2005. (This abstract has been extended in [HPS07]).

[HPS07] Daniel Hirschkoff, Damien Pous, and Davide Sangiorgi. An Efficient Abstract Machine for Safe Ambients. *Journal of Algebraic and Logic Programming*, 71(2):114–149, 2007.

[HR04] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Journal of Mathematical Structures in Computer Science*, 14(5):651–684, 2004.

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4):797–821, 1980.

[HY95] Kohei Honda and Nobuka Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[Joh02] Peter Johnstone. *Sketches of an Elephant – Volumes 1/2.* Oxford University Press, 2002.

[JT52] Bjarni Jónsson and Alfred Tarski. Boolean algebras with operators: Part II. *American Journal of Mathematics*, 75:127–162, 1952.

[Kna28] Brownislaw Knaster. Un théorème sur les fonctions d'ensembles. *Annales de la Société Polonaise de Mathématiques*, 6:133–134, 1928.

[KW06] Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In *Proc. 33rd POPL*, pages 141–152. ACM Press, 2006.

[Las98] Søren B. Lassen. Relational Reasoning about Contexts. In Andrew D. Gordon and Andrew M. Pitts, editors, *Higher Order Operational Techniques in Semantics*. Cambridge University Press, 1998.

[Las05] Søren B. Lassen. Eager Normal Form Bisimulation. In *Proc. 20th LICS*, pages 345–354, 2005.

[LS00] Francesca Levi and Davide Sangiorgi. Controlling Interference in Ambients. In *Proc. 27th POPL*, pages 352–368. ACM Press, 2000. An extended abstract appeared in [LS03].

[LS03] Francesca Levi and Davide Sangiorgi. Mobile Safe Ambients. *ACM Transactions on Progr. Lang. and Sys.*, 25(1):1–69, 2003.

[LW85] Xavier Leroy and Pierre Weis (projet Cristal, INRIA). The OCaml programming language, 1985. `http://caml.inria.fr`.

[Lyn50] Roger C. Lyndon. The representation of relational algebras. *The Annals of Mathematics (2)*, 51:707–729, 1950.

[Lyn56] Roger C. Lyndon. The representation of relation algebras, II. *The Annals of Mathematics (2)*, 63:294–307, 1956.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil99] Robin Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[Mon64] J. Donald Monk. On representable relation algebras. *Michigan Mathematics Journal*, 11:207–210, 1964.

[MOZ96] Aart Middeldorp, Hitoshi Ohsaki, and Hans Zantema. Transforming termination by self-labelling. In *In Proc. 13th CADE*, volume 1104 of *LNCS*, pages 373–387. Springer Verlag, 1996.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I/II. *Information and Computation*, 100(1):1–77, 1992.

[MS92a] Robin Milner and Davide Sangiorgi. Barbed Bisimulation. In *Proc. 19th ICALP*, volume 623 of *LNCS*, pages 685–695. Springer Verlag, 1992.

[MS92b] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *19th ICALP*, volume 623 of *LNCS*, pages 685–695. Springer Verlag, 1992.

[MS92c] Ugo Montanari and Vladimiro Sassone. Dynamic Congruence vs. Progressing Bisimulation for CCS. *Fundamenta Informaticae*, 16(1):171–199, 1992.

[Nes96] Uwe Nestmann. *On Determinacy and Nondeterminacy in Concurrent Programming*. PhD thesis, Technische Fakultät, Universität Erlangen, 1996.

[New42] Maxwell H. A. Newman. On theories with a combinatorial definition of « equivalence ». *Annals of Mathematics*, 43(2):223–243, 1942.

[NFP98] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Trans. Software Eng.*, 24(5):315–330, 1998.

[Ore44] Oystein Ore. Galois connexions. *Transactions of the American Mathematical Society*, 55:493–513, 1944.

[Par81] David Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science*, pages 167–183, 1981.

[Pit07] Andrew M. Pitts. Techniques for contextual equivalence in higher-order, typed languages. In *ESOP '07*, volume 4421 of *LNCS*, page 1. Springer Verlag, 2007. Invited talk.

[Pou05a] Damien Pous. GcPan webpage, 2005. http://perso.ens-lyon.fr/damien.pous/gcpan.

[Pou05b] Damien Pous. Up-to Techniques for Weak Bisimulation. In *Proc. 32th ICALP*, volume 3580 of *LNCS*, pages 730–741. Springer Verlag, 2005. A long version of this abstract appeared in [Pou07b].

[Pou05c] Damien Pous. Web appendix of [Pou05b, Pou06b], 2005. http://perso.ens-lyon.fr/damien.pous/upto.

[Pou06a] Damien Pous. On Bisimulation Proofs for the Analysis of Distributed Abstract Machines. In *Proc. TGC '06*, volume 4661 of *LNCS*, pages 150–166. Springer Verlag, 2006. An extended version has been submitted [Pou07c].

[Pou06b] Damien Pous. Weak Bisimulation up to Elaboration. In *Proc. CONCUR '06*, volume 4137 of *LNCS*, pages 390–405. Springer Verlag, 2006.

[Pou07a] Damien Pous. Complete Lattices and Up-to Techniques. In *Proc. APLAS '07*, volume 4807 of *LNCS*, pages 351–366. Springer Verlag, 2007.

[Pou07b] Damien Pous. New Up-to Techniques for Weak Bisimulation. *Theoretical Computer Science*, 380(1-2):164–180, 2007.

[Pou07c] Damien Pous. On Bisimulation Proofs for the Analysis of Distributed Abstract Machines. Technical Report 2007-31, LIP – ENS Lyon, 2007. (submitted, this is an extended version of [Pou06a]).

[Pra92] Vaughan Pratt. Origins of the calculus of binary relations. In *Proc. 7th LICS Conf.*, pages 248–254. IEEE Computer Society Press, 1992.

[PW97] Anna Philippou and David Walker. On confluence in the pi-calculus. In *Proc. 24th ICALP*, pages 314–324. Springer Verlag, 1997.

[Ros70] Barry K. Rosen. Tree-manipulating systems and church–rosser theorems. In *Proc. STOC*, pages 117–127. ACM Press, 1970.

[San98] Davide Sangiorgi. On the Bisimulation Proof Method. *Journal of Mathematical Structures in Computer Science*, 8:447–479, 1998.

[San06] Davide Sangiorgi. Personal communication, 2006.

[SKS07a] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environment Bisimulations for Higher-Order Languages. In *Proc. 22nd LICS*, pages 293–302, 2007.

[SKS07b] Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Logical Bisimulations and Functional Languages. In *Proc. FSEN '07*, volume 4767 of *LNCS*, pages 364–379. Springer Verlag, 2007.

[SM92] Davide Sangiorgi and Robin Milner. The problem of "Weak Bisimulation up to". In *Proc. 3rd CONCUR*, volume 630 of *LNCS*, pages 32–46. Springer Verlag, 1992.

[SS04] Alan Schmitt and Jean-Bernard Stefani. The kell calculus: A family of higher-order distributed process calculi. In *Global Computing*, volume 3267 of *LNCS*, pages 146–178. Springer Verlag, 2004.

[Str06] Georg Struth. Abstract abstract reduction. *Journal of Algebraic and Logic Programming*, 66(2):239–270, 2006.

[SV01] Davide Sangiorgi and Andrea Valente. A Distributed Abstract Machine for Safe Ambients. In *Proc. 28th ICALP*, volume 2076 of *LNCS*, pages 408–420. Springer Verlag, 2001. An extended abstract appeared in [GSV06].

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tar41] Alfred Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73–89, 1941.

[Tar55] Alfred Tarski. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific Journal of Mathematics*, 5(2):285–309, June 1955.

[TeR03] TeReSe. *Term Rewriting Systems*. Cambridge University Press, 2003.

[TG87] Alfred Tarski and Steven Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, 1987.

[US01] Asis Unyapoth and Peter Sewell. Nomadic pict: Correct Communication Infrastructure for Mobile Computation. In *Proc. 28th POPL*, pages 116–127. ACM Press, 2001.

[Ver95] Chris Verhoef. A congruence theorem for structured operational semantics with predicates and negative premises. *Nordic Journal Computing*, 2(2):274–302, 1995.

[vG90] Rob J. van Glabbeek. The linear time – branching time spectrum. In *Proc. 1st CONCUR*, volume 458 of *LNCS*, pages 278–297. Springer Verlag, 1990. An extended version appeared in [vG01].

[vG93] Rob J. van Glabbeek. The linear time – branching time spectrum II: The semantics of sequential systems with silent moves. In *Proc. 4th CONCUR*, volume 715 of *LNCS*, pages 66–81. Springer Verlag, 1993.

[vG01] Rob J. van Glabbeek. The linear time – branching time spectrum I. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North Holland, 2001.

[vGW96] Rob J. van Glabbeek and W. Peter Weijland. Branching time and abstraction in bisimulation semantics. *Journal of ACM*, 43(3):555–600, 1996.

[vO94] Vincent van Oostrom. Confluence by Decreasing Diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994.

[vO05] Vincent van Oostrom. Personal communication, 2005.

[Wal90] David Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990.

Index

Résumé

Bien que les langages de programmation actuels fournissent des niveaux d'abstraction de plus en plus élevés, les programmes définis de nos jours sont de plus en plus délicats à étudier : ils sont distribués, concurrents, interactifs, et bien souvent mobiles. De plus, le rôle parfois critique qu'ils endossent nécessite une analyse de plus en plus fine de leurs propriétés. Nous étudions dans cette thèse des techniques de preuve permettant de faciliter l'étude de tels programmes.

Nous développons tout d'abord une théorie des techniques "modulo" pour la co-induction, dans le cadre abstrait des treillis complets. Cette théorie, qui établit des résultats de modularité génériques, fournit un socle solide pour la suite de la thèse, dédiée aux techniques modulo pour la bisimilarité. Dans le cas dit "faible" ces techniques modulo souffrent de limitations, dues à des phénomènes bien connus en théorie de la réécriture. En utilisant les outils de ce domaine (normalisation forte et inductions bien fondées), nous définissons de nouvelles techniques afin de contourner ces limitations. L'utilité de ces techniques est illustrée par l'étude détaillée d'un algorithme distribué non trivial (il s'agit de l'optimisation d'un environement d'exécution pour des processus distribués), et pour lequel les techniques standard s'avèrent insuffisantes

D'autre part, en appliquant notre théorie des techniques modulo dans un espace de fonctions, nous montrons comment obtenir des techniques de second ordre, qui nous permettent de revisiter les techniques dites "modulo contexte" : nous définissons une méthode qui permet de simplifer grandement l'étude de telles techniques, en la ramenant de façon systématique à une étude de cas élementaire. Nous illustrons cette méthode en l'appliquant dans le cas de CCS.

Mots clefs

Sémantique – Algèbres de processus – Équivalences comportementales – Bisimulation – Techniques modulo – Co-induction – Treillis complets – Algèbres relationnelles – Machines abstraites distribuées.

Abstract

While programming languages tend to give higher abstraction levels to the programmer, the programs that are written nowadays tend to be more complex: these programs are distributed, concurrent, interactive, and often, mobile. Moreover the critical role they may play sometimes requires a really precise analysis of their properties. This dissertation is devoted to the study of proof techniques for the analysis of such programs.

We develop a theory of "up-to" techniques for coinduction, in the abstract setting of complete lattices. This theory contains new and general modularity results; it establishes the grounds for the reminder of the dissertation, where we focus on up-to techniques for bisimilarity. Up-to techniques for *weak* bisimilarity are known to be problematic. We show that these problems are related to similar phenomenons in term rewriting theory, and, using tools from this domain (strong normalisation and well-founded inductions), we develop up-to techniques going beyond existing ones. The benefits of these new techniques are illustrated by applying one of them in order to prove the correctness of a non-trivial distributed algorithm, for which standard techniques are not sufficient to give a satisfactory proof.

Independently, by applying our general theory of up-to techniques in the function space, we show how to obtain second-order techniques, that allow us to revisit the "up to context" techniques: we define a generic method that greatly simplifies the study of such techniques. We illustrate this method by using it to recover up to contexts techniques in the case of CCS.

Keywords

Semantics – Process algebras – Behavioural equivalences – Bisimulation – Up-to techniques – Coinduction – Complete lattices – Relation algebras – Distributed abstract machines.