Modélisation, Simulation et Vérification des Grands Réseaux de Régulation Biologique

Loïc Paulevé

ÉCOLE CENTRALE DE NANTES

ÉCOLE DOCTORALE

SCIENCES ET TECHNOLOGIES
DE L'INFORMATION ET DES MATHEMATIQUES

Année : 2011 N° B.U. :

Thèse de Doctorat

Spécialité : Automatique, Robotique, Traitement du Signal
et Informatique Appliquée

Présentée et soutenue publiquement par :

Loïc Paulevé

le 6 octobre 2011
à l'École Centrale de Nantes

Modélisation, Simulation et Vérification des Grands Réseaux de Régulation Biologique

Jury

Président :	Claude Jard	Professeur de l'École Normale Supérieure de Cachan, Antenne de Bretagne/IRISA
Rapporteurs :	François Fages	Directeur de recherche à l'INRIA Paris-Rocquencourt
	Serge Haddad	Professeur de l'École Normale Supérieure de Cachan/LSV
Examinateurs :	Jérôme Feret	Chargé de recherche à l'INRIA Paris-Rocquencourt/École Normale Supérieure
	Jean-Phillipe Vert	Directeur du Centre de Bio-informatique à Mines ParisTech
	Olivier Roux	Professeur de l'École Centrale de Nantes/IRCCyN
	Morgan Magnin	Maître de conférences à l'École Centrale de Nantes/IRCCyN

Directeur de thèse : **Pr. Olivier Roux**
Laboratoire / composante : IRCCyN / École Centrale de Nantes
Co-encadrant : **Dr. Morgan Magnin**
Laboratoire / composante : IRCCyN / École Centrale de Nantes

N° ED 503 -

Résumé / Summary

Modélisation, Simulation et Vérification des Grands Réseaux de Régulation Biologique

Les Réseaux de Régulation Biologique (RRB) sont communément utilisés en biologie systémique pour modéliser, comprendre et contrôler les dynamiques de production des protéines au sein des cellules. Bien qu'offrant une représentation très abstraite des systèmes biologiques, l'analyse formelle de tels modèles se heurte rapidement à l'explosion combinatoire des comportements engendrés.

Cette thèse traite de la modélisation, de la simulation et de la vérification formelle des grands RRB à travers l'introduction d'un nouveau formalisme : les *Frappes de Processus*. La simplicité de ce formalisme permet notamment l'élaboration d'analyses statiques efficaces des systèmes complexes en général.

Cette thèse aborde en premier lieu le raffinement du temps dans les modèles stochastiques via l'introduction d'un *facteur d'absorption de stochasticité*. Ceci apporte un compromis et une flexibilité entre des spécifications temporelles et stochastiques dans les modélisations hybrides. Une simulation générique (non-markovienne) des calculs de processus est alors proposée et appliquée aux Frappes de Processus.

En outre de l'analyse statique des points fixes des Frappes de Processus, cette thèse développe une interprétation abstraite de ces Frappes de Processus permettant des approximations supérieures et inférieures très efficaces de propriétés d'atteignabilité discrète. Cette analyse permet également de faire émerger des composants requis pour la satisfaction de ces propriétés, guidant ainsi le contrôle du système. D'une complexité théorique limitée, cette approche promet de supporter l'analyse de très grands RRB et constitue une ainsi contribution qui ouvre sur de multiples perspectives.

Mots clés : systèmes complexes ; réseaux de régulation biologique ; modélisation hybride ; simulation non-markovienne ; analyse statique ; vérification formelle ; interprétation abstraite ; contrôle.

Modelling, Simulation and Verification of Large Biological Regulatory Networks

Biological Regulatory Networks (BRN) are widely used in systems biology to model, understand and control the dynamics of production of proteins within cells. While offering a very abstract representation of biological systems, the formal analysis of such models rapidly suffers from the combinatorial explosion of the generated behaviours.

This thesis addresses the modelling, the simulation and the formal checking of large-scale BRNs through the introduction of a new framework: the *Process Hitting*. The simplicity of this formalism notably allows the definition of efficient static analyses of complex systems in general.

Firstly, this thesis focuses on refining time features within stochastic models through the introduction of a so-called *stochasticity absorption factor*. Such a technique brings flexibility and a compromise between time and stochastic specifications within hybrid models. A generic (non-Markovian) simulation of process calculi is then proposed and applied to the Process Hitting.

In addition to the static analysis of the fixed points of Process Hitting, this thesis develops an abstract interpretation of the Process Hitting, resulting in very efficient over- and under-approximations of discrete reachability properties. This analysis also allows the highlighting of components required to satisfy such properties, thus driving a control of the system. Having a limited theoretical complexity, this approach is promising for the analysis of very large BRNs, being then a contribution opening multiple outlooks.

Keywords: complex systems; biological regulatory networks; hybrid modelling; non-Markovian simulation; static analysis; model checking; abstract interpretation; control.

Remerciements

Ces trois ans de doctorat ont été un vrai bonheur, me permettant de m'exprimer et de me développer à la fois professionnellement et personnellement. De nombreuses personnes ont contribué plus ou moins directement à cette réussite, cette section leur est dédiée.

Tout d'abord, je tiens à remercier chaleureusement François Fages et Serge Haddad d'avoir accepté d'être les rapporteurs de ma thèse, ainsi que Claude Jard d'avoir présidé le jury, et Jérôme Feret et Jean-Phillipe Vert d'en avoir été les examinateurs. Tous m'ont apporté de précieux commentaires et conseils, et leur enthousiasme m'encourage pour la suite de mes travaux.

Bien sûr, je remercie vivement mon directeur de thèse, Olivier Roux, et mon co-encadrant, Morgan Magnin. Cette thèse a été pour moi trois ans de liberté inouïe, et c'est en immense partie grâce à la confiance qu'ils m'ont rapidement accordée, leurs grandes disponibilité et écoute, et leurs innombrables conseils, encouragements, et portes qu'ils m'ont ouvertes.

Je remercie également Luc Bougé et Claude Jard de m'avoir conseillé cette thèse à la fin de mon master au sein de l'École Normale Supérieure de Cachan, antenne de Bretagne.

Durant ces trois années, j'ai eu la chance de pouvoir côtoyer bon nombre de chercheurs dans plusieurs pays. En particulier, merci à Andrew Phillips de m'avoir accueilli quelques jours à Microsoft Research à Cambridge, et d'avoir ainsi pu démarrer une collaboration très fructueuse sur la simulation stochastique des modèles biologiques. Merci également à Adrien Richard pour ses très passionnantes discussions autour des Réseaux Booléens. Je suis très fier de partager quelques résultats avec lui, et j'espère qu'ils ne sont que le prélude à de nombreux autres ! Merci encore à Jérôme Feret de m'avoir accueilli quelques jours dans son équipe et permis d'améliorer grandement la formalisation de certains de mes résultats. Merci à Irena Rusu et Claude Jard d'avoir été membres de mon comité de suivi de thèse. Enfin, merci à Damien Eveillard, Jérémie Bourdon et Anne Siegel pour leur intérêt et leurs très intéressantes discussions.

L'ambiance de travail au sein de l'IRCCyN à l'École Centrale de Nantes a toujours été cordiale et très agréable. Ainsi, merci à Michel Malabre pour son accueil au sein de l'IRCCyN. Merci à Virginie Dupont pour sa bonne humeur et de m'avoir facilité bon nombre de démarches administratives. Merci à Didier Lime, Sébastien Faucou, Guillaume Moreau et Jean-Yves Martin pour leurs agréables échanges et conseils sur la recherche et l'enseignement.

Un immense merci à mes collègues thésards (dont une bonne partie ne l'est plus) avec qui j'ai pu perfectionner la cuisine du homard au barbecue, le tennis, le jazz manouche, la dégustation de bières et le débat polémique. Nombre d'entre eux sont devenus des amis. Donc, un grand merci à Tomas, Pedro, Louis-Marie, Sébastien, Carlos, Adrien, Émilie, Didier (bis), Charlotte, Roberto, Denis, Xavier, Dominique, Céline et Maxime, à qui je souhaite une grande réussite pour sa thèse qui commence. Merci également à mes collègues moniteurs avec qui j'ai eu grand plaisir de discuter, en particulier Thomas, Marie, Typhaine, Gaëlle, Clémence et Patrick. Enfin, merci à mes collègues et amis de promotion de Rennes : François, David, Rayan, Vincent (x2), Nicolas et Arnaud.

Je remercie grandement mes parents, ma petite sœur et ma famille toute entière pour leur soutient et encouragements. Merci également à Tal, Zak et Florence pour leur amitié. Et un grand merci à Thomas, Gaëlle et Alix, qui sera, dans quelques années, en âge de lire ce manuscrit.

Enfin, et surtout, je remercie Élodie, non seulement pour son soutient indéfectible durant ces deux dernières années et son aide à la correction de ce manuscrit, mais principalement d'avoir placé cette thèse au-delà d'un accomplissement en soi : une étape importante vers un épanouissement personnel, professionnel, et, je l'espère profondément, familial.

Table des matières

Chapitre **1**

Introduction

1.1 Contexte et Motivations

L'étude et la compréhension des systèmes réels (biologiques compris) repose généralement sur la création de modèles représentant de façon abstraite les différents composants, fonctions et comportements impliqués. La science informatique apporte plusieurs contributions en ce sens :

- des outils théoriques et pratiques pour la modélisation d'un système : langages formels, environnements de développement, etc. ;
- une forte puissance de calcul : calcul distribué, optimisation des algorithmes, etc. ;
- la justesse du calcul effectué : dans l'hypothèse où le calculateur ne présente pas de failles physiques, le résultat d'un calcul est toujours correct (par rapport à sa spécification).
- des méthodes de raisonnement automatique : preuves d'existence ou d'absence de comportements particuliers, mesures quantitatives sur les comportements (temps moyen, probabilités, ...), etc.

Ainsi, l'informatique fournit un fondement formel aux raisonnements effectués lors de l'étude d'un système. Cet apport est d'autant plus important lorsque le système est composé d'une multitude de sous-systèmes interagissant les uns avec les autres, produisant ainsi un comportement global complexe et empêchant un raisonnement humain intuitif intelligible et sans erreur.

Toutefois, cette approche par le calcul se heurte à deux défis importants. D'une part, la complexité du calcul engendré peut s'avérer un obstacle majeur à l'utilisation d'un raisonnement automatique. Notamment, il est prouvé que certains raisonnements sont intrinsèquement complexes, rendant ainsi leur mise en œuvre impossible pour l'analyse de très grands systèmes. D'autre part, une représentation compréhensible et compacte de la preuve calculée automatiquement peut être désirée : plus qu'une intuition sur le cheminement suivi, cela peut augmenter la confiance dans le résultat obtenu.

Les méthodes d'analyses dites « statiques » tentent d'apporter une réponse à ces deux défis. La caractéristique principale d'une analyse statique est sa capacité à dériver des propriétés d'un modèle sans l'exécuter, permettant ainsi de contourner la potentielle explosion combinatoire des comportements décrits. Typiquement, ces analyses produisent des approximations supérieures et inférieures du résultat attendu et peuvent alors se révéler non-concluantes selon le cas traité (réponse « joker »). De nombreuses techniques permettent la mise au point d'une analyse statique, comme l'analyse topologique du modèle, l'analyse du flot de contrôle, l'utilisation de contraintes, ou encore l'interprétation abstraite, pour en nommer quelques-unes. De manière générale, la construction et l'efficacité d'une analyse statique dépend fortement du type de propriété recherché et de la sémantique du langage formel sur lequel l'analyse s'effectue.

Cette thèse s'attelle à la modélisation, à la simulation, et à la vérification formelle des *grands* Réseaux de Régulation Biologique (RRB), et aborde les prémices de leur contrôle.

Étudiés dans le cadre de la biologie des systèmes, les RRB tendent à modéliser l'évolution des composants au sein d'un système biologique en fonction des influences positives et négatives qu'ils exercent entre eux. Ils sont notamment utilisés en biologie cellulaire pour comprendre les dynamiques de production des protéines au sein des cellules, et ont, par exemple, des applications pour la recherche sur le cancer et les thérapies géniques.

Notre contribution principale est le développement d'une analyse statique par interprétation abstraite permettant la décision de certaines propriétés d'atteignabilité. L'analyse développée possède une complexité réduite et ouvre ainsi l'étude des dynamiques de très grands RRB. De plus, notre démarche permet l'extraction de composants *clés* pour la satisfaction des propriétés, suggérant ainsi des cibles permettant d'acquerir le contrôle du système étudié (par une thérapie génique, par exemple).

Notre démarche repose sur la définition d'un nouveau formalisme, les *Frappes de Processus*. Ce formalisme se veut simple et dédié à la modélisation de systèmes complexes de grande taille. En particulier, les Frappes de Processus peuvent modéliser les RRB avec différents niveaux d'abstraction dans leur spécification, apportant ainsi une nouvelle souplesse pour leur étude. La structure simple des Frappes de Processus permet alors de dériver efficacement des propriétés sur les dynamiques engendrées.

Nous abordons également la simulation des RRB via les Frappes de Processus en offrant un compromis entre la spécification de contraintes temporelles et la stochasticité inhérente à de tels systèmes.

La section suivante présente une brève introduction aux RRB sur lesquels nos travaux ont des applications directes. La section 1.3 expose les différentes contributions de cette thèse et la section 1.4 présente le plan général de ce manuscrit de thèse. Enfin, la section 1.5 liste les principales notations mathématiques utilisées dans ce manuscrit.

1.2 Les Réseaux de Régulation Biologique

Les phénomènes de régulation jouent un rôle crucial dans de nombreux systèmes biologiques, tel la production des protéines au sein d'une cellule. La figure 1.1 schématise le phénomène de régulation génique : quand un gène est exprimé, il produit, à travers un processus de transcription et de traduction, une protéine, qui peut avoir un effet d'activation ou d'inhibition sur l'expression d'un autre gène, accélérant ou freinant ainsi la production d'autres protéines (ou sa propre production).

Ces informations qualitatives sur l'influence positive ou négative entre les composants d'un système biologique sont typiquement résumées par un *graphe des interactions* : les nœuds représentent des entités biologiques (gène, ARN, etc.), ou une abstraction de plusieurs entités, et sont reliés par des arcs orientés positifs ou négatifs dénotant respectivement une activation ou une inhibition. La figure 1.2(haut gauche) donne un exemple de graphe des interactions.

Ce graphe des interactions constitue ainsi la spécification la plus abstraite d'un RRB et n'indique pas précisément l'évolution de la concentration des composants en fonction de leurs régulateurs.

Les biologistes et les physiciens modélisent généralement les dynamiques des RRB par des équations différentielles ordinaires décrivant l'évolution des concentrations des protéines (Tyson & Oth-

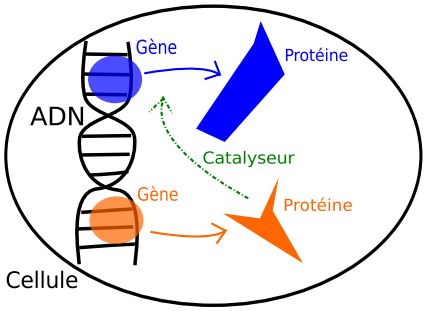


Figure 1.1 – Schéma d'un phénomène de régulation au sein d'une cellule : un gène produit une protéine (à travers un processus biologique complexe) qui a un effet de catalyseur (accélération ou décélération) sur la production d'une autre protéine.

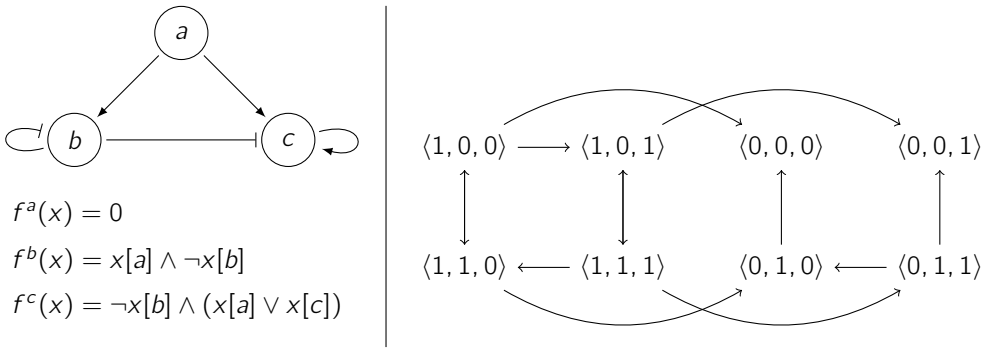


Figure 1.2 – (haut gauche) Exemple de graphe des interactions d'un RRB : un arc se terminant par une flèche (resp. barre) représente une régulation positive (resp. négative). (bas gauche) Exemple de fonctions discrètes (ici booléennes) : x est l'état global du système, $x[a]$ l'état du composant a. (droite) Dynamique discrète obtenue : les triplets sont dans l'ordre a, b, c.

mer, 1978). Toutefois, les constantes employées sont difficiles à estimer, et la résolution de telles équations est très coûteuse, limitant la taille des RRB traitables.

En 1973, le biologiste René Thomas introduit une modélisation dite discrète des RRB (Thomas, 1973) : la concentration des composants est quantifiée en un nombre fini de niveaux discrets ($\{0, 1, 2\}$ par exemple), et l'évolution du niveau d'un composant se spécifie en fonction du niveau de ses régulateurs via les *paramètres discrets de René Thomas* (voir le chapitre 2 pour plus de détails). La figure 1.2(bas gauche) montre un exemple de fonctions discrètes (équivalentes à la spécification des paramètres discrets de René Thomas) ; la dynamique obtenue est représentée dans la figure 1.2(droite).

Afin de reproduire avec plus de précision les dynamiques des RRB, les modélisations hybrides introduisent des dimensions continues (temps, probabilités) régissant les transitions entre les états discrets du RRB. Ces modélisations mettent en avant des aspects quantitatifs sur l'apparition de certains comportements et servent à apprécier l'importance de certains paramètres temporels ou stochastiques sur l'évolution du système.

Nous notons finalement que certains modèles biologiques peuvent être abstraits par des RRB, comme par exemple les systèmes de réactions biochimiques (Fages & Soliman, 2008*a*).

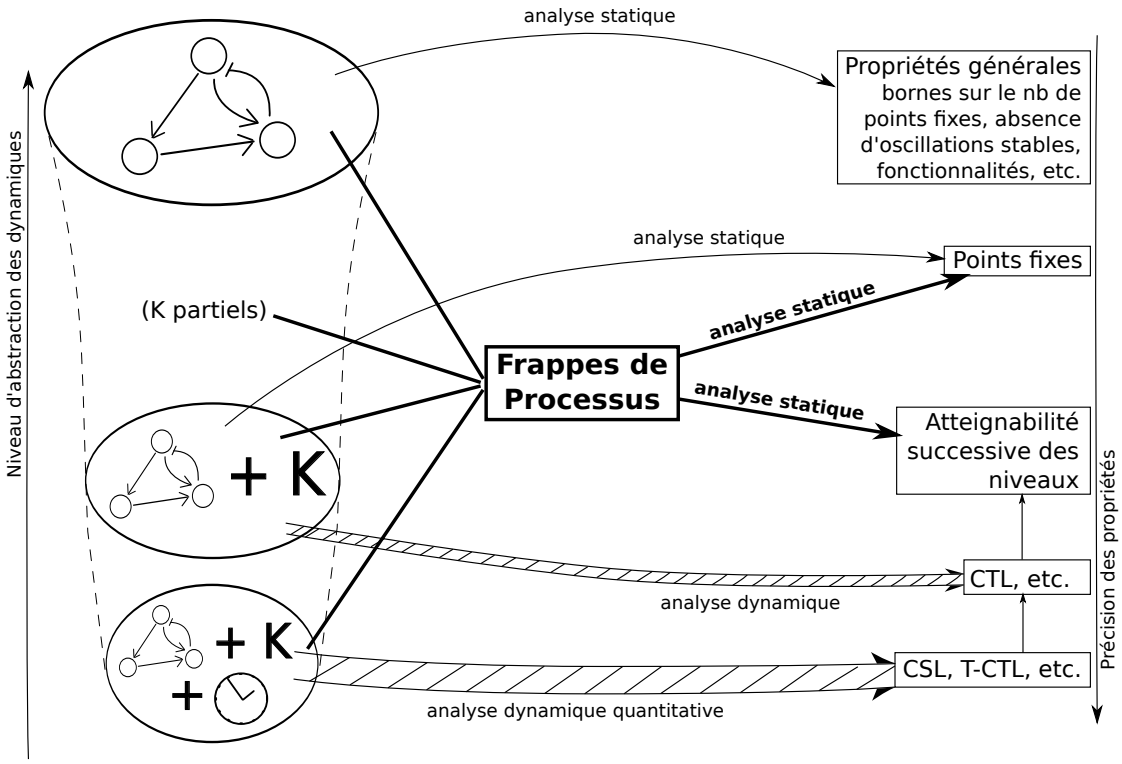


Figure 1.3 – Placement des Frappes de Processus par rapport aux analyses existantes des RRB. La largeur des flèches représente la complexité de l'analyse associée ; les modélisations et analyses liées aux Frappes de Processus apparaissent en gras.

1.3 Contributions

Cette thèse apporte trois contributions principales :

- la modélisation des RRB en Frappes de Processus, formalisme introduit dans cette thèse, permettant notamment la spécification partielle des fonctions d'interaction entre les composants ;
- l'introduction conjointe du temps et de l'aléatoire dans les modélisations hybrides via la définition d'un facteur d'absorption de stochasticité contrôlant la variance temporelle d'une action et via la définition d'une simulation générique des processus markoviens et non-markoviens ;
- la vérification formelle de propriétés discrètes par analyse statique des Frappes de Processus, à savoir la caractérisation des points fixes des dynamiques et des approximations supérieures et inférieures de propriétés d'atteignabilité.

La figure 1.3 place les Frappes de Processus par rapport aux analyses existantes des RRB. Pour différents niveaux d'abstraction dans la spécification d'un RRB, différentes analyses statiques permettent d'extraire efficacement certaines propriétés générales sur les dynamiques engendrées (voir notamment la thèse d'Adrien Richard (Richard, 2006)). Les analyses dites dynamiques explorent les comportements possibles du système afin de vérifier des propriétés précises sur des dynamiques engendrées, mais pour un coût souvent très élevé, limitant la taille des modèles traitables. Les analyses développées dans cette thèse ouvrent une nouvelle piste pour la dérivation statique de propriétés dynamiques précises dans les grands RRB.

Nous pouvons décliner les apports de cette thèse, dont les contributions principales précitées, selon six points : la définition du formalisme des Frappes de Processus ; le raffinement temporel des modèles en π-calcul stochastique ; la simulation générique des calculs de processus markoviens et non-markoviens ; l'analyse statique des points fixes des Frappes de Processus ; l'analyse statique à très grande échelle des propriétés d'atteignabilité au sein des Frappes de Processus ; et le développement du logiciel PINT.

Définition du formalisme des Frappes de Processus. Dédiées à la modélisation de grands systèmes complexes, les Frappes de Processus engendrent des modèles possédant une structure simple dont nous prenons avantage pour le développement d'analyses statiques efficaces. Les Frappes de Processus regroupent un nombre fini de processus séparés en un ensemble de *sortes*. À tout instant, un et un seul processus de chaque sorte est présent ; un processus peut être remplacé par un autre processus de même sorte par la *frappe* d'un autre processus présent. Particulièrement adaptées à la modélisation des RRB, les Frappes de Processus permettent une construction simple de la **dynamique généralisée des RRB**, c'est-à-dire la dynamique la plus large possible spécifiée par un graphe des interactions. Ensuite, par un processus de raffinement, il est possible de construire en Frappes de Processus une **modélisation des RRB spécifiés partiellement, ou totalement**. Ce procédé permet ainsi l'incorporation progressive de contraintes sur les paramètres discrets d'un RRB.

Raffinement temporel des modèles en π-calcul stochastique. Dans le but d'obtenir une dimension temporelle précise dans les modèles stochastiques, nous présentons une technique de modélisation introduisant un *facteur d'absorption de stochasticité* afin de réduire la variance temporelle d'une action spécifiée par un taux d'utilisation. Nous fournissons alors des outils permettant de traduire un intervalle de temps en paramètres stochastiques (couple d'un taux d'utilisation et d'un facteur d'absorption). Nous montrons également une construction du facteur d'absorption dans le π-calcul stochastique markovien et en PRISM, un vérificateur formel de modèles probabilistes. Appliquée aux Frappes de Processus, notre technique permet l'introduction conjointe de paramètres temporels et stochastiques, dimensions toutes deux nécessaires à la modélisation précise des RRB.

Simulation générique des calculs de processus markoviens et non-markoviens. Afin de faciliter la mise en œuvre d'un simulateur pour les calculs de processus, nous définissons une machine abstraite générique pour la simulation markovienne et non-markovienne. En utilisant le principe de la « compilation à la volée », notre machine peut être instanciée pour différentes méthodes de simulation et avec de nombreux formalismes en spécifiant seulement certaines fonctions propres à leur sémantique. Nous présentons une instanciation de méthodes de simulation markoviennes et non-markoviennes ; ainsi que pour le π-calcul stochastique, le calcul bioambient et le calcul κ. Une preuve générique de la correction de notre machine est également démontrée. Ce travail a été effectué en collaboration avec Andrew Phillips et Matthew Lakin (Microsoft Research, Cambridge, UK). Nous instancions alors cette machine abstraite pour la simulation des Frappes de Processus munis de paramètres stochastiques et temporels.

Analyse statique des points fixes des Frappes de Processus. Nous démontrons une caractérisation complète des points fixes des dynamiques engendrées par des Frappes de Processus via une analyse topologique du modèle. L'énumération des points fixes est réduite à l'énumération de

n-cliques dans un graphe n-parti, résultant d'une représentation complémentaire des Frappes de Processus. Appliqué aux modélisations de la dynamique généralisée des RRB, ce résultat a débouché sur la **caractérisation des points fixes topologiques des RRB**, en collaboration avec Adrien Richard (I3S & CNRS, Sophia Antipolis, France). Ceci complète les analyses statiques existantes sur les graphes des interactions en extrayant du graphe ces points fixes ne dépendant que de la topologie du graphe et non des fonctions précises de la dynamique.

Analyse statique à très grande échelle des propriétés d'atteignabilité au sein des Frappes de Processus. En exploitant la structure particulière des Frappes de Processus, nous définissons une analyse statique par interprétation abstraite permettant de approximer supérieurement ou inférieurement des propriétés d'atteignabilité successive de processus. En combinant deux abstractions complémentaires des Frappes de Processus, une propriété d'atteignabilité donnée peut être récursivement et itérativement raffinée jusqu'à l'obtention potentielle d'une décision positive ou négative sur sa faisabilité. Dans l'état actuel, notre méthode peut également être non-concluante. La complexité théorique de cette analyse est limitée, promettant son applicabilité à l'analyse de très grandes Frappes de Processus et donc de très grands RRB. Expérimentées sur des Frappes de Processus modélisant des RRB regroupant plus de cent composants, les décisions se calculent en quelques centièmes de seconde, alors que les méthodes classiques de vérification formelle échouent face à l'explosion combinatoire des dynamiques sous-jacentes. De plus, notre approche permet l'extraction des processus nécessaires à la satisfaction de la propriété d'atteignabilité donnée, information pouvant **amener au contrôle de l'atteignabilité dans le système réel** modélisé.

Développement du logiciel PINT. Programmé en OCaml, le logiciel PINT (`http://process.hitting.free.fr`) implémente les résultats de cette thèse : spécification textuelle des Frappes de Processus, simulation et analyses statiques. Structuré autour d'une bibliothèque utilisable par d'autres logiciels, PINT aspire à être la base des futurs développements autour des Frappes de Processus.

1.4 Organisation du Manuscrit

La suite de ce manuscrit est structurée de la façon suivante :

Le chapitre 2 résume l'état de l'art des principales techniques de modélisation des RRB ainsi que des principales analyses statiques développées.

La partie I introduit le nouveau formalisme des Frappes de Processus : le chapitre 3 établit la définition des Frappes de Processus et détaille leur application à la modélisation des RRB (de la dynamique généralisée au raffinement par la construction des fonctions entre les composants) ; le chapitre 4 discute de l'expressivité des Frappes en Processus en proposant une traduction vers les Automates Finis Communiquants, le π-Calcul et les Réseaux de Petri ; puis en étudiant la traduction des Réseaux Discrets (également utilisés pour la modélisation des RRB) et des Automates Finis Communicants en Frappes de Processus avec Priorités.

La partie II étudie l'introduction conjointe de paramètres temporels et stochastiques dans les modèles en général : le chapitre 5 présente le facteur d'absorption de stochasticité afin de réduire la variance temporelle dans le cadre général du π-calcul stochastique. La construction de ce facteur en π-calcul stochastique markovien et en PRISM est également montrée, et des estimateurs permettant

de traduire un intervalle de temps en paramètres stochastiques sont fournis. Le chapitre 6 définit une machine abstraite permettant la simulation générique markovienne et non-markovienne des calculs de processus, réduisant ainsi la difficulté d'implémenter un simulateur pour un calcul donné (travail effectué en collaboration avec Andrew Phillips et Matthew Lakin, Microsoft Research, Cambridge, UK). Enfin le chapitre 7 applique ces résultats aux Frappes de Processus en montrant notamment une traduction vers PRISM afin de vérifier des propriétés quantitatives sur les modèles.

La partie III présente les analyses statiques développées sur les Frappes de Processus : le chapitre 8 commence par la caractérisation des points fixes topologiques des RRB (travail effectué en collaboration avec Adrien Richard, I3S & CNRS, Sophia Antipolis, France), puis montre la caractérisation des points fixes des Frappes de Processus via une analyse structurelle. Le chapitre 9 établit une analyse statique efficace par interprétation abstraite des Frappes de Processus afin de approximer supérieurement et inférieurement des propriétés d'atteignabilité. C'est une contribution majeure de cette thèse.

La partie IV traite de la mise en œuvre des résultats apportés dans les parties I, II et III pour la modélisation, simulation et vérification de cas réels de RRB : le chapitre 10 présente le logiciel PINT développé au cours de cette thèse et implémentant les résultats sur les Frappes des Processus ; le chapitre 11 applique les méthodes développées aux RRB, et démontre notamment l'efficacité des analyses statiques pour les propriétés d'atteignabilité présentées dans le chapitre 9 pour des RRB regroupant plus d'une centaine de composants.

La partie V conclut cette thèse en détaillant quelques ouvertures principales (chapitre 12) et en discutant des contributions apportées (chapitre 13).

L'appendice A fournit l'instanciation et la correction de la machine abstraite générique développée dans le chapitre 6 pour le calcul bioambiant et κ.

L'appendice B contient les codes sources PINT complets des applications étudiées dans le chapitre 11.

Aide à la lecture. Les parties II et III peuvent être lues de manière indépendante. Chaque partie et chaque chapitre est préfixé par un court texte résumant son contenu.

1.5 Notations Mathématiques

Nous définissons ici les notations principales utilisées tout au long de ce manuscript :

Connecteurs logiques $\wedge$ (conjonction), $\vee$ (disjonction), $\neg$ (négation).

Ensembles Étant donné un *ensemble* fini $S = \{e_1, \ldots, e_n\}$, $|S| = n$ est sa cardinalité ; $\wp(S)$ est l'ensemble des parties de S.
$[i; j] = \{i, i+1, \ldots, j\}$ avec i, j des entiers naturels ; $[i; j] = \emptyset$ si $j < i$.
$\{x \in S \mid P\}$ est l'ensemble des éléments de S satisfaisant le prédicat P.
Nous écrivons $\widetilde{O}$ pour dénoter un ensemble fini $\{O_1, .., O_N\}$.

Multi-ensembles $[e_1, e_1, \ldots, e_n]$ dénote un multi-ensemble (les éléments peuvent avoir une multiplicité supérieure à 1). L'union de deux multi-ensembles M et M' se note $M \uplus M'$, et leur différence $M \uplus M'$.
Nous écrivons $\overline{I}$ pour dénoter un multi-ensemble fini $[I_1, .., I_N]$. Nous autorisons également l'écriture d'un multi-ensemble $\overline{I}$ comme un ensemble de paires $\{(I_1, i_1), .., (I_M, i_M)\}$, où chaque paire (I, i) dénote un élément I et sa multiplicité i correspondante.

Associations Nous écrivons une association finie comme $R = \{O_1 \mapsto A_1, \ldots, O_N \mapsto A_N\}$;
$\mathsf{dom}(R)$ est le domaine de l'association R et $R(O_i)$ l'élément associé à O_i ;
$R\{O_i \mapsto A'_i\}$ dénote l'association R modifiée par l'association de O_i à A'_i ; si R' est une association, nous écrivons $R\{R'\}$ pour R modifiée par toutes les associations décrites dans R' (superposition).

Séquences Étant donnée une *séquence* finie d'éléments $A = e_1 :: \ldots :: e_n$, $|A| = n$ est la longueur de la séquence ; $\mathbb{I}^A = [1; |A|]$ est l'ensemble des indices de A ; et $A_i = e_i, \forall i \in \mathbb{I}^A$.
La séquence vide se note ε et $A_{i..j}$ est la sous-séquence $A_i :: \ldots :: A_j$; $A_{i..j} = \varepsilon$ si $j < i$.

$\mathsf{pppf}\{x_0\}\,(x \mapsto x')$ est le plus petit point fixe, si il existe, de $f(x) = x'$ appliquée initialement sur x_0.

Chapitre **2**

Modélisations et Analyses des Réseaux de Régulation Biologique

Ce chapitre effectue un tour d'horizon des différentes techniques de modélisation et de vérification des Réseaux de Régulation Biologique (RRB). Nous nous concentrons sur trois niveaux d'abstraction courants des RRB : le graphe des interactions, les modèles discrets et les modèles hybrides. Nous détaillons ensuite les différentes analyses des RRB en nous focalisant principalement sur les analyses statiques : analyses topologiques du graphe des interactions, représentations symboliques des fonctions discrètes et réductions de modèles.

2.1 Introduction

Les Réseaux de Régulation Biologique (RRB) abstraient les systèmes biologiques en ne considérant que les effets régulateurs entre les composants. Nous parlons de régulation positive (resp. négative) quand la présence soutenue d'un composant a participe à augmenter (resp. diminuer) la présence d'un composant b ; a est alors un activateur (resp. inhibiteur) de b.

L'étude des régulations entre les composants d'un système biologique aide à comprendre les dynamiques de production de ces différents composants. Dans le cadre de la biologie cellulaire, par exemple, certaines protéines ont un effet régulateur sur la production d'autres protéines (via l'inhibition ou l'activation de l'expression d'un gène, par exemple). Or le comportement d'une cellule dépend fortement de la concentration de certaines protéines, commandant alors, par exemple, une division ou l'exécution d'une certaine fonction. Ainsi, l'analyse des RRB permet de comprendre et prédire le comportement d'une cellule en se concentrant sur les dynamiques de production des protéines inter-régulées.

Ce chapitre ne détaillera pas les modélisations des processus biologiques à l'échelle moléculaire, spécifiant le comportement précis de chaque élément biologique (protéine, plasmide, etc.). Bien que permettant de simuler des comportements biologiques très détaillés, ces approches requièrent des langages formels très évolués, rendant plus complexe leur analyse formelle. Parmi ces méthodes, nous pouvons citer les modélisations utilisant le langage κ (Danos, Feret, Fontana & Krivine, 2008), le π-calul stochastique (Kuttler & Niehren, 2006), le *Beta Workbench* (Dematte, Priami & Romanel, 2008*b*), Bio-PEPA (Ciocchetta & Hillston, 2009) ou encore DSD (Phillips & Cardelli, 2009), pour en nommer quelques-uns.

Les modélisations des RRB se veulent être des abstractions importantes des systèmes biologiques : elles mettent en jeu un « processus » par espèce biologique (un gène, type de protéine, une abstraction d'un composant biologique) au lieu d'un processus par élément biologique ; de plus, l'évolution d'un processus est gouvernée uniquement par l'état de ses régulateurs. Par la suite, nous dénotons une espèce biologique par le terme générique de composant.

La section 2.2 présente trois niveaux de spécification des RRB couramment utilisés, correspondant à différents degrés d'abstraction du système étudié :

1. Le graphe des interactions peut être considéré comme la spécification la plus abstraite d'un RRB : seules les informations qualitatives de régulation entre les composants sont référencées (a est un activateur de b, par exemple).
2. Les modélisations discrètes affectent à chaque composant un ensemble fini discret de niveaux qualitatifs ($\{0, 1, 2\}$, par exemple), reflétant une quantification de sa concentration réelle, et spécifient l'évolution des composants en fonction de leurs régulateurs, cette dernière pouvant être définie par les *paramètres discrets de René Thomas*.
3. Enfin, les modélisations hybrides ajoutent une dimension continue gouvernant les transitions discrètes du système, généralement par l'attribution de délais (aléatoires ou pris dans un intervalle de temps) à l'application de régulations.

Les modélisations continues (par exemple, par les équations différentielles ordinaires) ne sont pas détaillées dans ce chapitre ; leur lien avec les modélisations sus-citées est étudié notamment dans (de Jong, 2002; Richard, Comet & Bernot, 2006).

Après un bref préambule sur les méthodes d'analyse dynamique, la section 2.3 dresse un état de l'art des analyses statiques opérant sur les RRB. Trois familles d'analyses ressortent :

- Les analyses topologiques du graphe des interactions, permettant la dérivation de caractéristiques globales de la dynamique (bornes sur le nombre de points fixes, possibilité d'observer des oscillations soutenues, etc.)
- Les manipulations algébriques sur des représentations symboliques des RRB, permettant notamment une caractérisation efficace des points fixes.
- Les réductions de modèles, permettant la suppression de certains composants tout en conservant un comportement similaire au modèle original.

Enfin, la section 2.4 établit un bilan de ces méthodes de modélisation et d'analyse des RRB.

2.2 État de l'Art des Modélisations Discrètes des RRB

Cette section décrit les principales méthodes de modélisation discrète des RRB, en partant du graphe des interactions jusqu'à un aperçu des modélisations hybrides existantes.

2.2.1 Graphe des Interactions

Le graphe des interactions d'un RRB se veut une représentation simple et qualitative des régulations entre les composants : chaque composant est représenté par un nœud ; un arc orienté positif (resp. négatif) de a vers b dénote que a est un activateur (resp. inhibiteur) de b. Dans le cadre des modélisations discrètes, où chaque composant possède un nombre fini de niveaux ($\{0, 1, 2\}$, par exemple), ce graphe des interactions peut être agrémenté d'informations supplémentaires (si connues) afin de mieux visualiser le rôle des régulations impliquées.

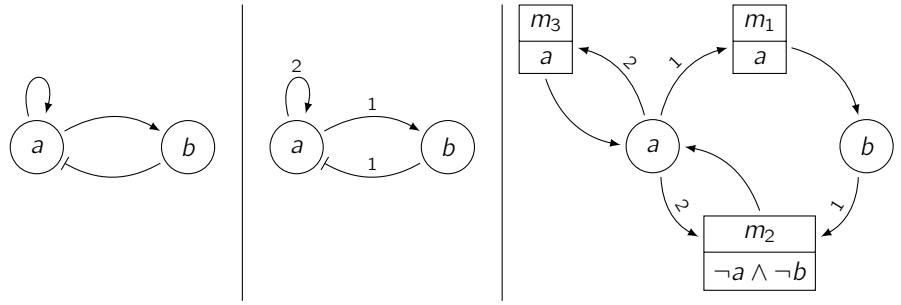


Figure 2.1 – Différents niveaux de représentation du graphe des interactions d'un RRB : (gauche) graphe simple ; (milieu) graphe avec seuils ; (droite) graphe avec multiplexes.

La figure 2.1 résume ces différents niveaux de spécification d'un RRB. Un arc positif se termine par une flèche alors qu'un arc négatif se termine par une barre. Nous détaillons le reste des formalismes graphiques dans la suite de cette sous-section.

Graphe des interactions avec seuils. L'activation (ou l'inhibition) effective de b par a dépend généralement du niveau de a (sa concentration, par exemple). Lorsque ce seuil n'est pas atteint, la régulation peut être considérée soit inoffensive, soit inversée (a devient un inhibiteur de b).

Dans le cadre des modélisations discrètes des RRB, nous affectons à chaque composant un nombre fini de niveaux discrets en partant de 0 ; ainsi, nous pouvons libeller une régulation dans le graphe des interactions par le niveau minimal du régulateur à partir duquel l'influence est effective. Bernot, Cassez, Comet, Delaplace, Müller & Roux (2007) proposent une généralisation de cette spécification en notant explicitement sur les arcs les niveaux du régulateur pour lesquels la régulation est effective et les niveaux pour lesquels elle est inversée (les niveaux non référencés étant considérés comme sans effet sur la cible). Toutefois, dans un souci de simplicité, nous ne considérons dans la suite de cette thèse que les graphes des interactions avec une valeur seuil.

La figure 2.1(milieu) présente un exemple de graphe des interactions avec seuils (libellés sur les arcs).

Graphe des interactions avec multiplexes. Dans les RRB, de nombreuses régulations s'effectuent en *coopération* : si a et c sont des activeurs de b, l'activation peut n'être réellement effective que si à la fois a et c sont présents. Dans le cas de certains systèmes biologiques, cette coopération est le fruit de la formation d'un complexe entre les composants (a s'associe avec c et le complexe $a+c$ active b). De manière plus abstraite, les coopérations peuvent également s'effectuer entre des composants présents et absents.

Lorsqu'elles sont connues, ces coopérations peuvent être spécifiées dans le graphe des interactions via l'introduction de multiplexes (Bernot, Comet & Khalis, 2008), facilitant alors la compréhension du système à la simple lecture du graphe. De par l'ajout de ces précisions dans le modèle, les multiplexes raffinent les dynamiques spécifiées par un simple graphe des interactions, et réduisent le champ de recherche des paramètres discrets permettant de spécifier complètement le comportement du système (voir la sous-section suivante).

La figure 2.1(droite) montre un exemple de graphe des interactions avec multiplexes, représentés par un rectangle divisé en deux parties : la partie haute contient le nom du multiplexe (pouvant alors

être utilisé dans des multiplexes connexes) ; la partie basse spécifie l'expression booléenne de son activation. Une expression booléenne contient les variables correspondant aux composants parents du multiplexe : a est vrai si et seulement si le niveau de composant a est supérieur ou égal au seuil de l'arc entrant sur le multiplexe. Ainsi, dans le cas de m_2, $\neg a \wedge \neg b$ est vraie si et seulement si le niveau de a est inférieur à 2 et le niveau de b inférieur à 1 (c.-à-d., 0).

2.2.2 Modélisations Discrètes : Réseaux Booléens et Réseaux Discrets

Les modélisations discrètes affectent à chaque composant un ensemble fini et dénombrable de niveaux qualitatifs, correspondant, par exemple à une quantification d'une concentration : deux niveaux ($\{0,1\}$) pour les Réseaux Booléens (Thomas, 1973) ; ou un nombre fini de niveaux ($\{0,1,2\}$, par exemple) pour les Réseaux Discrets (Bernot et coll., 2007; Richard et coll., 2006). Dans le cas des Réseaux Discrets, le nombre de niveaux qualitatifs d'un composant est généralement borné par le nombre de régulations sortantes de ce composant dans le graphe des interactions.

Ainsi, l'état du système à un instant donné, que nous notons x, est défini par le niveau courant de chaque composant. Nous écrivons $x[a]$ pour le niveau du composant a dans l'état global x.

La dynamique de chaque composant se décrit généralement suivant deux méthodes équivalentes : par la notion de *ressource* et de *point focal* (paramètre discret de René Thomas) en utilisant le graphe des interactions, ou par la définition directe des fonctions discrètes des composants. Nous détaillons ici l'obtention des fonctions discrètes par les paramètres de René Thomas.

Étant donné un état x du Réseau (Booléen ou Discret), les *ressources* d'un composant a listent ses activateurs et inhibiteurs effectifs, et sont obtenues via une analyse du graphe des interactions avec seuils : un composant b est un activateur (resp. inhibiteur) effectif de a dans l'état x si, soit b régule positivement (resp. négativement) a avec un seuil t et $x[b] \geq t$, soit b régule négativement (resp. positivement) a avec un seuil t' et $x[b] < t'$.

Un *paramètre discret de René Thomas* associe alors à un composant a le niveau vers lequel il tend en présence de ses ressources positives R^+ et négatives R^-. Nous notons K_{a,R^+,R^-} un tel paramètre. Ainsi, pour chaque composant a, 2^k paramètres sont à définir, où k est le nombre de régulateurs de a.

Enfin, pour chaque composant a, nous définissons une fonction discrète f^a associant à un état global x le prochain niveau du composant a. Usuellement (dans le cas non booléen), ce niveau n'est pas directement la valeur du paramètre de René Thomas correspondant, mais la valeur suivante allant en direction de ce point focal :

$$f^a(x) = x[a] + \begin{cases} 1 & \text{si } K_{a,R_a^+(x),R_a^-(x)} > x[a] \\ -1 & \text{si } K_{a,R_a^+(x),R_a^-(x)} < x[a] \\ 0 & \text{sinon,} \end{cases}$$

où $R_a^+(x)$ (resp. $R_a^-(x)$) sont les ressources positives (resp. négatives) de a dans l'état x.

Une dynamique globale peut alors être obtenue en choisissant un ordonnancement des mises à jour à appliquer, allant typiquement de la dynamique dite synchrone (tous les composants sont mis à jour en même temps, équation (2.1)) à la dynamique asynchrone (un seul composant choisi de manière indéterministe est mis à jour, équation (2.2)), qui est la plus usitée. Des dynamiques intermédiaires sont bien entendu possibles (voir par exemple Siebert & Bockmayr, 2006; Aracena,

$K_{a,\emptyset,\{a,b\}} = 0$ $\quad$ $K_{b,\emptyset,\{a\}} = 0$
$K_{a,\{a\},\{b\}} = 2$ $\quad$ $K_{b,\{a\},\emptyset} = 1$
$K_{a,\{b\},\{a\}} = 2$
$K_{a,\{a,b\},\emptyset} = 2$

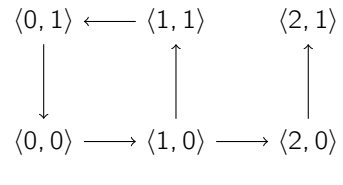


Figure 2.2 – Exemple de paramètres discrets de René Thomas (gauche) pour le graphe des interactions avec seuil de la figure 2.1(milieu), avec la dynamique asynchrone obtenue (droite), où les composantes sont dans l'ordre a, b.

Goles, Moreira & Salinas, 2009).

$$F(x) = \langle f^{a^1}(x), \ldots, f^{a^n}(x) \rangle \tag{2.1}$$
$$F(x) = \langle x[a^1], \ldots, x[a^{m-1}], f^{a^m}(x), x[a^{m+1}], \ldots, x[a^n] \rangle \tag{2.2}$$

Partant du graphe des interactions avec seuils de la figure 2.1(milieu), la figure 2.2 montre un exemple de paramètres de René Thomas avec la dynamique (asynchrone) obtenue.

Étant données les fonctions discrètes des composants, nous pouvons inférer le graphe des interactions (sans seuil) sous-jacent : un arc positif (resp. négatif) est ajouté du composant b vers a si il existe deux états x, x' tels que $x[b] < x'[b]$ et pour tout composant $c \neq b$, $x[c] = x'[c]$, et tel que $f^a(x) < f^a(x')$ (resp. $f^a(x) > f^a(x')$) ; autrement dit, b régule positivement (resp. négativement) a si la seule variation positive de b entraîne, pour au moins un état, l'augmentation (resp. diminution) du niveau de a. Il est à noter qu'en l'absence de contraintes sur les paramètres discrets le graphe ainsi obtenu peut être radicalement différent de celui de départ. Des contraintes garantissant une cohérence avec le graphe des interactions initial sont discutées dans (Bernot et coll., 2007).

Enfin, afin de profiter des nombreuses méthodes d'analyses existantes, Chaouiya, Remy & Thieffry (2008) proposent également un encodage exact des Réseaux Discrets en Réseaux de Petri. La thèse de doctorat de Banks (2009) étudie également différentes modélisations et analyses qualitatives des RRB via les Réseaux de Petri.

2.2.3 Modélisations Hybrides

Dans le but d'obtenir des propriétés quantitatives sur les modèles de RRB (probabilités d'observer un certain comportement, temps moyen, etc.), les modélisations hybrides ajoutent une composante continue gouvernant les transitions entre les états discrets.

Ces spécifications hybrides sont particulièrement adaptées à la modélisation des RRB, systèmes intrinsèquement aléatoires, et où la notion de délai peut jouer un rôle majeur dans la détermination d'un comportement donné. Ainsi, deux types de modélisations hybrides se dessinent majoritairement : les modélisations ajoutant une composante stochastique et celles ajoutant une dimension strictement temporelle.

Les modèles stochastiques affectent aux transitions un délai sous la forme d'une variable aléatoire, généralement de distribution exponentielle, garantissant un système markovien. Dans le cadre des modélisations des RRB, nous pouvons citer l'utilisation de Réseaux de Petri stochastiques (Heiner, Gilbert & Donaldson, 2008), du π-calcul stochastique (Maurin, Magnin & Roux, 2009),

de κ (Danos, Feret, Fontana & Krivine, 2007), ou encore de Biocham (Rizk, Batt, Fages & Soliman, 2008).

Dans le cadre des modélisations de haut niveau (au sens abstrait) des RRB sur lesquelles cette thèse se concentre, l'utilisation de modélisations stochastiques markoviennes porte principalement un intérêt pour le calcul de probabilités d'observation de certains comportements. En effet, une transition dans un Réseau Discret peut être vue comme abstrayant une multitude de réactions faisant augmenter la concentration d'un composant jusqu'à lui faire franchir un niveau qualitatif. Ainsi, un délai suivant une variable exponentielle est peu pertinent avec une telle hypothèse, contrairement à la notion de probabilité de transition.

Les modèles temporels se focalisent, eux, sur des délais pris généralement dans un intervalle de temps fixé ou suivant une certaine équation différentielle. Nous pouvons citer l'utilisation de Réseaux de Petri temporisés (Popova-Zeugmann, Heiner & Koch, 2005), d'automates temporisés (Siebert & Bockmayr, 2006), d'automates hybrides linéaires (Ahmad, Bernot, Comet, Lime & Roux, 2006) ou encore d'automates hybrides non-linéaires (Alur, Belta, Kumar, Mintz, Pappas, Rubin & Schug, 2002). De telles modélisations apportent un raffinement très fort sur les dynamiques discrètes de départ : certaines transitions deviennent impossibles du fait des contraintes temporelles, supprimant bon nombre de comportements, comme démontré par exemple dans (Ahmad, Bourdon, Eveillard, Fromentin, Roux & Sinoquet, 2009).

2.3 État de l'Art des Vérifications Formelles des RRB

La vérification formelle de modèles apporte des preuves sur l'absence, la présence ou la quantification de comportements particuliers afin de valider ou réfuter un modèle donné (en le confrontant à des données expérimentales, par exemple), mais également de prédire certaines dynamiques et ainsi aider à la formulation d'hypothèses quant à la compréhension du système étudié.

Comme sous-entendu dans la section précédente, les modèles discrets des RRB utilisent des formalismes classiques utilisés dans d'autres domaines de l'informatique (Réseaux de Petri, automates temporisés, hybrides, etc.), profitant alors de toutes les avancées en terme de vérification formelle pour l'analyse des modèles biologiques.

La plupart des modèles des systèmes biologiques possèdent cependant une caractéristique qui les différencie de nombreux autres champs d'application de l'informatique : ils regroupent de très nombreuses entités qui, tout en possédant indépendamment un comportement simple, produisent via leurs interactions un comportement global complexe. Ainsi, les méthodes « classiques » de vérification se heurtent rapidement à l'explosion combinatoire des comportements à analyser.

Nous nous concentrons alors sur les techniques d'analyses dites statiques, dérivant des propriétés sur la dynamique du modèle sans analyser explicitement ses comportements engendrés. Ces méthodes reposent souvent sur une analyse des structures du modèle et peuvent avoir recours à des techniques d'abstraction afin d'en simplifier le comportement. Contrairement aux analyses dynamiques, une analyse statique dépend fortement de la sémantique du modèle et des propriétés recherchées. En pratique, les réponses sont souvent des approximations supérieures ou inférieures de la décision recherchée (donnant ainsi un espace « flou », où aucune décision exacte ne peut être dérivée).

Nous commençons cette section par un très bref aperçu des méthodes d'analyses dynamiques existantes (sous-section 2.3.1), puis nous détaillons les analyses statiques dévouées aux modélisa-

tions des RRB : les analyses topologiques du graphe des interactions (sous-section 2.3.2), l'utilisation d'opérations algébriques sur des représentations symboliques des fonctions discrètes (sous-section 2.3.3) et les techniques de réduction de modèles (sous-section 2.3.4).

2.3.1 Analyses Dynamiques

Les analyses dynamiques permettent usuellement de vérifier la concordance entre les comportements d'un modèle et une spécification exprimée, par exemple, en logique temporelle (comme CTL (Clarke & Emerson, 1981)), stochastique (comme CSL (Baier & Hermanns, 2003)) ou temporisée (comme TCTL (Alur, Courcoubetis & Dill, 1990)). Ces analyses possèdent ainsi de nombreux avantages : les logiques citées précédemment permettent d'exprimer de nombreux types de comportements, et donc ont potentiellement un grand champ d'application ; et les algorithmes de vérification peuvent être génériques : il suffit de savoir générer les transitions possibles à partir d'un état du modèle, et donc dépendent très peu du langage analysé.

Cependant, la vérification de ces logiques est un problème intrinsèquement coûteux en mémoire (et donc également en temps d'exécution) lorsque le nombre d'états à explorer est grand (Schnoebelen, 2002).

De nombreux travaux ont ainsi porté sur la compression de la mémoire requise par de telles vérifications en utilisant des représentations symboliques de l'espace des états, via les diagrammes de décision (par exemple, les BDD, *Binary Decision Diagrams* (Bryant, 1986)), pouvant être utilisés, par exemple, hiérarchiquement (Couvreur & Thierry-Mieg, 2005; Hamez, Thierry-Mieg & Kordon, 2009).

En pratique ces méthodes apportent un gain considérable sur le temps d'exécution des vérifications des modèles générant un très grand nombre d'états ; mais leur performance peut dépendre de paramètres liés à leur représentation symbolique (dans le cas des BDD par exemple, l'ordre choisi pour représenter les composantes de l'état impacte fortement leur efficacité), et de par leur complétude conservent une complexité théorique importante.

Enfin, l'utilisation des techniques de dépliages de Réseaux de Petri (Esparza & Heljanko, 2008; Chatain & Jard, 2006) n'a été que peu étudiée, à notre connaissance, dans le cadre des analyses dynamiques des RRB.

2.3.2 Analyses Topologiques du Graphe des Interactions

Lors de la modélisation d'un RRB, les informations qualitatives relatant des régulations entre les composants biologiques sont souvent les premières données accessibles. Ainsi posséder des analyses dérivant des propriétés sur la dynamique à partir du seul graphe des interactions est d'un grand intérêt, et a été le fruit de nombreux travaux.

Attracteurs : points fixes et oscillations soutenues

Les fameuses conjectures de René Thomas (1981), prouvées depuis dans de nombreux formalismes, sont un exemple d'analyse statique très efficace du graphe des interactions dont le résultat peut se lire directement sur le graphe :

1. l'existence de plusieurs états stables (ou points fixes) requiert la présence d'un circuit positif dans le graphe des interactions ;
2. l'existence d'oscillations soutenues requiert la présence d'un circuit négatif dans le graphe des interactions.

La première conjecture a notamment été démontrée dans le cadre des modélisations booléennes des RRB (Remy, Ruet & Thieffry, 2008; Richard, 2006), puis des modélisations discrètes (Richard & Comet, 2007).

Le nombre de points fixes des dynamiques peut alors être borné : étant donné le graphe des interactions de la dynamique F, si la suppression de k composants supprime tout circuit positif, alors F possède au plus 2^k points fixes dans le cadre des Réseaux Booléens (Aracena, 2008), ou au plus $\prod_{j=1}^{k} l_j$ points fixes dans le cadre des Réseaux Discrets (Richard, 2009), où l_j est le nombre de niveaux du j^{e} composant supprimé (si $k = 0$, le produit vaut 1).

Certaines contreparties de cette conjecture ont également été démontrées dans le cadre des Réseaux Booléens : si le graphe des interactions de la dynamique F est sans cycle (non-orienté) négatif et que tous les composants possèdent au moins un prédécesseur et aucun n'est non-signé, alors F possède au moins deux points fixes (un prédécesseur d'un composant est non-signé si il est à la fois un activateur et un inhibiteur du composant) ; il est également démontré que si le graphe ne possède aucun circuit positif et que tout composant possède au moins un prédécesseur et aucun n'est non-signé, alors F ne possède aucun point fixe (Aracena, Demongeot & Goles, 2004; Aracena, 2008).

La seconde conjecture a également été démontrée dans le cadre booléen (Remy et coll., 2008) et discret (Richard, 2010) : la présence d'oscillations soutenues implique la présence d'un circuit négatif dans le graphe des interactions. Un corollaire de cette propriété est que l'absence de circuit négatif implique la présence d'au moins un point fixe dans la dynamique.

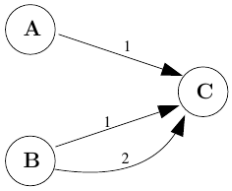
Cohérence avec des données expérimentales à l'état stable

Guziolowski, Bourde, Moreews & Siegel (2009) proposent de vérifier la cohérence d'un graphe des interactions avec des données expérimentales mesurant, à état stationnaire, la diminution ou l'augmentation des composants partant d'une solution initiale. Typiquement les données obtenues sont de la forme « le niveau de *a* est plus élevé, celui de *b* plus faible, etc. ». Les variations observées doivent alors s'expliquer par la topologie du graphe des interactions : par exemple, si *b* diminue significativement, alors un de ses activateurs a été diminué ou un de ses inhibiteurs a été augmenté.

Des analyses similaires sont également proposées par Klamt, Saez-Rodriguez, Lindquist, Simeoni & Gilles (2006), utilisant une analyse de dépendance entre les composants afin de déterminer la cohérence d'un graphe des interactions avec des données expérimentales.

2.3.3 Opérations Algébriques sur les Diagrammes de Décision

Naldi, Thieffry & Chaouiya (2007) proposent différentes analyses des Réseaux Discrets utilisant des opérations algébriques sur des Diagrammes de Décision Multi-valués (DDM) afin de déterminer efficacement les états stables de la dynamique et la fonctionnalité des circuits du graphe des interactions.



x_A	x_B	$\mathcal{K}_{x_C}(x_A, x_B)$
0	0	0
0	1	1
0	2	1
1	0	0
1	1	1
1	2	2

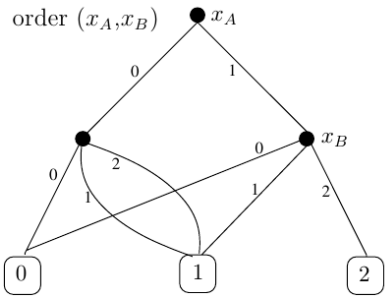


Figure 2.3 – Exemple de représentation des paramètres de René Thomas en DDM (Naldi et coll., 2007)

Un DDM est un graphe orienté acyclique possédant une seule racine. Chaque nœud représente le test d'une variable et possède autant d'arcs sortants que de valeurs possibles : l'arc libellé par la valeur de la variable est alors sélectionné et amène au test d'une autre variable, jusqu'à arriver sur une feuille, contenant alors la valeur de la décision. Étant donné un état x définissant n variables $x_1, \ldots, x_n$, il existe un unique chemin partant de la racine correspondant aux valeurs des variables et arrivant à une feuille.

Partant d'un Réseau Discret complètement paramétré, les paramètres de René Thomas de chaque composant sont encodés en DDM. La figure 2.3, tirée de (Naldi et coll., 2007), illustre ce procédé. Deux analyses sont alors produites à partir de ces représentations en DDM.

Le produit de deux DDM de racines respectives D^1 et D^2 est alors défini (nous supposons l'ordre des variables identique dans les deux DDM) :

- Si D^1 est une valeur (feuille), le produit est obtenu en multipliant toutes les valeurs de D^2 par D^1 (ou symétriquement en inversant D^1 et D^2).
- Si D^1 et D^2 testent la même variable x_i, le produit teste x_i et chaque valeur possible est associée au produit des fils correspondants de D^1 et D^2.
- Si D^1 teste une variable possédant un rang inférieur à celle de D^2, le produit obtenu est D^1 dont les fils résultent du produit avec D^2 (ou symétriquement en inversant D^1 et D^2).

Fonctionnalité des circuits

Étant donné un arc de a^{i-1} vers a^i dans le graphe des interactions avec seuil, nous notons $\theta_{i-1,i}$ le seuil de cette interaction et écrivons a^{i-1} franchit positivement le seuil $\theta_{i-1,i}$ si a^{i-1} passe du niveau $\theta_{i-1,i} - 1$ au niveau $\theta_{i-1,i}$, et dans le sens inverse pour le franchissement négatif.

Le contexte fonctionnel d'un circuit entre les composants $C = a^1, \ldots, a^m$ est l'ensemble des états x pour lesquels, pour tout a^{i-1} appartenant au circuit, le franchissement positif ou négatif du seuil $\theta_{i-1,i}$ par a^{i-1} entraîne le franchissement du seuil $\theta_{i,i+1}$ par a^i.

Ainsi pour chaque arc (a^{i-1}, a^i) libellé avec un seuil $\theta_{i-1,i}$, un DDM Γ_{a^{i-1},a^i} est dérivé de ceux encodant les paramètres discrets des composants a^{i-1} et a^i en associant à tout état x du système la valeur 1 (resp. -1) si le franchissement positif du seuil $\theta_{i-1,i}$ par a^{i-1} entraîne le franchissement positif (resp. négatif) du seuil $\theta_{i,i+1}$ par a^i.

Le DDM Γ_{a^{i-1},a^i} associe ainsi le contexte fonctionnel du circuit C localement pour l'arc (a^{i-1}, a^i) en notant le signe de cette fonctionnalité. Le contexte global du circuit est alors obtenu en effectuant

le produit des DDM $\Gamma_{a^1,a^2}, \ldots, \Gamma_{a^m,a^1}$.

Points fixes

Partant des DDM représentant les paramètres de René Thomas de chaque composant, un DDM S_a associant la valeur 1 à l'état x si $f^a(x) = x[a]$ et 0 sinon est dérivé pour chaque composant a.

L'ensemble des points fixes est alors obtenu en calculant le produit des DDM entre tous les composants : les états étant associés à la valeur 1 sont les points fixes de la dynamique.

Implémentée dans le logiciel GINsim (Gonzalez, Naldi, Sánchez, Thieffry & Chaouiya, 2006), cette méthode offre de très bonnes performances pour l'analyse des points fixes et des fonctionnalités dans les RRB. Toutefois, l'ordonnancement des variables impacte fortement la compacité des représentations en DDM, et donc l'efficacité de cette analyse, pouvant limiter la taille des RRB analysables. Dans le cas général, la recherche de ce meilleur ordonnancement possède un coût exponentiel (Nikolski, 2000).

2.3.4 Réductions de Modèles

La réduction d'un modèle est une étape naturelle lorsque celui-ci est trop détaillé, rendant son analyse très coûteuse : un modèle regroupant moins de composants tout en se comportant similairement est alors recherché.

Les méthodes de réductions ne permettent pas directement d'extraire des propriétés sur les dynamiques du système. Toutefois, elles peuvent permettre de comprendre le rôle de certains composants en étudiant leur impact sur la dynamique globale du modèle.

Naldi, Remy, Thieffry & Chaouiya (2009) étudient la réduction d'un Réseau Discret en supprimant un composant a dépourvu d'auto-régulation : les régulateurs de a sont prolongés aux composants régulés par a ; pour chaque composant régulé b, sa fonction discrète est modifiée en remplaçant toute occurrence de l'état de a par le résultat de la fonction discrète f^a (par exemple, $f^b(x) = x[c] \vee x[a]$ devient $f^b(x) = x[c] \vee f^a(x)$). Naldi et coll. démontrent alors les propositions suivantes : les atteignabilités des composants conservés sont réduites (des transitions peuvent être supprimées par la réduction) ; les points fixes sont conservés ; les attracteurs cycliques sont conservés.

Dans le cadre des modélisations par équations différentielles des RRB (et plus généralement des systèmes de réactions), Radulescu, Gorban, Zinovyev & Lilienbaum (2008) établissent des méthodes de réductions en se basant sur la vitesse des réactions : selon les relations entre les réactions, celles ayant une vitesse faible ou élevée peuvent être supprimées sans impacter la dynamique globale. Cette méthode de réduction permet alors d'extraire les réactions critiques dont les vitesses peuvent influencer de manière notable la dynamique du système.

Nous citons enfin les travaux de Gay, Soliman & Fages (2010) permettant d'établir automatiquement des relations de réduction entre différents modèles en graphes bipartis décrivant un système de réactions (que nous pouvons considérer comme plus génériques que les réseaux discrets). Un modèle est alors considéré comme la réduction d'un autre si le premier peut être obtenu par un ensemble d'opérations de suppression et de fusion des réactions.

2.4 Discussion

Les modélisations des RRB s'aident de la définition de plusieurs niveaux d'abstractions dans leur spécification, partant du graphe des interactions seul à la paramétrisation discrète puis hybride des dynamiques, comme le schématise la figure 1.3 page 4.

L'utilisation de techniques d'analyses statiques est motivée par la volonté d'extraire efficacement des propriétés sur les dynamiques d'un RRB. Appliquées au seul graphe des interactions, des propriétés générales sur la dynamique (bornes sur le nombre de points fixes, possibilité d'observer des oscillations soutenues, etc.) peuvent ainsi être déduites via une analyse topologique du graphe. Des techniques de manipulation algébrique sur les modèles en Réseaux Discrets ont également été présentées, ainsi que des réductions de modèle garantissant la conservation de certaines propriétés dynamiques.

La dérivation de propriétés dynamiques plus précises s'effectue par l'utilisation des méthodes de vérification formelle décrites dans la sous-section 2.3.1 généralement coûteuses et peu utilisables pour l'analyse de grands RRB à cause de l'explosion combinatoire des comportements. L'utilisation de techniques de dépliages de Réseaux de Petri, limitant l'explosion combinatoire due aux entrelacements des actions parallèles, permettrait sûrement d'améliorer de telles méthodes de vérifications, sans toutefois réduire la complexité théorique de l'analyse.

Notre approche se différencie par la définition d'un nouveau formalisme, les Frappes de Processus, permettant la modélisation discrète et hybride des RRB et sur lequel nous avons pu développer une analyse statique possédant une complexité théorique faible (au prix d'une potentielle absence de conclusion) promettant ainsi une bonne gestion des grands RRB.

Première partie

Les Frappes de Processus

Nous présentons les *Frappes de Processus* (ou *Process Hitting*), un nouveau formalisme introduit durant cette thèse dédié à la modélisation de systèmes complexes. Les Frappes de Processus regroupent un nombre fini de processus séparés en un ensemble de *sortes*. À tout instant, un et un seul processus de chaque sorte est présent ; un processus peut être remplacé par un autre processus de même sorte par la *frappe* d'un autre processus présent.
La simplicité de ce formalisme amènera au développement d'analyses statiques efficaces, présentées dans la partie III.
Le chapitre 3 donne la définition formelle des Frappes de Processus et décrit leur utilisation pour la modélisation des RRB. En particulier, les Frappes de Processus permettent la modélisation de RRB partiellement spécifiés à travers un procédé de raffinement partant de la *dynamique généralisée* du graphe des interactions.
Le chapitre 4 étudie la relation entre les Frappes de Processus et des formalismes existants comme les Automates Finis communicants, les Réseaux de Petri, le π-Calcul et les Réseaux Discrets utilisés pour la modélisation des RRB.

Chapitre **3**

Modélisation des Réseaux de Régulation Biologique en Frappes de Processus

Nous présentons le formalisme des Frappes de Processus (ou Process Hitting), introduit au cours de cette thèse. Les Frappes de Processus ont été conçues pour modéliser les systèmes concurrents complexes, et en particulier les Réseaux de Régulation Biologique (RRB). Elles décrivent la façon dont les processus interagissent entre eux : un processus frappe un autre processus (ou lui-même) pour le faire bondir en un autre processus actif. Nous montrons ici que les Frappes de Processus permettent une modélisation efficace des RRB en proposant plusieurs niveaux d'abstraction des dynamiques. Ce chapitre établit également le lien entre la modélisation des RRB proposée en Frappes de Processus et la modélisation par les réseaux discrets. En particulier, nous montrons que les paramètres discrets de René Thomas peuvent être déduits d'un modèle en Frappes de Processus.

3.1 Préliminaires

Ce chapitre introduit un nouveau formalisme pour la modélisation de systèmes complexes concurrents : les *Frappes de Processus* (ou *Process Hitting*).

Les Frappes de Processus regroupent un ensemble fini de *processus*, divisés en *sortes* : un processus appartient à une et une seule sorte. À tout instant, un et un seul processus de chaque sorte est actif, dénotant l'état courant d'une sorte. Une sorte change de processus actif après la *frappe* du processus actif par au plus un autre processus courant.

Ce formalisme se veut simple et peut être considéré comme une restriction de nombreux autres languages formels existants (voir chapitre 4). Cette simplicité est produite par l'introduction de contraintes sur les composants mis en jeu, ainsi que dans leur mode d'interaction. En particulier, deux processus de la même sorte ne peuvent être présents au même instant, et le remplacement d'un processus est contrôlé par la présence d'un seul autre processus.

Notre motivation pour l'introduction de ce formalisme repose sur l'intuition que la simplicité des Frappes de Processus engendre des modèles possédant une structure à partir de laquelle la dynamique sous-jacente peut être aisément devinée. Nous démontrons effectivement dans la partie III des analyses statiques des Frappes de Processus permettant d'extraire efficacement les points

fixes de la dynamique ainsi que des propriétés sur l'atteignabilité successive de processus à partir d'analyses toplogiques et d'interprétations abstraites des Frappes de Processus.

Dans ce chapitre, nous montrons que les Frappes de Processus sont particulièrement adaptées à la modélisation des dynamiques discrètes des Réseaux de Régulation Biologique (RRB). Nous exhibons une construction entièrement automatique de Frappes de Processus à partir du seul graphe des interactions d'un RRB que nous appellons la *dynamique généralisée* du RRB. Cette dynamique se veut la plus large possible tout en respectant certaines contraintes apportées par la topologie du graphe des interactions. Ensuite, par un procédé de raffinement (automatisable), la spécification (complète ou partielle) des fonctions discrètes entre les composants du RRB peut être construite en Frappes de Processus.

Ainsi, en autorisant plusieurs dégrés de précision dans la spécification des RRB, la modélisation par Frappes de Processus est particulièrement adaptée à l'étude de grands RRB dont les connaissances précises des interactions ne sont pas forcément connues.

Nous traitons également dans ce chapitre l'interprétation des Frappes de Processus en terme de RRB. Étant données des Frappes de Processus modélisant complètement un RRB, nous pouvons dériver les paramètres discrets de René Thomas. Ainsi, le résultat d'un travail de modélisation en Frappes de Processus peut être exporté vers d'autres méthodes reposant sur la spécification des paramètres discrets des RRB, permettant ainsi d'exploiter une potentielle complémentarité avec les outils de l'état de l'art. Nous montrons également l'inférence d'un graphe des interactions faisant état des régulations positives ou négatives entre les sortes de Frappes de Processus données.

Ce chapitre est structuré de la façon suivante. La section 3.2 présente formellement les Frappes de Processus et définit les opérateurs et objets associés. La section 3.3 propose une définition des RRB basée sur (Richard et coll., 2006). La section 3.4 montre une abstraction automatique des RRB en Frappes de Processus via la construction de la dynamique généralisée d'un graphe des interactions. Une stratégie de raffinement de la dynamique des Frappes de Processus par spécification de coopérations entre des sortes est présentée en section 3.5. L'interprétation des Frappes de Processus en RRB est abordée par la section 3.6. Des exemples de Frappes de Processus (arbitraires et appliquées à la modélisation de RRB) sont présentés dans la section 3.7. Enfin, la section 3.8 discute des contributions apportées par ce chapitre.

La plupart des résultats de ce chapitre sont publiés dans (Paulevé, Magnin & Roux, 2011*b*).

3.2 Les Frappes de Processus (ou Process Hitting)

Cette section présente le nouveau formalisme des Frappes de Processus (ou *Process Hitting*) sur lequel repose une majeure partie des résultats présentés dans ce manuscrit de thèse.

Un modèle en Frappes de Processus définit un nombre fini de *processus* concurrents regroupés en un ensemble de *sortes*. Un processus appartient à une et une seule sorte et est noté a_i où a est la sorte et i l'identifiant du processus au sein de la sorte a. À chaque instant, un et un seul processus de chaque sorte est présent, donnant un état des Frappes de Processus.

Les interactions concurrentes entre les processus sont définies par un ensemble d'*actions*. Ces actions décrivent le remplacement d'un processus par un autre de la même sorte, conditionné par la présence d'au plus un autre processus de l'état courant des Frappes de Processus. Une action est dénotée par $a_i \to b_j \not\uparrow b_k$, où a_i, b_j, b_k sont les processus de sortes a et b. Nécessairement, $b_j \neq b_k$

et $a = b \Rightarrow a_i = b_j$. Une action $h = a_i \to b_j \Rsh b_k$ se lit « a_i *frappe* b_j pour le faire *bondir* en b_k », où a_i, b_j, b_k sont appelés respectivement *frappeur*, *cible* et *bond*, et peuvent être obtenus avec les opérateurs $\mathsf{frappeur}(h), \mathsf{cible}(h), \mathsf{bond}(h)$, respectivement. Nous parlons d'*auto-frappe* quand $\mathsf{frappeur}(h) = \mathsf{cible}(h)$. La définition 3.1 formalise les Frappes de Processus.

Définition 3.1 (Frappes de Processus)**.** Les *Frappes de Processus* sont définies par un triplet $(\Sigma, L, \mathcal{H})$:
- $\Sigma = \{a, b, \ldots\}$ est l'ensemble fini et dénombrable des sortes,
- $L = \prod_{a\in\Sigma} L_a$ est l'ensemble des états, où $L_a = \{a_0 \ldots a_{l_a}\}$ est l'ensemble fini et dénombrable des processus de sorte $a \in \Sigma$, avec l_a un entier positif. On pose $a \neq b \Rightarrow a_i \neq b_j \ \forall (a_i, b_j) \in L_a \times L_b$,
- $\mathcal{H} = \{a_i \to b_j \Rsh b_k, \cdots \mid (a, b) \in \Sigma^2 \wedge (a_i, b_j, b_k) \in L_a \times L_b \times L_b \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ est l'ensemble fini des actions.

L'ensemble de tous les processus est noté **Proc** ($\mathbf{Proc} = \{a_i \mid a \in \Sigma \wedge a_i \in L_a\}$).

La sorte d'un processus a_i est extraite par $\Sigma(a_i) = a$ et l'ensemble des sortes présentes dans une action $h \in \mathcal{H}$ par $\Sigma(h) = \{\Sigma(\mathsf{frappeur}(h)), \Sigma(\mathsf{cible}(h))\}$. Étant donné un état $s \in L$, le processus de sorte $a \in \Sigma$ présent dans s est dénoté $s[a]$, ce qui correspond à la a^{e} coordonnée de l'état s. Nous définissons les notations suivantes : si $a_i \in L_a$, $a_i \in s \Leftrightarrow s[a] = a_i$; et si $ps \in \wp(\mathbf{Proc})$, nous notons $ps \subseteq s \Leftrightarrow \forall a_i \in ps, s[a] = a_i$.

Une action $h = a_i \to b_j \Rsh b_k \in \mathcal{H}$ est *jouable* dans $s \in L$ si et seulement si $s[a] = a_i$ et $s[b] = b_j$. Dans un tel cas, l'état résultant du jeu de l'action h dans s est dénoté $(s \cdot h)$, où $(s \cdot h)[b] = b_k$ et $\forall c \in \Sigma, c \neq b, (s \cdot h)[c] = s[c]$. Pour une raison de clarté, nous abrégeons $((s \cdot h) \cdot h')$, $h' \in \mathcal{H}$ par $(s \cdot h \cdot h')$.

Nous considérons maintenant les séquences (ou suites) d'actions : si A est une séquence d'actions, l'ensemble des sortes présentes dans A est donné par $\Sigma(A) = \bigcup_{n\in\mathbb{I}^A} \Sigma(A_n)$. Le premier (resp. dernier) processus de sorte a apparaissant dans la séquence est dénoté par $\mathsf{preum}_a(A)$ (resp. $\mathsf{der}_a(A)$) :

$$\mathsf{preum}_a(A) = \begin{cases} \varnothing & \text{si } a \notin \Sigma(A), \\ \mathsf{frappeur}(A_m) & \text{si } m = \min\{n \in \mathbb{I}^A \mid a \in \Sigma(A_n)\} \wedge \Sigma(\mathsf{frappeur}(A_m)) = a, \\ \mathsf{cible}(A_m) & \text{sinon si } m = \min\{n \in \mathbb{I}^A \mid a \in \Sigma(A_n)\} \wedge \Sigma(\mathsf{cible}(A_m)) = a \ ; \end{cases} \tag{3.1}$$

$$\mathsf{der}_a(A) = \begin{cases} \varnothing & \text{si } a \notin \Sigma(A), \\ \mathsf{bond}(A_m) & \text{si } m = \max\{n \in \mathbb{I}^A \mid a \in \Sigma(A_n)\} \wedge \Sigma(\mathsf{bond}(A_m)) = a, \\ \mathsf{frappeur}(A_m) & \text{sinon si } m = \max\{n \in \mathbb{I}^A \mid a \in \Sigma(A_n)\} \wedge \Sigma(\mathsf{frappeur}(A_m)) = a \ . \end{cases} \tag{3.2}$$

Parmi les séquences d'actions, les séquences particulières composées exclusivement d'actions successivement jouables sont appelées des *scénarios* (définition 3.2). Un scénario δ est dit jouable dans un état $s \in L$ si et seulement si δ_1 est jouable dans s et pour tout $n \in \mathbb{I}^\delta, n < |\delta|$, δ_{n+1} est jouable dans l'état $(s \cdot \delta_1 \cdot \cdots \cdot \delta_n)$; ou, de manière équivalente, δ est jouable dans s si et seulement si $\mathsf{support}(\delta) \subseteq s$ (équation (3.3)). L'état résultant du jeu séquentiel du scénario dans s est dénoté par $s \cdot \delta$. Nous pouvons facilement montrer que $\forall a_i \in \mathsf{fin}(\delta), (s \cdot \delta)[a] = a_i$ et $\forall b \in \Sigma \setminus \Sigma(\delta), (s \cdot \delta)[b] = s[b]$; où $\mathsf{fin}(\delta)$ est défini par l'équation (3.4).

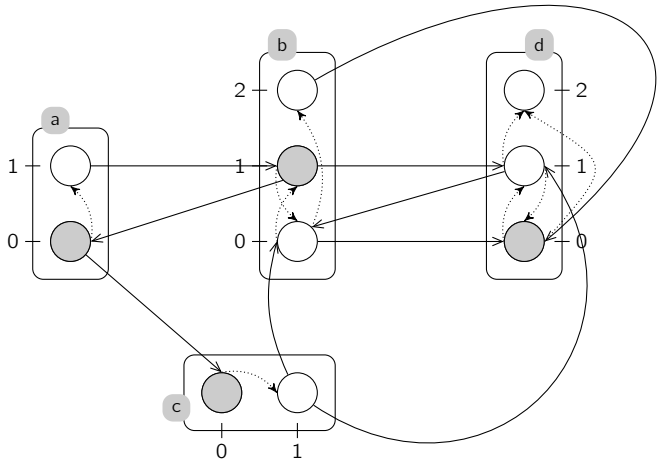


Figure 3.1 – Un exemple de Frappes de Processus. Les sortes sont représentées par des boîtes libellées, les processus par des cercles (les marques font référence aux identifiants des processus au sein de la sorte ; par exemple, a_0 est le processus marqué 0 dans la boîte libellée a). Une action (par exemple $a_0 \to c_0 \Rsh c_1$) est représentée par une paire d'arcs orientés ; la partie correspondant à la frappe (a_0 vers c_0) est en ligne pleine, la partie correspondant au bond (c_0 en c_1) est en ligne pointillée. L'état actuel contient les processus grisés : $\langle a_0, b_1, c_0, d_0 \rangle$.

Définition 3.2 (Scénario (**Sce**))**.** Étant données les Frappes de Processus $(\Sigma, L, \mathcal{H})$, un *scénario* δ est une séquence d'actions dans $\mathcal{H}$ telle que pour tout $n \in \mathbb{I}^{\delta}$, $a_i = \mathsf{frappeur}(\delta_n)$ (resp. $\mathsf{cible}(\delta_n)$) $\Rightarrow \mathsf{der}_a(\delta_{1..n-1}) \in \{\varnothing, a_i\}$. L'ensemble de tous les scénarios est dénoté par **Sce**. $\mathsf{support}(\delta)$ et $\mathsf{fin}(\delta)$ renvoient respectivement aux premiers et derniers processus de chaque sorte.

$$\mathsf{support}(\delta) = \{p \in \mathbf{Proc} \mid \Sigma(p) \in \Sigma(\delta) \wedge p = \mathsf{preum}_{\Sigma(p)}(\delta)\} \ , \tag{3.3}$$

$$\mathsf{fin}(\delta) = \{p \in \mathbf{Proc} \mid \Sigma(p) \in \Sigma(\delta) \wedge p = \mathsf{der}_{\Sigma(p)}(\delta)\} \ . \tag{3.4}$$

La figure 3.1 représente les Frappes de Processus $(\Sigma, L, \mathcal{H})$ où $\Sigma = \{a, b, c, d\}$, $L = \{a_0, a_1\} \times \{b_0, b_1, b_2\} \times \{c_0, c_1\} \times \{d_0, d_1, d_2\}$ et $\mathcal{H} = \{a_0 \to c_0 \Rsh c_1, a_1 \to b_1 \Rsh b_0, c_1 \to b_0 \Rsh b_1, b_1 \to a_0 \Rsh a_1, b_0 \to d_0 \Rsh d_1, b_1 \to d_1 \Rsh d_2, d_1 \to b_0 \Rsh b_2, c_1 \to d_1 \Rsh d_0, b_2 \to d_0 \Rsh d_2\}$. Jouer l'action $b_1 \to a_0 \Rsh a_1$ dans l'état $\langle a_0, b_1, c_0, d_0 \rangle$ amène dans l'état $\langle a_1, b_1, c_0, d_0 \rangle$. $\delta = a_0 \to c_0 \Rsh c_1 :: b_1 \to a_0 \Rsh a_1 :: a_1 \to b_1 \Rsh b_0 :: b_0 \to d_0 \Rsh d_1 :: d_1 \to b_0 \Rsh b_2$ est un scénario jouable dans l'état $s = \langle a_0, b_1, c_0, d_0 \rangle$, résultant en l'état $s \cdot \delta = \langle a_1, b_2, c_1, d_1 \rangle$.

Remarque 3.1. Les Frappes de Processus peuvent être considérées comme un type particulier des Automates Finis Communicants (Brand & Zafiropulo, 1983). Ce lien sera détaillé et discuté dans le chapitre 4.

3.3 Une Définition des Réseaux de Régulation Biologique

Dans le cadre de ce manuscrit, un Réseau de Régulation Biologique (RRB) regroupe un nombre fini de composants à chacun desquels un nombre fini de niveaux discrets est associé. Un RRB est généralement caractérisé par un graphe des interactions ainsi qu'un paramétrage discret, dit de René Thomas (Thomas, 1973). Alors que le graphe des interactions précise le sens des influences

(positives ou négatives) entre les composants, les paramètres spécifient précisément le niveau vers lequel un composant évoluera en fonction de l'état de ses régulateurs.

Nous nous basons sur la définition des RRB discrets présentée dans (Richard et coll., 2006) ; des définitions plus générales (par exemple celle présentée dans (Bernot et coll., 2007)) sont discutées en section 3.4. Étant donné un composant a, un intervalle de niveaux $[a_0; a_{l_a}]$ lui est associé. Par abus de notation, si $a_i \in L_a$ et $k \in \mathbb{N}$, $a_i + k = a_{i+k}$; de même si $a_j \in L_a$, $a_i - a_j = i - j$. Le graphe des interactions (définition 3.3) liste les régulations entre les composants. Une régulation est soit positive, soit négative, et possède un seuil : lorsque le niveau du régulateur est au-delà de ce seuil (ou égal), son influence sur le composant régulé conserve le signe de la régulation ; lorsque le niveau du régulateur est en deçà du seuil, son influence possède le signe inverse. Un composant ayant une influence positive est nommé *activateur*, un composant ayant une influence négative est nommé *inhibiteur*. Étant donnés deux composants a et b, on note $\mathsf{niveaux}_+(a \to b)$ (resp. $\mathsf{niveaux}_-(a \to b)$) les niveaux de a où a est un activateur (resp. inhibiteur) de b (définition 3.4).

Définition 3.3 (Graphe des Interactions)**.** Un *Graphe des Interactions* est un triplet (Γ, E_+, E_-) où Γ est un nombre fini de nœuds, et E_+ (resp. E_-) $\subset \{a \xrightarrow{t} b \mid a, b \in \Gamma \wedge t \in \mathbb{N}\}$ est l'ensemble des régulations positives (resp. négatives) entre deux nœuds, libellées avec un seuil.
Une régulation de a vers b est référencée de manière unique : si $a \xrightarrow{t} b \in E_+$ (resp. E_-), $\forall a \xrightarrow{t'} b \in E_+$ (resp. E_-), $t = t'$ et $\nexists t', a \xrightarrow{t'} b \in E_-$ (resp. E_+).

Définition 3.4 (Niveaux effectifs ($\mathsf{niveaux}$))**.** Étant donné un graphe des interactions (Γ, E_+, E_-) et deux composants $a, b \in \Gamma$:
– si $a \xrightarrow{t} b \in E_+$, $\mathsf{niveaux}_+(a \to b) = [t; l_a]$ et $\mathsf{niveaux}_-(a \to b) = [0; t-1]$;
– si $a \xrightarrow{t} b \in E_-$, $\mathsf{niveaux}_+(a \to b) = [0; t-1]$ et $\mathsf{niveaux}_-(a \to b) = [t; l_a]$;
– sinon, $\mathsf{niveaux}_+(a \to b) = \mathsf{niveaux}_-(a \to b) = \emptyset$.

Un paramètre de René Thomas $K_{a,A,B}$ spécifie le niveau vers lequel évolue le composant a soumis aux influences positives des composants dans A et aux influences négatives des composants dans B. Nous étendons cette définition standard à la spécification d'un intervalle (définition 3.5). A et B sont appelés les *ressources* de a (définition 3.6) et dépendent de l'état courant du RRB, noté s ($s[a]$ est le niveau de a dans l'état s).

Définition 3.5 (Paramètre discret $K_{a,A,B}$)**.** $K_{a,A,B} = [i_1; i_2]$ est un intervalle non-vide vers lequel tend le composant a en présence des activateurs référencés dans A et des inhibiteurs référencés dans B. Ainsi $0 \leq i_1 \leq i_2 \leq l_a$.
On note $j < K_{a,A,B} \Leftrightarrow j < i_1$ et $j > K_{a,A,B} \Leftrightarrow j > i_2$.

Définition 3.6 (Ressources $\mathsf{Res}_a(s)$)**.** Étant donné un état s du RRB, on note $\mathsf{Res}_a(s) = A, B$ les activateurs et inhibiteurs de a dans s définis comme :

$$A = \{b \in \Gamma \mid s[b] \in \mathsf{niveaux}_+(b \to a)\}$$
$$B = \{b \in \Gamma \mid s[b] \in \mathsf{niveaux}_-(b \to a)\}$$

Enfin, nous spécifions la *dynamique asynchrone* d'un RRB (définition 3.7) : étant donné un état s, une transition vers l'état s' est gouvernée par le changement de niveau d'un seul composant, qui se dirige unitairement vers la valeur de $K_{a,\mathrm{Res}_a(s)}$.

Définition 3.7 (Dynamique asynchrone)**.** Soit s un état du RRB. L'état succédant à s est donné par la fonction indéterministe $f(s)$:

$$f(s) = s' \Longrightarrow \exists a \in \Gamma, s'[a] = f^a(s) \wedge \forall b \in \Gamma, b \neq a, s[b] = s'[b] \quad \text{, avec}$$

$$f^a(s) = \begin{cases} s[a]+1 & \text{si } s[a] < K_{a,\mathrm{Res}_a(s)} \\ s[a] & \text{si } s[a] \in K_{a,\mathrm{Res}_a(s)} \\ s[a]-1 & \text{si } s[a] > K_{a,\mathrm{Res}_a(s)} \end{cases}$$

3.4 Dynamique Généralisée des Réseaux de Régulation

Cette section aborde l'interprétation automatique d'un RRB en Frappes de Processus, en s'intéressant aux dynamiques contraintes par le seul graphe des interactions, c.-à-d. en faisant abstraction du paramétrage discret du RRB. Le but est de produire un ensemble de dynamiques qui contient toutes les dynamiques possibles générées par un paramétrage donné du RRB. Ceci permet certaines analyses de dynamiques sur les RRB sans connaître leur paramétrage précis ; par exemple la vérification d'absence de comportements.

La *dynamique généralisée* du graphe des interactions peut se résumer en ces simples règles : le niveau d'un composant croît (resp. décroît) si et seulement si au moins un de ses régulateurs l'active (resp. l'inhibe) ; l'absence de régulateur est considérée comme la présence simultanée d'un activateur et d'un inhibiteur. La définition 3.8 formalise cette construction.

Définition 3.8 (Dynamique généralisée $\mathbf{PH}(\mathcal{G})$)**.** Les Frappes de Processus de la dynamique généralisée du graphe des interactions $\mathcal{G} = (\Gamma, E_+, E_-)$ sont données par $\mathbf{PH}(\mathcal{G})$, avec :

$$\mathbf{PH}(\mathcal{G}) = (\Gamma, \prod_{a\in\Gamma} L_a, \bigcup_{(a,b)\in\Gamma^2} \mathcal{H}_a^b) \quad \text{, où } L_a = \{a_0, \ldots, a_{l_a}\}\text{, et}$$

– si $b \xrightarrow{t} a \in E_+ \cup E_-$,

$$\mathcal{H}_a^b = \{b_k \to a_i \Rsh a_j \mid b_k \in L_b, a_i, a_k \in L_a, |i-j| = 1 \wedge i < j \Rightarrow k \in \mathsf{niveaux}_+(b \to a) \wedge i > j \Rightarrow k \in \mathsf{niveaux}_-(b \to a)\} \ ;$$

– sinon, si $a = b$ et $\nexists c \in \Gamma, c \xrightarrow{t} a \in E_+ \cup E_-$,

$$\mathcal{H}_a^a = \{a_i \to a_i \Rsh a_{i-1} \mid 0 < i \leq l_a\} \cup \{a_i \to a_i \Rsh a_{i+1} \mid 0 \leq i < l_a\} \ ;$$

– sinon, $\mathcal{H}_a^b = \emptyset$.

Nous remarquons que tout état s du RRB avec le graphe des interactions $\mathcal{G}$ est un état de $\mathbf{PH}(\mathcal{G})$, et réciproquement. Afin d'assurer le bien-fondé de la construction proposée, nous imposons la condition 3.1 sur les valeurs possibles des paramètres discrets des RRB : si tous les régulateurs d'un composant sont activateurs, alors le composant tend vers son niveau le plus élevé ; si tous

les régulateurs sont inhibiteurs, son niveau tend vers zéro. Cette condition peut être vue à la fois comme une version restreinte et « allégée » de la condition de monotonicité des paramètres discrets discutée dans (Bernot et coll., 2007). La condition de monotonicité pourrait se résumer en « plus il y a d'activateurs (resp. d'inhibiteurs), plus le paramètre discret est élevé (resp. faible) ». Dans notre condition 3.1, nous n'imposons pas d'ordre entre les valeurs de paramètres discrets, mais nous imposons leurs valeurs extrêmes.

Condition 3.1 (Extrémité des paramètres discrets)**.** Soit (Γ, E_+, E_-) un graphe des interactions, et soit $R_a = \{b \in \Gamma \mid b \xrightarrow{t} a \in E_+ \cup E_-\}$ l'ensemble des régulateurs de $a \in \Gamma$, nous imposons :
- $K_{a,R_a,\emptyset} = l_a$; et
- $K_{a,\emptyset,R_a} = 0$.

Nous pouvons alors prouver le théorème 3.1 qui établit la préservation des transitions au sein du RRB par son encodage en Frappes de Processus.

Théorème 3.1. *Soit f la dynamique asynchrone d'un RRB ayant $\mathcal{G} = (\Gamma, E_+, E_-)$ pour graphe des interactions et des paramètres discrets vérifiant la condition 3.1. Avec $\mathbf{PH}(\mathcal{G}) = (\Gamma, L, \mathcal{H})$, pour tout état $s \in L$, $f(s) = s', s \neq s' \Longrightarrow \exists h \in \mathcal{H}, s \cdot h = s'$.*

Démonstration. Si $f(s) = s'$ et $s \neq s'$, il existe $a \in \Gamma$ tel que $s'[a] - s[a] = \pm 1$ et $\forall b \in \Gamma, b \neq a, s[b] = s'[b]$. Soient $A, B = \mathsf{Res}_a(s)$ les activateurs et inhibiteurs de a dans s.
- si a n'a aucun régulateur ($A \cup B = \emptyset$), nous remarquons que $s[a] \to s[a] \Rsh s[a'] \in \mathcal{H}$;
- si $\exists b \in A$ et $\exists c \in B$, alors $s[b] \in \mathsf{niveaux}_+(b \to a)$ et $s[c] \in \mathsf{niveaux}_-(c \to a)$ donc soit $s[b] \to s[a] \Rsh s[a'] \in \mathcal{H}$, soit $s[c] \to s[a] \Rsh s[a'] \in \mathcal{H}$.
- si $\exists b \in A$ et $B = \emptyset$, alors, d'après la condition 3.1, $s'[a] - s[a] = 1$; donc $s[b] \to s[a] \Rsh s[a'] \in \mathcal{H}$;
- enfin, si $\exists c \in B$ et $A = \emptyset$, alors, d'après la condition 3.1, $s'[a] - s[a] = -1$; donc $s[c] \to s[a] \Rsh s[a'] \in \mathcal{H}$.

□

Nous soulignons le fait que la complexité de la construction de la dynamique généralisée en Frappes de Processus est linéaire suivant le nombre d'arcs dans le graphe des interactions.

La validité de la dynamique généralisée obtenue dépend évidemment de la définition des RRB introduite en section 3.3 et de la condition 3.1. Nous établissons plusieurs remarques :
- la condition 3.1 pourrait être relâchée en spécifiant, par exemple, les niveaux minimum m_a et maximum M_a d'un composant a (à la place de 0 et l_a). La construction rajouterait alors des frappes de a sur a amenant a_0 en a_{m_a} et a_{l_a} en a_{M_a}.
- Dans la définition étendue des RRB proposée par Bernot et coll. (2007), le rôle d'activation ou d'inhibition d'un régulateur ne dépend pas d'un seuil : b peut activer a quand b_1 ou b_3 est présent, et peut inhiber a quand b_2 est présent. Si on admet que b est toujours un activateur ou inhibiteur de a, notre construction peut être étendue simplement en modifiant la définition des fonctions $\mathsf{niveaux}_+$ et $\mathsf{niveaux}_-$ (définition 3.4) pour prendre en compte cette nouvelle spécification.
- Si nous autorisons un régulateur b de a à avoir des niveaux où il n'est ni activateur ni inhibiteur, il faut alors considérer les niveaux inactifs de b comme des niveaux à la fois activateurs et inhibiteurs. Encore une fois, cela peut s'étendre facilement dans notre construction en modifiant les fonctions $\mathsf{niveaux}_+$ et $\mathsf{niveaux}_-$.

3.5 Raffinement des Dynamiques par Coopération

En Frappes de Processus, le bond d'un processus est conditionné par la présence d'*au plus* un autre processus. Nous nous demandons alors comment modéliser des bond conditionnés par la présence de plusieurs autres processus. Par exemple, dans le cadre des RRB, étant donnés deux composants a et b régulant tous deux un composant c, l'effet de a sur c peut dépendre du niveau de b : on parle alors de *coopération* entre les sortes a et b sur c. Ces coopérations sont généralement spécifiées par des fonctions booléennes entre les composants (Richard et coll., 2006; Bernot et coll., 2008). Nous montrons comment construire de telles fonctions en Frappes de Processus.

Soit $(\Sigma, L, \mathcal{H})$ des Frappes de Processus et soit $\sigma \subset \Sigma$ un sous-ensemble de sortes coopérant sur un processus c_j pour le faire bondir en c_k. L'ensemble des états formés par les processus coopérants est dénoté par $S = \prod_{z\in\sigma} L_z$. Le sous-ensemble des états où la coopération est effective (où c_j doit bondir en c_k) est défini par $\top \subset S$.

Nous proposons une construction de cette coopération en ajoutant une nouvelle sorte υ au sein des Frappes de Processus. Cette sorte est appelée *sorte coopérative* et contient un processus pour chaque état possible des processus coopérants : $L_\upsilon = \{\upsilon_\varsigma, \forall \varsigma \in S\}$. Chaque processus coopérant z_i, $z \in \sigma$, frappe tout processus υ_ς où $\varsigma[z] \neq z_i$ pour le faire bondir en $\upsilon_{\varsigma'}$ où $\varsigma'[z] = z_i$ et $\varsigma'[a] = \varsigma[a], \forall a \in \Sigma, a \neq z$. On note $\mathcal{H}_\sigma$ cet ensemble d'actions (équation (3.5)). Ainsi, le processus actif de sorte υ reflète l'état courant des processus coopérants.

Les frappes sur le processus c_j (pour le faire bondir en c_k) provenant des processus de sorte dans σ (notées $\mathcal{H}_{rm}$, équation (3.6)) sont alors remplacées par des frappes provenant des processus de sorte υ correspondant aux états sélectionnés dans $\top$ ($\mathcal{H}_{coop}$, équation (3.7)).

$$\mathcal{H}_\sigma = \{z_i \to \upsilon_\varsigma \Rsh \upsilon_{\varsigma'} \mid z \in \sigma \wedge z_i \in L_z \wedge \varsigma \in S \wedge \varsigma[z] \neq z_i \wedge \varsigma'[z] = z_i \wedge (\varsigma'[a] = \varsigma[a], \forall a \in \Sigma, a \neq z)\} \tag{3.5}$$

$$\mathcal{H}_{rm} = \{z_i \to c_j \Rsh c_k \in \mathcal{H} \mid z \in \sigma\} \tag{3.6}$$

$$\mathcal{H}_{coop} = \{\upsilon_\varsigma \to c_j \Rsh c_k \mid \varsigma \in \top\} \ . \tag{3.7}$$

Les Frappes de Processus raffinées par cette coopération sont définies par l'équation (3.8).

$$(\Sigma \cup \{\upsilon\}, L \times L_\upsilon, (\mathcal{H} \setminus \mathcal{H}_{rm}) \cup \mathcal{H}_\sigma \cup \mathcal{H}_{coop}) \ . \tag{3.8}$$

Exemple. Soit $(\{a, b, c\}, \{a_0, a_1\} \times \{b_0, b_1\} \times \{c_0, c_1\}, \mathcal{H})$ les Frappes de Processus contenant les frappes $\{a_0 \to c_0 \Rsh c_1, b_1 \to c_0 \Rsh c_1\} \subset \mathcal{H}$. Le raffinement par la coopération entre a_0 et b_1 sur c_0 ($\sigma = \{a, b\}$, $\top = \{\langle a_0, b_1\rangle\}$) est illustré par la figure 3.2.

Dans le cadre de l'interprétation complète d'une fonction discrète de la forme $f : L_{a^1} \times \cdots \times L_{a^n} \mapsto L_c$ (une telle fonction associe un état des sortes coopérantes $\sigma = \{a^1, \ldots, a^n\}$ à un processus bond c_k), les ensembles $\mathcal{H}_{rm}$ (équation (3.6)) et $\mathcal{H}_{coop}$ (équation (3.7)) sont remplacés respectivement par $\mathcal{H}^f_{rm}$ (équation (3.9)) et $\mathcal{H}^f_{coop}$ (équation (3.10)).

$$\mathcal{H}^f_{rm} = \{z_i \to c_j \Rsh c_k \in \mathcal{H} \mid z \in \sigma, z_i \in L_z, c_j, c_k \in L_c\} \tag{3.9}$$

$$\mathcal{H}^f_{coop} = \{\upsilon_\varsigma \to c_j \Rsh c_k \mid c_j \in L_c \wedge f(\varsigma) = c_k\} \ . \tag{3.10}$$

La complexité d'une telle construction est exponentielle suivant le nombre de sortes coopérantes : la sorte coopérative υ regroupe $|L_{a^1}|\cdot\ldots\cdot|L_{a^n}|$ processus. Nous remarquons toutefois qu'il est possible

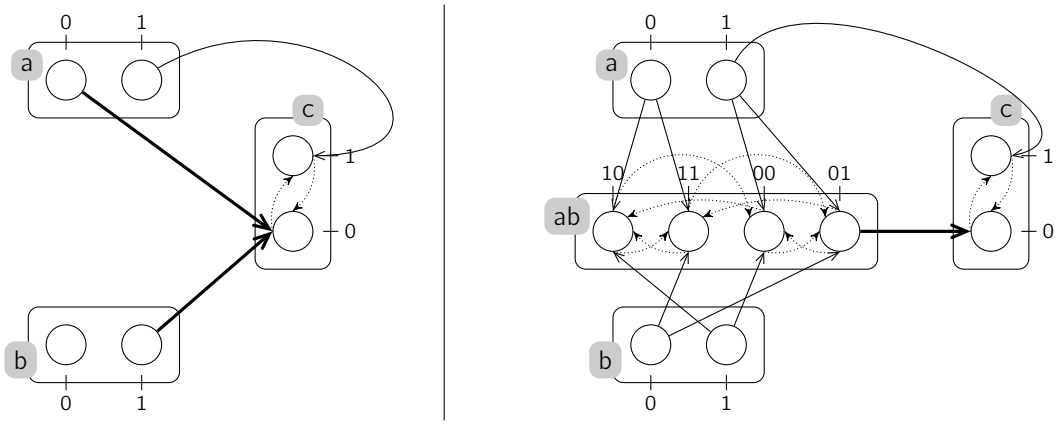


Figure 3.2 – Construction d'une frappe coopérative entre a_0 et b_1 sur c_0 (lignes épaisses) : $\sigma = \{a, b\}$, $\top = \{\langle a_0, b_1\rangle\}$, $\upsilon = ab$. ab_{01} est le processus correspondant à l'état $\langle a_0, b_1\rangle$.

de réduire drastiquement la taille de υ en factorisant la fonction réalisée, comme illustré dans la figure 3.3.

Il est important de remarquer que cette construction impose un décalage temporel dans l'application de la coopération : une frappe est remplacée par une succession de frappes. L'impact d'un tel décalage sur la dynamique obtenue est étudié dans le chapitre 4.

Appliqué sur des Frappes de Processus modélisant la dynamique généralisée d'un RRB, le raffinement par coopération permet d'étudier une sous-dynamique correspondant à l'application de certaines contraintes sur les fonctions entre les composants (ou, de manière équivalente, sur les paramètres discrets). En particulier, notre méthode de raffinement permet la spécification partielle de ces fonctions (par exemple, sur trois régulateurs, on ne précise la coopération qu'entre deux), la dynamique obtenue étant la plus large satisfaisant les nouvelles contraintes. Nous notons finalement que la traduction en Frappe de Processus d'un RRB entièrement spécifié est complètement automatisable à l'aide des définitions présentées dans cette section.

Enfin, nous remarquons que cette notion de *sorte coopérative* rappelle la notion de *complexe* en biologie : deux composants s'associent pour former un complexe, et le complexe agit sur le composant cible.

3.6 Des Frappes de Processus vers les Réseaux de Régulation

Dans cette section, nous établissons des liens entre un modèle en Frappes de Processus et un Réseau de Régulation Biologique. Dans un premier temps, nous cherchons à extraire les paramètres discrets d'un RRB correspondant au modèle en Frappes de Processus. Enfin, nous traitons l'inférence d'un graphe des interactions.

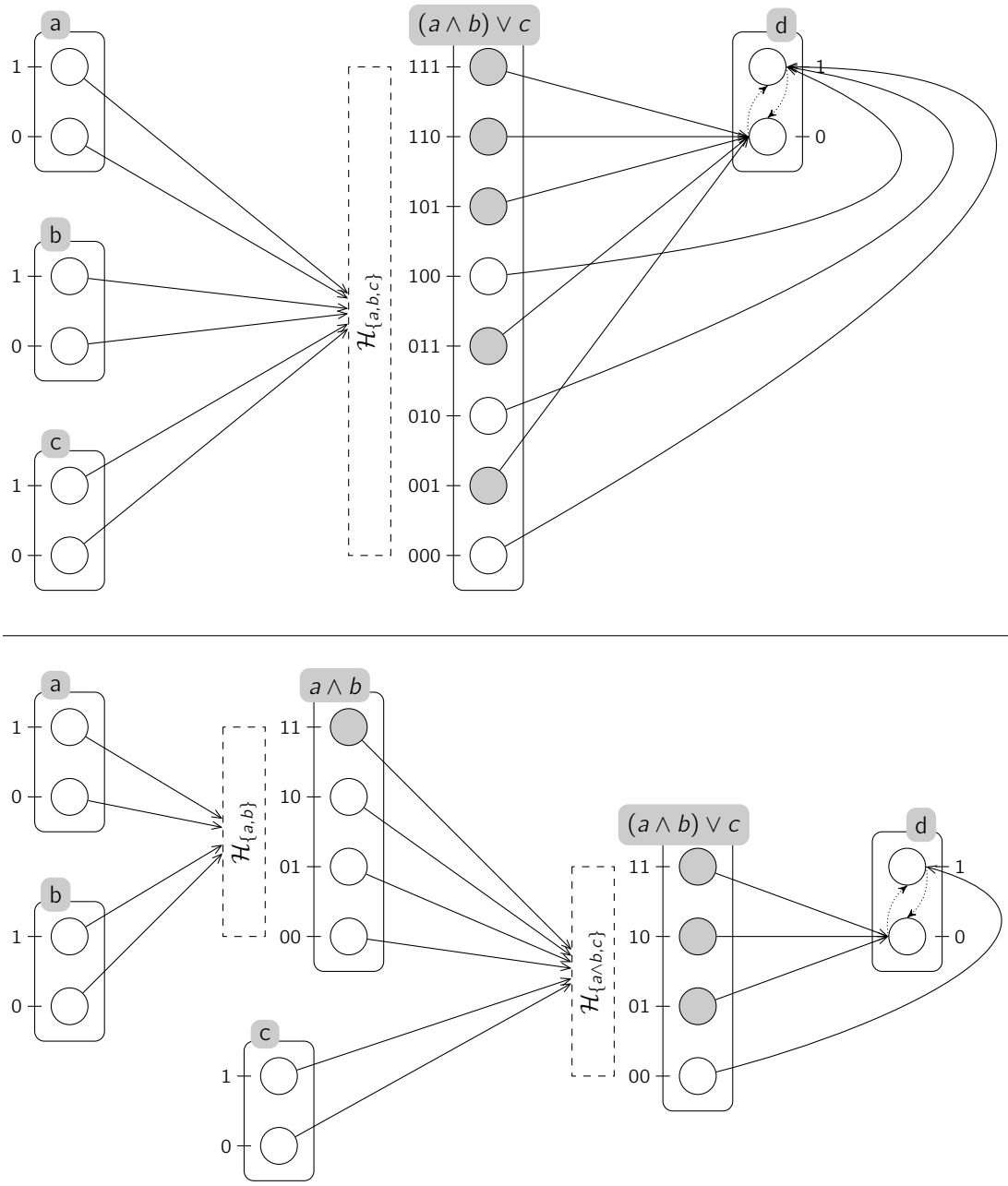


Figure 3.3 – Construction de la fonction discrète $f^d(a, b, c) = (a_1 \wedge b_1) \vee c_1$: (haut) une seule sorte coopérative est créée, résultant en un nombre exponentiel de processus à l'intérieur de cette sorte. (bas) Factorisation des coopérations : une sorte pour la coopération entre a et b ($a \wedge b$) est créée, puis une sorte pour la coopération entre $a \wedge b$ et c est créée. Le nombre de processus par sorte est ainsi fortement réduit. Les processus grisés sont ceux satisfaisant l'expression booléenne associée à la sorte coopérative. Pour une raison de clarté, les frappes $\mathcal{H}_\sigma$ (équation (3.5)) vers les sortes coopératives ne sont pas dessinées, mais symbolisées par un rectangle en traits discontinus.

3.6.1 Inférence des Paramètres Discrets de René Thomas

Comme présenté dans la section 3.3, un paramètre discret de René Thomas spécifie le niveau attracteur d'un composant quand ses régulateurs sont dans une configuration donnée. De nombreux outils et méthodes dédiés à l'étude de RRB utilisent l'ensemble complet des paramètres de René Thomas comme donnée nécessaire (par exemple Richard et coll., 2006; Siebert & Bockmayr, 2006; Ahmad et coll., 2006). Dans cette sous-section, nous donnons une méthode pour inférer automatiquement les valeurs des paramètres de René Thomas à partir d'un modèle de RRB en Frappes de Processus.

Soient $\mathcal{G} = (\Gamma, E_+, E_-)$ un graphe des interactions et $(\Sigma, L, \mathcal{H})$ les Frappes de Processus où les sortes sont soit des composants de $\mathcal{G}$, soit des sortes coopératives, c.-à-d. $\Sigma = \Gamma \cup \{\upsilon^1, \ldots, \upsilon^u\}$ avec $\forall \upsilon \in \{\upsilon^1, \ldots, \upsilon^u\}, \Sigma(\upsilon) \subset \Gamma$, où $\Sigma(\upsilon)$ est l'ensemble des sortes coopérantes représentées par υ. Soit $K_{a,A,B}$ le paramètre de René Thomas à inférer. Pour chaque sorte $b \in A \cup B$, nous définissons son contexte $C^b_{a,A,B}$ comme étant le sous-ensemble de processus de L_b imposé par le paramètre de René Thomas : si $b \in A$ (resp. B), seuls les processus correspondant aux niveaux effectifs positifs (resp. négatifs) (définition 3.4) sont considérés. Pour chaque processus $b \in \Gamma$ qui ne régule pas a (c.-à-d. $b \notin A \cup B$), son contexte $C^b_{a,A,B}$ est simplement L_b (équation (3.11)). Le contexte $C^\upsilon_{a,A,B}$ d'une sorte coopérative est l'ensemble des états regroupant les contextes des sortes coopérantes (équation (3.12)).

$$\forall b \in \Gamma,\ C^b_{a,A,B} = \begin{cases} \text{niveaux}_+(b \to a) & \text{si } b \in A, \\ \text{niveaux}_-(b \to a) & \text{si } b \in B, \\ L_b & \text{sinon.} \end{cases} \tag{3.11}$$

$$\forall \upsilon \in \{\upsilon^1, \ldots, \upsilon^u\},\ C^\upsilon_{a,A,B} = \{\upsilon_\varsigma \mid \varsigma \in \textstyle\prod_{b \in \Sigma(\upsilon)} C^b_{a,A,B}\}\ . \tag{3.12}$$

Nous dénotons par $\mathcal{H}_{a,A,B}$ le sous-ensemble des actions $\mathcal{H}$ frappant un processus de sorte a qui peuvent être jouées dans les contextes fixés (équation (3.13)). Un processus de sorte a est atteignable si il appartient au contexte $C^a_{a,A,B}$, ou si il est le bond d'une action dans $\mathcal{H}_{a,A,B}$. L'ensemble de tels processus est noté $L^?_{a,A,B}$ (équation (3.14)). L'ensemble des processus de sorte a atteignables et non frappés par des actions dans $\mathcal{H}_{a,A,B}$ est noté $L^*_{a,A,B}$ (équation (3.15)). Alors, tant que les processus actifs restent dans les contextes imposés, si le processus de sorte a est dans $L^*_{a,A,B}$, il ne sera jamais frappé. Nous appelons cet ensemble $L^*_{a,A,B}$ l'ensemble des *processus focaux* de a.

$$\mathcal{H}_{a,A,B} = \{b_i \to a_j \curvearrowright a_k \in \mathcal{H} \mid \exists b \in \Gamma, b_i \in C^b_{a,A,B} \wedge a_j \in C^a_{a,A,B}\} \tag{3.13}$$

$$L^?_{a,A,B} = C^a_{a,A,B} \cup \{a_k \mid \exists b_i \to a_j \curvearrowright a_k \in \mathcal{H}_{a,A,B}\} \tag{3.14}$$

$$L^*_{a,A,B} = L^?_{a,A,B} \setminus \{a_j \mid \exists b_i \to a_j \curvearrowright a_k \in \mathcal{H}_{a,A,B}\}\ . \tag{3.15}$$

Il reste à vérifier que les processus focaux sont bien des attracteurs, c.-à-.d que les actions $\mathcal{H}_{a,A,B}$ font bondir les processus de sorte a en direction des processus focaux. Si une telle condition est satisfaite, alors les processus focaux correspondent à la valeur du paramètre de René Thomas recherché. Il est à noter que toutes ces opérations se calculent en un temps linéaire par rapport au nombre d'actions au sein des Frappes de Processus.

La condition 3.2 impose, pour un contexte donné, que toutes les frappes sur un processus a_j le font bondir en direction de tous les processus attracteurs référencés dans $L^*_{a,A,B}$. Ceci assure que $L^*_{a,A,B}$ représente un intervalle de niveaux (propriété 3.1). Ainsi, $L^*_{a,A,B}$ est bien la valeur du paramètre de René Thomas (étendu aux intervalles) $K_{a,A,B}$ (théorème 3.2).

Condition 3.2 (Les processus focaux sont attracteurs)**.** $\forall b_i \to a_j \nrightarrow a_k \in \mathcal{H}_{a,A,B}$, $\forall a_f \in L^*_{a,A,B}$, $|f-k| < |f-j|$.

Propriété 3.1. *Si $L^*_{a,A,B}$ satisfait la condition 3.2, alors $L^*_{a,A,B}$ est un intervalle.*

Démonstration. Si $L^*_{a,A,B} = \{a_f, \ldots, a_{f'}\}$ n'est pas un intervalle, il existe $b_i \to a_j \nrightarrow a_k \in \mathcal{H}_{a,A,B}$ tel que $f < j < f'$. Si la condition 3.2 s'applique, nous avons $|f-k| < |f-j| \Rightarrow k < j \Rightarrow |f'-k| > |f'-j|$ ce qui est contradictoire. □

Théorème 3.2. *Soient $\mathcal{G} = (\Gamma, E_+, E_-)$ un graphe des interactions et $(\Gamma \cup \Upsilon, L, \mathcal{H})$ les Frappes de Processus issues d'un raffinement depuis* **PH**$(\mathcal{G})$*. Soit A (resp. B) $\subseteq \Gamma$ les sortes activant (resp. inhibant) effectivement la sorte a. Si $L^*_{a,A,B} \neq \emptyset$ et si la condition 3.2 est vérifiée, alors $K_{a,A,B} = L^*_{a,A,B}$.*

Démonstration. Par définition de $L^*_{a,A,B}$ et par application de la condition 3.2 et de la propriété 3.1. □

En conséquence, il peut exister des configurations où l'inférence des paramètres de René Thomas est impossible. Dans le cas où $L^*_{a,A,B} = \emptyset$, le processus de sorte a est instable dans le contexte donné : il existe toujours une configuration où il peut se faire frapper. Dans le cas où la condition 3.2 n'est pas vérifiée, le processus de sorte a peut, de manière indéterministe, soit devenir instable, soit être attiré par des points focaux opposés. Une des raisons principales de la présence d'un indéterminisme sur les processus focaux au sein des Frappes de Processus est l'absence de coopération entre des frappes sur une même cible : elles peuvent indépendamment faire bondir la cible vers des directions opposées, alors qu'une coopération imposerait un comportement localement déterministe.

Nous laissons ouverte la question du réalisme biologique de telles dynamiques instables ou indéterministes dans un contexte donné.

3.6.2 Inférence du Graphe des Interactions

Étant données des Frappes de Processus $\mathcal{PH} = (\Sigma, L, \mathcal{H})$ et un ensemble de composants $\Gamma \subseteq \Sigma$, nous cherchons à extraire des Frappes de Processus les influences (positives ou négatives) entre les composants de Γ. Pour chaque sorte $a \in \Gamma$, nous supposons une relation d'ordre totale $\prec_a$ entre les processus L_a (typiquement, $a_i \prec_a a_{i+1}$).

L'inférence se déroule en deux étapes : la suppression des sortes coopératives (celles non référencées dans Γ) et l'inférence des influences.

Suppression des sortes coopératives. Soit $c \in \Sigma \setminus \Gamma$, nous définissons la projection $\mathcal{PH} \downarrow c$ (définition 3.9) qui supprime la sorte c et remplace les paires d'actions $a_i \to c_k \nrightarrow c_{k'}$, $c_{k'} \to b_j \nrightarrow b_{j'}$ par une seule action $a_i \to b_j \nrightarrow b_{j'}$.

Définition 3.9 (Projection de Frappes de Processus)**.** Étant données des Frappes de Processus $\mathcal{PH} = (\Sigma, L, \mathcal{H})$, leur *projection selon $c \in \Sigma$* est définie par $\mathcal{PH} \downarrow c = (\Sigma \setminus \{c\}, \prod_{a \in \Sigma \setminus \{c\}} L_a, \mathcal{H}')$ avec

$$\mathcal{H}' = \{a_i \to b_j \nrightarrow b_{j'} \in \mathcal{H} \mid a \neq c \wedge b \neq c\}$$
$$\cup \{a_i \to b_j \nrightarrow b_{j'} \mid \{a_i \to c_k \nrightarrow c_{k'}, c_{k'} \to b_j \nrightarrow b_{j'}\} \subseteq \mathcal{H}\}$$

Inférence du graphe des interactions. Soient $(\Gamma, L^{\Gamma}, \mathcal{H}^{\Gamma})$ les Frappes de Processus $\mathcal{PH}$ dont les sortes absentes de Γ ont été projetées. Étant donnés $a, b \in \Gamma$, nous cherchons l'influence (positive ou négative) de a sur b. De manière similaire au calcul des processus focaux de la sous-section précédente, pour tout $a_i \in L_a$, on définit L^*_{b,a_i} les processus focaux de sorte b en présence de a_i (équation (3.16)).

$$L^*_{b,a_i} = \{b_j \in L_b \mid \nexists a_i \to b_j \Rsh b_k \in \mathcal{H}^{\Gamma}\} \tag{3.16}$$

Soient $a_i, a_j \in L_a$, tels que $a_i \prec_a a_j$ et tels que L^*_{b,a_i} et L^*_{b,a_j} satisfassent la condition 3.2. Nous pouvons alors écrire $L^*_{b,a_i} = [b_{i1}; b_{i2}]$ et $L^*_{b,a_j} = [b_{j1}; b_{j2}]$. Alors, a a une influence positive sur b si $L^*_{b,a_i} <_{[]} L^*_{b,a_j}$; a a une influence négative sur b si $L^*_{b,a_j} <_{[]} L^*_{b,a_i}$, où la relation $<_{[]}$ est donnée par l'équation (3.17). La définition 3.10 résume ces opérations.

$$[b_{i1}; b_{i2}] <_{[]} [b_{j1}; b_{j2}] \iff (b_{i1} < b_{j1} \wedge b_{i2} \leq b_{j2}) \vee (b_{i1} \leq b_{j1} \wedge b_{i2} < b_{j2}) \ . \tag{3.17}$$

Définition 3.10 ($\mathbf{GI}((\Sigma, L, \mathcal{H}), \Gamma)$)**.** Le graphe des interactions entre les composants Γ inféré depuis les Frappes de Processus $\mathcal{PH} = (\Sigma, L, \mathcal{H})$ est noté $\mathbf{GI}(\mathcal{PH}, \Gamma) = (\Gamma, E_+, E_-)$. Dans le cadre des Frappes de Processus $\mathcal{PH} \downarrow v^1 \cdots \downarrow v^n$ où $\{v^1, \ldots, v^n\} = \Sigma \setminus \Gamma$, alors pour tout $a, b \in \Gamma$, si $\exists a_i, a_j \in L_a, a_i \prec_a a_j$, tels que L^*_{b,a_i} et L^*_{b,a_j} satisfont la condition 3.2 :
– $L^*_{b,a_i} <_{[]} L^*_{b,a_j} \implies a \to b \in E_+$;
– $L^*_{b,a_j} <_{[]} L^*_{b,a_i} \implies a \to b \in E_-$;

En conclusion de cette inférence, nous remarquons qu'il existe des cas où un régulateur est marqué à la fois comme ayant une action positive et négative sur un même composant. L'inférence des seuils des interactions n'est pas abordée par cette section.

3.7 Exemples « Jouets » de Frappes de Processus

Dans cette section, nous exhibons quelques exemples simples de Frappes de Processus : des Frappes de Processus arbitraires et des Frappes de Processus de la dynamique généralisée, puis raffinée, de RRB élémentaires. Des applications des Frappes de Processus aux RRB sont détaillées dans le chapitre 11.

Frappes de Processus arbitraires et atteignabilités

- La figure 3.4 représente les Frappes de Processus d'un problème élémentaire de concurrence : atteindre le processus z_2 depuis l'état initial $\langle a_0, b_0, c_0, z_0 \rangle$ (processus grisés) requiert le jeu de l'action $b_0 \to c_0 \Rsh c_1$ avant l'action $b_0 \to b_0 \Rsh b_1$.
- La figure 3.5 représente des Frappes de Processus à la dynamique plus complexe. En considérant l'état initial $\langle a_1, b_1, z_1, r_0 \rangle$, nous notons, par exemple, que l'action $b_1 \to z_2 \Rsh z_1$ est jouable au plus deux fois, rendant l'atteinte du processus r_3 impossible.

Frappes de Processus modélisant des RRB

Nous présentons la modélisation de deux RRB dont les graphes des interactions sont donnés par la figure 3.6.

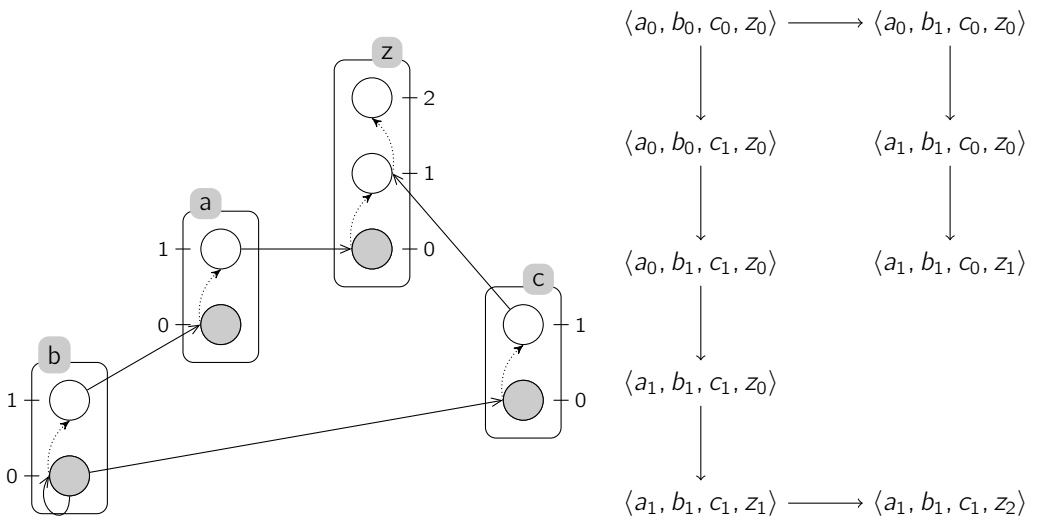


Figure 3.4 – (gauche) Exemples de Frappes de Processus arbitraires. Les processus grisés représentent l'état initial (ici, $\langle a_0, b_0, c_0, z_0\rangle$). (droite) Graphe des états en partant de l'état initial indiqué.

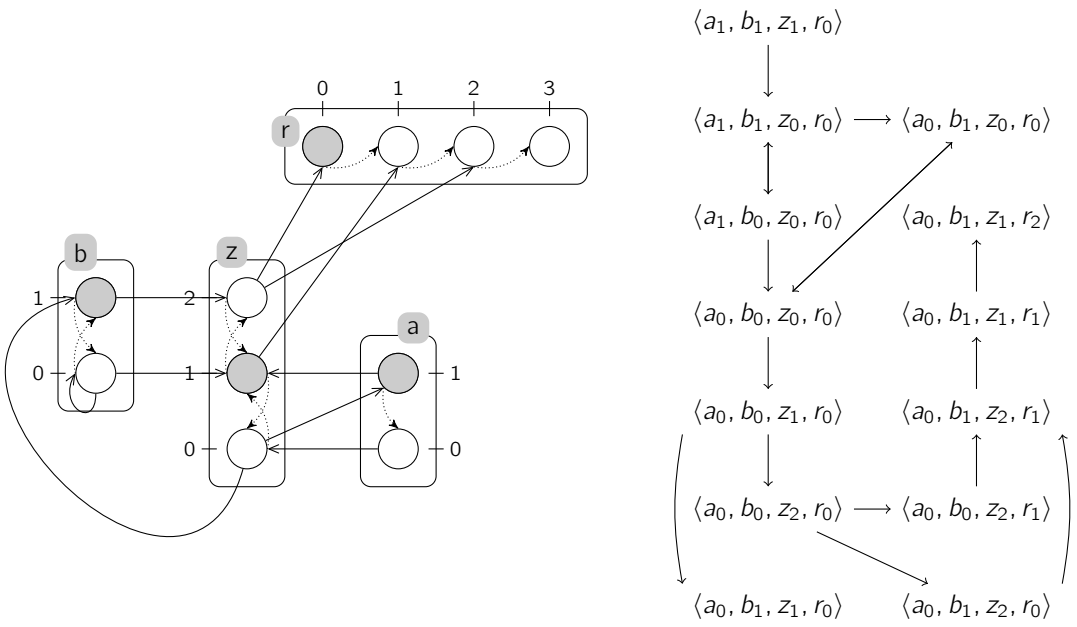


Figure 3.5 – (gauche) Exemple de Frappes de Processus arbitraires. Les processus grisés représentent l'état initial (ici, $\langle a_1, b_1, z_1, r_0\rangle$). (droite) Graphe des états en partant de l'état initial indiqué.

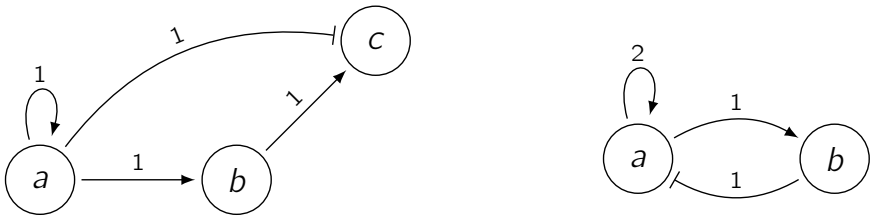
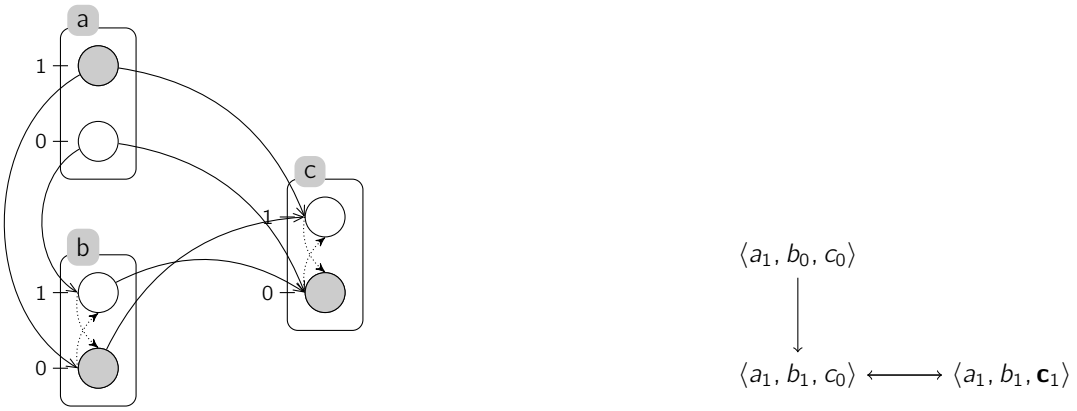


Figure 3.6 – Graphes des interactions de deux RRB élémentaires : (gauche) Boucle d'anticipation incohérente ; (droite) *Pseudomonas æruginosa*. Les arcs se terminant par une flèche représentent les régulations positives ; ceux se terminant par une barre représentent les régulations négatives. Le libellé des arcs représente le seuil de ces régulations.



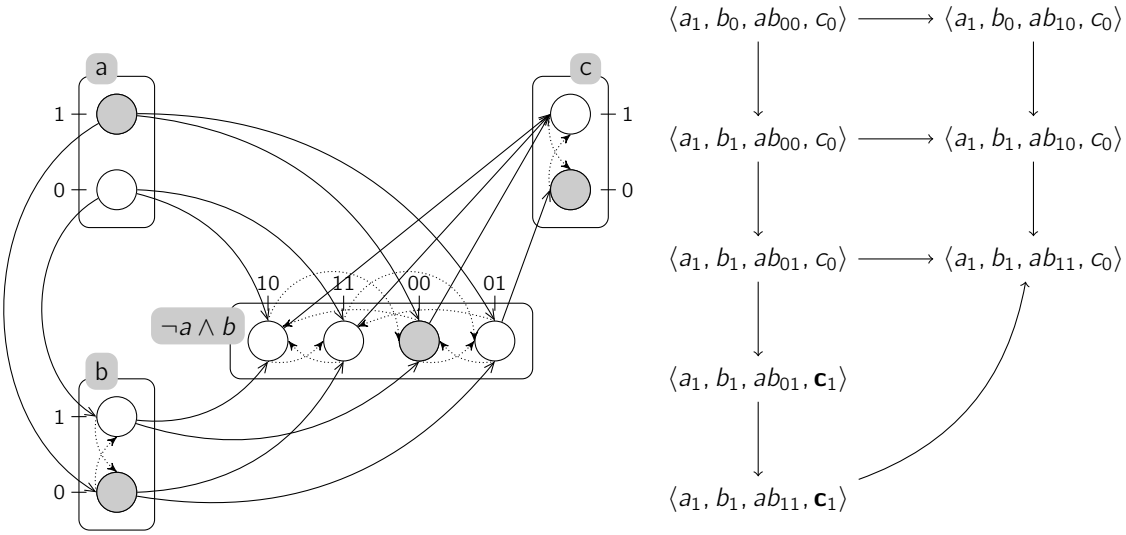


Figure 3.7 – Frappes de Processus (haut) de la dynamique généralisée du graphe des interactions de la boucle d'anticipation incohérente présentée dans la figure 3.6(gauche) ; (bas) issues d'un raffinement des Frappes de Processus d'en haut. Le graphe des états engendré par les deux Frappes des Processus à partir de l'état initial (processus grisés) est dessiné à leur droite.

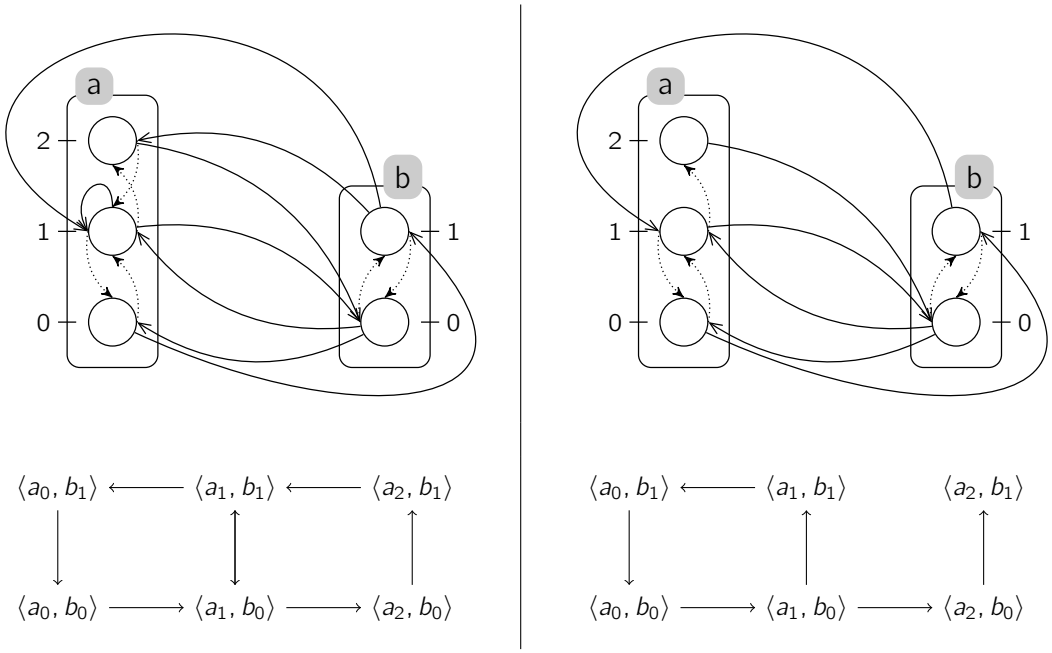


Figure 3.8 – Frappes de Processus (gauche) de la dynamique généralisée du graphe des interactions de *Pseudomonas æruginosa* présenté dans la figure 3.6(droite) ; (droite) issues d'un raffinement des Frappes de Processus de gauche. (bas) Graphes des états engendrés par les deux Frappes des Processus.

Boucle d'anticipation incohérente (figure 3.6(gauche)) : un composant a exerce un effet positif sur c au travers d'un composant b, alors qu'il exerce un effet négatif directement sur c. Ce type de motif de RRB est très courant et très étudié (Mangan & Alon, 2003). Nous proposons une modélisation par les Frappes de Processus via la dynamique généralisée et une dynamique raffinée par coopération entre a et c (réalisation de la fonction booléenne $c = \neg a \wedge b$). La figure 3.7 présente les deux Frappes de Processus obtenues, et donne le graphe des états accessibles depuis l'état où a est présent, et b et c absents. Le processus c_1 apparaît en gras dans le graphe des états. Dans le cas de la coopération entre a et b, nous choisissons ab_{00} comme état initial : cela correspond au cas où a vient tout juste d'arriver, et n'a donc pas encore mis à jour la sorte coopérative.

Pseudomonas æruginosa (figure 3.6(droite)) : ce RRB très simple permet d'illustrer le rôle des circuits de rétro-actions (Thomas & Kaufman, 2001). Il représente les interactions entre deux composants a et b au sein d'une bactérie. Ces deux composants engendrent un régime cyclique inoffensif qui peut dégénérer en une prolifération contrôlée par le composant a. Dans la figure 3.8, nous proposons une modélisation par les Frappes de Processus via la dynamique généralisée du graphe des interactions (gauche) et une dynamique raffinée en supprimant des actions ($a_1 \to a_1 \not\uparrow a_0$ et $b_1 \to a_2 \not\uparrow a_1$) pour obtenir le comportement voulu (droite).

3.8 Discussion

Dans ce chapitre, nous avons introduit les Frappes de Processus, un nouveau formalisme adapté à la modélisation des systèmes dynamiques complexes. Si elles sont ici appliquées à la modélisation des RRB, les Frappes de Processus pourraient tout aussi bien modéliser d'autres types de systèmes concurrents. Le lien formel entre les Frappes de Processus et d'autres formalismes classiques est détaillé dans le chapitre suivant.

Nous avons abordé la modélisation de la dynamique discrète des RRB en proposant plusieurs niveaux de précision dans leur spécification. Un niveau très abstrait, que nous appelons la dynamique généralisée, est, intuitivement, la dynamique la plus permissive compte tenu des contraintes imposées par la topologie du graphe des interactions d'un RRB. Enfin, en utilisant des mécanismes de raffinement, la spécification de fonctions de dynamique discrète est possible via la construction de sortes additionnelles de coopération entre des processus.

La coopération rappelle la notion biologique de formation de complexe, où plusieurs composants vont s'associer pour agir sur une cible donnée. La construction des coopérations en Frappes de Processus est théoriquement exponentielle selon le nombre de sortes coopérantes. Toutefois, comme nous l'avons montré, il est facile de réduire fortement le coût d'une telle construction en factorisant les coopérations.

Une inférence des paramètres discrets d'un RRB à partir des Frappes de Processus le modélisant a été montrée, ainsi que l'inférence d'un graphe des interactions. Ce procédé permet de retrouver une spécification « classique » d'un RRB après un travail de modélisation et de raffinement en utilisant les Frappes de Processus.

La création des Frappes de Processus a été motivée par la volonté de posséder un langage de spécification élémentaire. Comme montré dans le reste de ce manuscrit de thèse, cette simplicité va permettre de construire des analyses formelles peu coûteuses, ouvrant la possibilité d'étudier de très grands RRB (partie III), tout en gardant la possibilité d'obtenir des dynamiques précises (partie II).

Chapitre **4**

Sur l'Expressivité des Frappes de Processus

Afin de mieux comprendre la relation entre les Frappes de Processus et les formalismes standards pour les processus concurrents, nous étudions dans ce chapitre les traductions possibles des Frappes de Processus vers ces formalismes classiques et vice-versa en cherchant à établir des relations de bisimulations.
Cette étude se déroule en deux temps : tout d'abord nous montrons l'existence de traductions triviales des Frappes de Processus vers les Automates Finis Communicants, le π-Calcul et les Réseaux de Petri ; ces traductions offrent une relation de bisimulation entre les modèles obtenus. Dans un second temps, nous abordons la traductions des Réseaux Discrets (chapitre 2) puis des Automates Finis Communicants en Frappes de Processus. Afin de conserver une relation de bisimulation (faible) entre les modèles, nous augmentons au préalable l'expressivité des Frappes de Processus en ajoutant la notion de classes de priorités pour les actions. Les traductions obtenues possèdent une complexité exponentielle selon l'arité des interactions (c.-à-d. le nombre de processus entrant en jeu) et polynomiale selon leur nombre.

4.1 Préliminaires

Dans ce chapitre, nous discutons de l'expressivité des Frappes de Processus, formalisme introduit dans le chapitre 3. Cette notion d'expressivité est importante car elle donne une certaine vision du type de comportement qu'un formalisme peut décrire. L'expressivité d'un formalisme est généralement établie en le comparant à d'autres formalismes existants. Cette comparaison s'acquiert par la définition d'un encodage (une traduction) du formalisme étudié vers un autre formalisme et inversement et le choix des propriétés à conserver pour considérer le formalisme traduit comme équivalent au formalisme initial. Avec l'existence ou l'absence de tels encodages, nous obtenons un classement entre les formalismes selon leur expressivité, et plus précisément, selon une définition d'équivalence choisie : si il existe un encodage de A en B vérifiant l'équivalence voulue, alors B est au moins aussi expressif que A, et réciproquement.

Étant donné un modèle M_A exprimé dans un formalisme A, nous notons $M_B = \mathrm{B}(M_A)$ sa traduction dans le formalisme B. Nous supposons que la sémantique de A (resp. B) induit une relation de transition $\rightarrow_A$ (resp. $\rightarrow_B$) entre deux états de M_A (resp. de M_B). L'encodage de A vers B produit également une relation entre les états de M_A et M_B : étant donné un état s de M_A, nous notons $[\![s]\!]$ l'état correspondant dans M_B ; de même, étant donné un état x de M_B, nous notons

$[\![x]\!]$ l'état correspondant dans M_A.

Typiquement, nous allons vérifier l'existence d'une relation de bisimulation $\mathcal{R}$ entre les états de M et de B(M) telle que pour tout état s de M, $(s, [\![s]\!]) \in \mathcal{R}$, où $\mathcal{R}$ vérifie : si il existe s' tel que $(s, x) \in \mathcal{R}$ et $s \rightarrow_A s'$, alors il existe x' tel que $x \rightarrow_B x'$ et $(s', x') \in \mathcal{R}$, et réciproquement. À défaut, nous chercherons une relation de bisimulation faible : si $(s, x) \in \mathcal{R}$ et $s \rightarrow_A s'$, alors il existe x' tel que $x \rightarrow_B^* x'$ et $(s', x') \in \mathcal{R}$, et réciproquement, où $\rightarrow_B^*$ dénote une suite finie de transitions $\rightarrow_B$.

En guise de préambule, nous traitons la comparaison entre les Frappes de Processus et les Automates Finis. Un Automate Fini est défini par un ensemble d'états S et une relation de transition $\rightarrow_{AF} \subset S \times S$; nous notons $x \rightarrow_{AF} x'$ si la transition de l'état $x \in S$ vers $x' \in S$ est définie dans l'automate. Nous remarquons tout d'abord que toutes Frappes de Processus $(\Sigma, L, \mathcal{H})$ peuvent être exprimées sous la forme d'un Automate Fini $(S, \rightarrow_{AF})$: nous posons $S = L$ et $\forall s, s' \in L, s \rightarrow_{AF} s' \Leftrightarrow \exists h \in \mathcal{H}, s \cdot h = s'$; l'encodage et le décodage entre les états de S et de L est la fonction identité. Il est trivial de remarquer que nous obtenons ainsi une bisimulation entre les Frappes de Processus et leur traduction en Automate Fini.

De même, la traduction d'un Automate Fini en Frappes de Processus est directe : nous créons une seule sorte a et un processus de sorte a par état dans S. Pour chaque transition de l'automate nous créons une auto-frappe du processus de départ pour bondir au processus d'arrivée. Nous obtenons alors les Frappes de Processus $(\Sigma, L, \mathcal{H})$ avec $\Sigma = \{a\}$, $L = L_a = \{a_x \mid x \in S\}$, et $\mathcal{H} = \{a_x \rightarrow a_x \Rsh a_{x'} \mid x \rightarrow_{AF} x'\}$. Cette traduction est linéaire selon le nombre d'états et de transitions de l'Automate Fini et il est facile de vérifier que nous obtenons également une bisimulation entre un Automate Fini et sa traduction en Frappes de Processus.

Ainsi, les Frappes de Processus et les Automates Finis possèdent la même expressivité. Étant donné un modèle exprimé dans un langage formel résultant en un ensemble fini d'états et de transitions, il est donc possible d'obtenir des Frappes de Processus décrivant un comportement identique.

Toutefois, nous établissons plusieurs critiques sur cette approche : obtenir des Frappes de Processus agissant sur une seule sorte est sans intérêt et va à l'encontre de l'essence de ce formalisme qui repose sur la séparation des processus en différentes sortes ; de plus la transformation d'un modèle mettant en relation des composants agissant en parallèle (exprimés dans un langage formel quelconque) en un ensemble complet de transitions (c.-à-d. en Automate Fini) est d'une complexité généralement exponentielle selon le nombre de composants.

Afin d'obtenir une vision plus précise des relations entre divers formalismes pour les processus concurrents, des contraintes supplémentaires peuvent être requises sur la définition de la fonction d'encodage (voir par exemple (Palamidessi, 2003)). Dans le cadre de notre étude, focalisée sur les modèles engendrant un nombre fini d'états, nous nous concentrons sur des encodages possédant une complexité réduite comparée à la complexité du calcul de l'ensemble complet des transitions. En pratique, les encodages employés conservent la structure générale du modèle de départ : à chaque composant du modèle de départ sera associé un composant du modèle d'arrivée. Nous nous autorisons toutefois l'ajout de composants supplémentaires lors de l'encodage, si nécessaire.

Dans un premier temps, nous nous intéressons à la traduction des Frappes de Processus dans les formalismes classiques des Automates Finis Communicants, du π-Calcul et des Réseaux de Petri (section 4.2). Ces traductions sont toutes triviales et préservent totalement la structure des Frappes de Processus. Ceci montre que les Frappes de Processus sont une restriction d'un grand nombre de formalismes pour les processus concurrents.

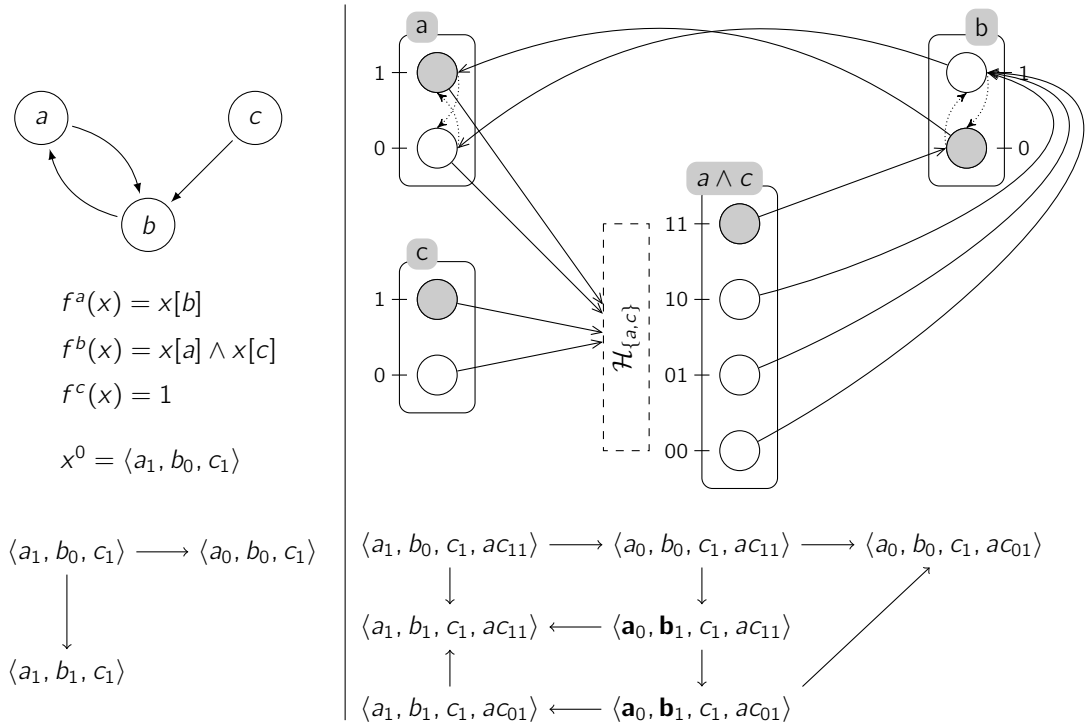


Figure 4.1 – (droite) Frappes des Processus résultant de la modélisation du Réseau Discret présenté à (gauche) avec leur graphe respectif des états accessibles depuis $\langle a_1, b_0, c_1 \rangle$. Les différences majeures entre la dynamique des Frappes de Processus et du Réseau Discret sont en gras.

La section 4.3 aborde ensuite la traduction des Réseaux Discrets (chapitre 2) et des Automates Finis Communicants en Frappes de Processus. Nous avons déjà exhibé une construction des Réseaux Discrets en Frappes de Processus dans la section 3.5 page 30 : cette construction ajoute une sorte « coopérative » pour chaque composant afin d'encoder la fonction associée. Comme discuté dans cette section 3.5, ce procédé ajoute un décalage temporel dans l'application d'une fonction discrète : la sorte coopérative est mise à jour par les composants, puis, elle frappe le processus actif de la sorte associée pour le faire bondir vers le processus représentant la nouvelle valeur de la fonction discrète. Il est important de noter que ceci fait perdre la bisimulation dans le cas général, la figure 4.1 en est un exemple : alors que le couple de processus $\langle a_0, b_1 \rangle$ n'est pas atteignable depuis l'état $\langle a_1, b_0, c_1 \rangle$ du Réseau Discret, les Frappes de Processus correspondantes permettent son atteinte, via l'état $\langle a_0, b_0, c_1, ac_{11} \rangle$ où la sorte coopérative ac est déphasée par rapport à l'état des sortes a et c.

Nous définissons alors dans la sous-section 4.3.1 les Frappes de Processus avec Priorités, où les actions sont divisées en classes de priorités (tant qu'une action d'une priorité $k-1$ est possible, aucune action de priorité supérieure à k n'est jouable). Nous montrons que ceci augmente l'expressivité des Frappes de Processus en exhibant des traductions des Réseaux Discrets et Automates Finis Communicants offrant une bisimulation faible et une complexité seulement exponentielle en l'arité des interactions (et polynomiale en leur nombre). L'intérêt des classes de priorités dans les Frappes de Processus est également discuté dans le cadre de l'analyse quantitative des Frappes de Processus dans le chapitre 12.

4.2 Les Frappes de Processus : une Restriction de Nombreux Formalismes

Nous traitons dans cette section la traduction des Frappes de Processus en Automates Finis Communicants (sous-section 4.2.1), en π-Calcul (sous-section 4.2.2) et en Réseau de Petri (sous-section 4.2.3), trois formalismes très classiques utilisés pour la modélisation de processus concurrents.

Les traductions obtenues sont triviales et assurent une relation de bisimulation entre les Frappes de Processus et le modèle traduit. La figure 4.2 illustre la traduction d'un exemple de Frappes de Processus dans ces trois formalismes, et montre une forte similarité structurelle entre les modèles obtenus. Les notations graphiques employées pour chaque formalisme sont détaillées dans leurs sous-sections respectives.

4.2.1 Vers les Automates Finis Communicants

Les Frappes de Processus peuvent être considérées comme une classe particulière des Automates Finis Communicants (Brand & Zafiropulo, 1983) où au plus deux automates (les sortes) partagent un libellé de synchronisation (une action), et un et un seul automate change d'état (de processus) à chaque synchronisation (jeu d'une action). Nous donnons ici une définition des Automates Finis Communicants et présentons formellement la traduction des Frappes de Processus dans ce formalisme.

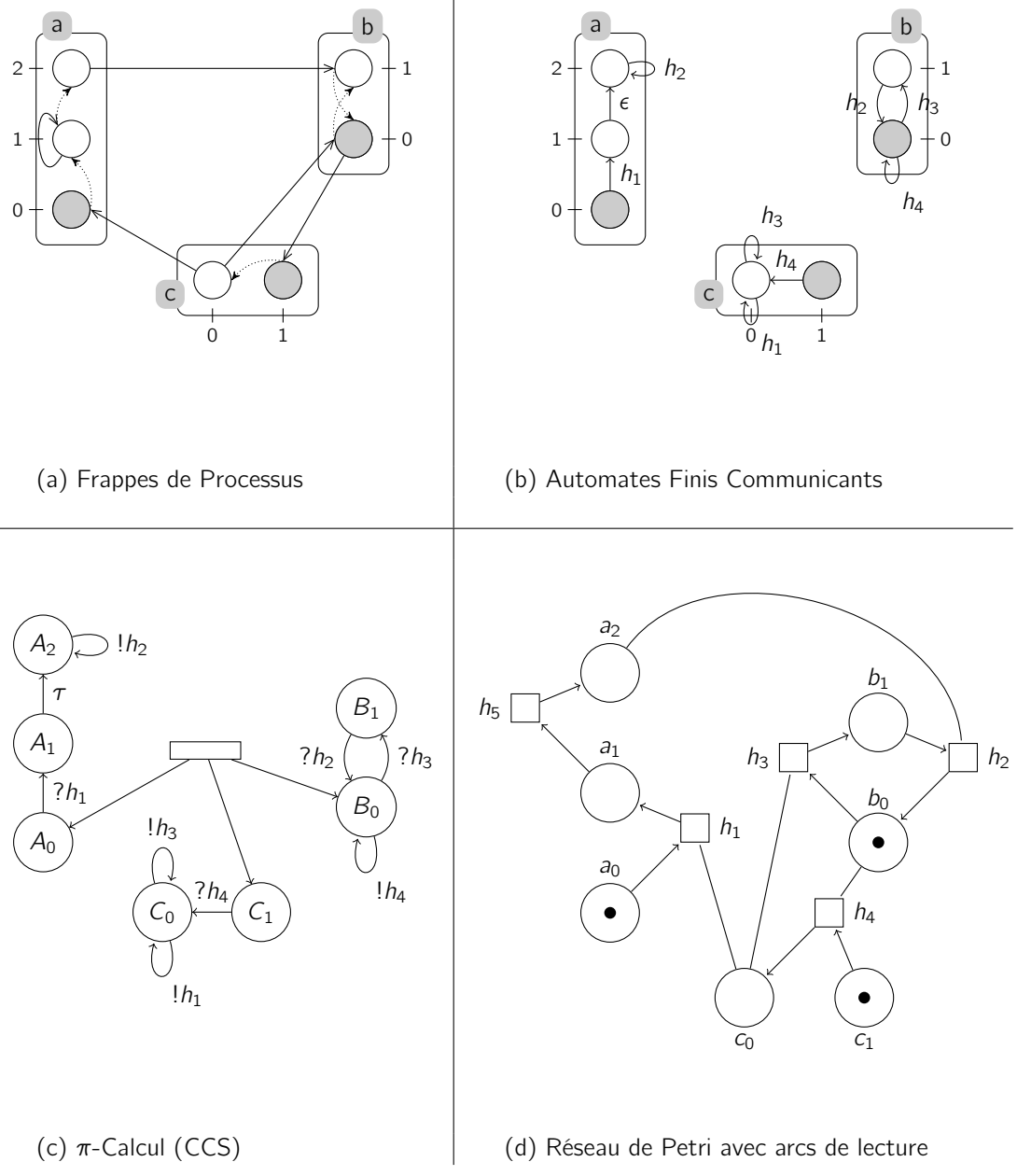
La définition 4.1 présente une formalisation des Automates Finis Communicants inspirée de (Brand & Zafiropulo, 1983). Les Automates Finis Communicants regroupent un nombre fini d'automates $a^1, \ldots, a^n$, chacun possédant un nombre fini d'états S_{a^m}. Deux types d'actions sont alors possibles : les transitions internes, libellées ϵ : un automate a effectue une transition depuis l'état a_i vers a_j, notée $a_i \xrightarrow{\epsilon} a_j$; et les transitions synchronisées par un libellé $\ell \in \mathcal{L}$: si au moins deux automates possèdent une transition libellée par ℓ depuis leur état courant, alors tous les automates possédant une transition libellée ℓ depuis leur état actuel appliquent cette transition de manière synchrone. Le nombre d'automates appliquant simultanément une transition est appelé *arité de l'interaction* ; l'arité maximale d'une interaction peut être déduite de la définition T des transitions.

Définition 4.1 (Automates Finis Communicants)**.** Un ensemble d'Automates Finis Communicants est défini par un quadruplet $(\mathcal{A}, S, \mathcal{L}, T)$, avec :
- $\mathcal{A} = \{a^1, \ldots, a^n\}$, l'ensemble fini des automates communicants ;
- $S = \prod_{a \in \mathcal{A}} S_a$, avec $\forall a \in \mathcal{A}$, $S_a = \{a_1, \ldots, a_{l_a}\}$, l'ensemble fini des états de l'automate a ;
- $\mathcal{L} = \{\ell_1, \ldots, \ell_m\}$, l'ensemble fini des libellés de synchronisation ;
- $T = \bigcup_{a \in \mathcal{A}} T_a$, où, pour tout automate $a \in \mathcal{A}$, $T_a = \{a_i \xrightarrow{\ell} a_j, \cdots \mid a_i, a_j \in S_a \wedge \ell \in \mathcal{L} \cup \{\epsilon\}\}$, l'ensemble fini des transitions de a.

Soient $x, x' \in S$ deux états globaux, nous notons $x \rightarrow_{AFC} x'$ une transition de x vers x', où :

$$x \rightarrow_{AFC} x' \iff \begin{cases} \exists a \in \mathcal{A}, x[a] \xrightarrow{\epsilon} x'[a] \in T_a \wedge \forall b \in \mathcal{A}, b \neq a, x[b] = x'[b] & \text{ou,} \\ \exists \ell \in \mathcal{L}, as = \{a \in \mathcal{A} \mid \exists a_j \in S_a, x[a] \xrightarrow{\ell} a_j \in T_a\}, |as| \geq 2 \\ \quad \wedge \forall a \in as, x[a] \xrightarrow{\ell} x'[a] \in T_a \wedge \forall b \in \mathcal{A} \setminus as, x[b] = x'[b] \end{cases}$$

où $x[a]$ est l'état de l'automate a dans x.



(a) Frappes de Processus

(b) Automates Finis Communicants

(c) π-Calcul (CCS)

(d) Réseau de Petri avec arcs de lecture

Figure 4.2 – Traduction d'un exemple de Frappes de Processus (a) en Automates Finis Communicants (b), en π-Calcul (c) et en Réseau de Petri avec arcs de lecture (d).

Nous notons **AFC**$(\Sigma, L, \mathcal{H})$ la traduction en Automates Finis Communicants des Frappes de Processus $(\Sigma, L, \mathcal{H})$ (définition 4.2) : à chaque sorte est associé un automate et chaque processus de cette sorte est associé à un état de l'automate correspondant. Deux types de transitions sont alors définis : les transitions internes correspondant aux auto-frappes (frappeur et cible identiques) : $a_i \to a_i \Rsh a_j$ devient $a_i \xrightarrow{\epsilon} a_j$, et les transitions synchronisées correspondant aux autres frappes : $h = b_k \to a_i \Rsh a_j$ devient le couple de transitions $a_i \xrightarrow{h} a_j$ et $b_k \xrightarrow{h} b_k$. Les états des Frappes de Processus et des Automates Finis Communicants étant les mêmes, le théorème 4.1 ci-dessous établit la relation de bisimulation entre $(\Sigma, L, \mathcal{H})$ et **AFC**$(\Sigma, L, \mathcal{H})$.

Définition 4.2 (**AFC**$(\Sigma, L, \mathcal{H})$)**.** Étant données des Frappes de Processus $(\Sigma, L, \mathcal{H})$, nous définissons les Automates Fini Communicants correspondants **AFC**$(\Sigma, L, \mathcal{H}) = (\mathcal{A}, S, \mathcal{L}, T)$, avec :
– $\mathcal{A} = \Sigma$; $S = L$;
– $\mathcal{L} = \{h \in \mathcal{H} \mid \mathsf{frappeur}(h) \neq \mathsf{cible}(h)\}$;
– $\forall a \in \Sigma, T_a = \{a_i \xrightarrow{\epsilon} a_j \mid h = a_i \to a_i \Rsh a_j \in \mathcal{H}\}$
$\cup \{a_i \xrightarrow{h} a_j \mid h = b_k \to a_i \Rsh a_j \in \mathcal{H}, \wedge b_k \neq a_i\}$
$\cup \{b_k \xrightarrow{h} b_k \mid h = b_k \to a_i \Rsh a_j \in \mathcal{H}, \wedge b_k \neq a_i\}$.
Étant donné un état $s \in L$ des Frappes de Processus, nous définissons $[\![s]\!] \in S$ l'état correspondant dans les Automates Finis Communicants obtenus : $s = [\![s]\!]$.

Théorème 4.1. *Étant donnés des Frappes de Processus* $(\Sigma, L, \mathcal{H})$ *et les Automates Finis Communicants* **AFC**$(\Sigma, L, \mathcal{H}) = (\mathcal{A}, S, \mathcal{L}, T)$ *:* $\forall s, s' \in L, s \to_{FP} s' \iff [\![s]\!] \to_{AFC} [\![s']\!]$.

Démonstration. Soit $s' = s \cdot h, h = b_k \to a_i \Rsh a_j \in \mathcal{H}$, alors par construction de **AFC**$(\Sigma, L, \mathcal{H})$, si $b_k = a_i$, $a_i \xrightarrow{\epsilon} a_j \in T_a$; si $b_k \neq a_i$, $\{a_i \xrightarrow{h} a_j, b_k \xrightarrow{h} b_k\} \subset T$; donc $[\![s]\!] \to_{AFC} [\![s']\!]$. De même, $[\![s]\!] \to_{AFC} [\![s']\!]$ implique
– $\exists a \in \mathcal{A}, \exists a_i \xrightarrow{\epsilon} a_j \in T_a$ ($\iff h = a_i \to a_i \Rsh a_j \in \mathcal{H}$), avec $[\![s]\!][a] = a_i$; ou
– $\exists a, b \in \mathcal{A}, \exists \ell \in \mathcal{L}, \exists$ un unique sous-ensemble $\{a_i \xrightarrow{\ell} a_j, b_k \xrightarrow{\ell} b_k\} \subset T$ ($\iff \ell = h = b_k \to a_i \Rsh a_j \in \mathcal{H}$), avec $[\![s]\!][a] = a_i$ et $[\![s]\!][b] = b_k$;
d'où $s' = s \cdot h$. □

La figure 4.2(b) donne la traduction en Automates Finis Communicants de l'exemple des Frappes de Processus de la figure 4.2(a). Nous avons choisi un formalisme graphique très proche de celui des Frappes de Processus : un automate est représenté par une boîte libellée par le nom de l'automate ; chaque état est représenté par un cercle libellé par le nom de l'état ; une transition est représentée par un arc orienté, libellé par ϵ si il s'agit d'une transition interne ou par un libellé référencé dans $\mathcal{L}$ si il s'agit d'une transition synchronisée.

4.2.2 Vers le π-Calcul

Le π-Calcul (Milner, 1989) est une algèbre de processus modélisant des processus communicant via des canaux mobiles. Ce formalisme a depuis connu de très nombreuses extensions, dont le π-calcul stochastique (Priami, 1995) que nous étudions en détail dans la partie II.

Nous présentons ici une définition informelle et incomplète du π-Calcul, nous envoyons le lecteur à la définition 5.1 page 62 pour une définition complète d'une variante stochastique synchrone.

Une expression en π-calcul est composée deux types d'objets : des processus, pouvant être définis indépendamment, et des canaux partagés par certains processus. Un processus A possède la capacité d'écrire (resp. de lire) sur un canal h pour ensuite devenir le processus A', noté $!h.A'$ (resp. $?h.A'$). Les entrées et les sorties sont des actions synchronisées, c.-à-d. un processus écrivant est bloqué jusqu'à la lecture d'un processus sur le même canal ; ces processus changent alors simultanément. Un processus peut également exécuter une action interne (notée τ), devenir le processus nul ($\mathbf{0}$), ou exécuter une action parmi plusieurs (par exemple $!h_1.A' + !h_2.A'$, noté également $\sum_{h \in \{h_1, h_2\}} !h.A'$). La composition parallèle de deux processus A_1 et A_2 se note $A_1 \mid A_2$ ou $\prod_{i \in \{1,2\}} A_i$.

Nous notons $\mathbf{\Pi C}(\Sigma, L, \mathcal{H})$ la traduction en π-Calcul des Frappes de Processus $(\Sigma, L, \mathcal{H})$ (définition 4.3) : pour chaque processus a_i des Frappes de Processus, un processus π-calcul A_i est défini comme effectuant un choix parmi trois types d'actions : une action interne vers A_j ($\tau.A_j$) pour chaque action $a_i \to a_i \Rsh a_j \in \mathcal{H}$; une écriture sur le canal h et le retour en A_i ($!h.A_i$) pour chaque action $h = a_i \to b_j \Rsh b_k \in \mathcal{H}, a_i \neq b_j$; une lecture sur le canal h pour devenir A_j ($?h.A_j$) pour chaque action action $h = b_k \to a_i \Rsh a_j \in \mathcal{H}, b_k \neq a_i$.

Définition 4.3 ($\mathbf{\Pi C}(\Sigma, L, \mathcal{H})$)**.** Pour toute sorte $a \in \Sigma$ et tout processus $a_i \in L_a$, nous définissons le processus π-Calcul A_i comme suit :

$$A_i ::= \sum_{\substack{h = a_i \to b_j \Rsh b_k \in \mathcal{H} \\ a \neq b}} !h.A_i + \sum_{\substack{h = b_j \to a_i \Rsh a_k \in \mathcal{H} \\ a \neq b}} ?h.A_k + \sum_{h = a_i \to a_i \Rsh a_k \in \mathcal{H}} \tau.A_k$$

Étant donné un état $s \in L$ des Frappes de Processus, nous définissons $[\![s]\!]$ le processus π-Calcul correspondant comme étant la composition parallèle des processus présents dans s :

$$[\![s]\!] = \prod_{a \in \Sigma} A_i \qquad \text{si } s[a] = a_i \ .$$

Nous remarquons que cette traduction est très proche de la traduction des Frappes de Processus en Automates Finis Communicants donnée dans la définition 4.2. Ainsi, la correction de la traduction des Frappes de Processus en π-Calcul peut être simplement dérivée du théorème 4.1.

La figure 4.2(c) donne la traduction en π-Calcul des Frappes de Processus de la figure 4.2(a). Le formalisme graphique est celui défini dans (Phillips, Cardelli & Castagna, 2006) : un processus est représenté par un cercle ; une action est représentée par un arc libellé par l'action et orienté vers le processus suivant ; la composition parallèle de processus est représentée par un rectangle relié aux processus composés.

4.2.3 Vers les Réseaux de Petri

Les Réseaux de Petri (Merlin, 1974) sont largement utilisés pour la modélisation et la vérification des systèmes concurrents, et connaissent également de nombreuses extensions. Nous utilisons ici une définition des Réseaux de Petri augmentés par des arcs de lecture (Vogler, Semenov & Yakovlev, 1998), formalisée dans la définition 4.4.

Un Réseau de Petri regroupe un ensemble fini de places et de transitions. Les transitions sont reliées aux places par des arcs ordinaires ou des arcs de lecture. Un marquage d'un Réseau de Petri (assimilable à un état) est alors un multi-ensemble de places. Une transition t est tirable si et seulement si toutes les places reliées aux arcs, ordinaires ou de lecture, entrant sur la transition (notées $\mathsf{pre}(t)$) sont présentes dans le marquage. L'application d'une transition t retire du marquage toutes les places reliées aux arcs ordinaires entrants (notées ${}^\bullet t$) et ajoute les places reliées aux arcs sortants (notées $t^\bullet$).

Définition 4.4 (Réseaux de Petri avec arcs de lecture)**.** Un réseau de Petri avec arcs de lecture est un quadruplet (S, T, W, R), avec :
– $S = \{p^1, \ldots, p^n\}$, l'ensemble fini des places ;
– $T = \{t^1, \ldots, t^m\}$, l'ensemble fini des transitions ;
– $W \subseteq S \times T \cup T \times S$, l'ensemble des arcs (ordinaires) ;
– $R \subseteq S \times T$, l'ensemble des arcs de lecture.
Un *marquage* M est un multi-ensemble de places. La transition entre deux marquages M et M' est définie de la façon suivante :

$$M \to_{RP} M' \iff \exists t \in T, \mathsf{pre}(t) \subset M \wedge M' = (M \uplus t^\bullet) \uplus {}^\bullet t$$

où $\mathsf{pre}(t) = \{p \in S \mid (p,t) \in W \cup R\}$, $t^\bullet = \{p \in S \mid (p,t) \in W\}$, et ${}^\bullet t = \{p \in S \mid (t,p) \in W\}$.

Nous notons $\mathbf{RP}(\Sigma, L, \mathcal{H})$ la traduction en Réseau de Petri des Frappes de Processus $(\Sigma, L, \mathcal{H})$ (définition 4.5) : à chaque processus est associée une place et à chaque action de $\mathcal{H}$ est associée une transition. Pour chaque action $h = b_k \to a_i \Rsh a_j \in \mathcal{H}$, nous ajoutons un arc ordinaire de la place a_i vers la transition h et un arc ordinaire de la transition h vers la place a_j ; si $b_k \neq a_i$, nous ajoutons également un arc de lecture entre la place b_k et la transition h. Nous remarquons que cette traduction aurait pu se faire sans arcs de lecture en ajoutant alors un arcs depuis b_k vers h puis de h vers b_k. Un état s des Frappes de Processus est traduit comme étant le marquage regroupant un exemplaire de chaque place correspondant à un processus présent dans s.

Définition 4.5 ($\mathbf{RP}(\Sigma, L, \mathcal{H})$)**.** Étant données des Frappes de Processus $(\Sigma, L, \mathcal{H})$, nous définissons le Réseau de Petri correspondant $\mathbf{RP}(\Sigma, L, \mathcal{H}) = (S, T, W, R)$, avec :
– $S = \bigcup_{a \in \Sigma} L_a$;
– $T = \mathcal{H}$;
– $W = \{(a_i, h) \mid h \in \mathcal{H} \wedge \mathsf{cible}(h) = a_i\} \cup \{(h, a_j) \mid h \in \mathcal{H} \wedge \mathsf{bond}(h) = a_j\}$;
– $R = \{(b_k, h) \mid h \in \mathcal{H} \wedge \mathsf{frappeur}(h) = b_k \wedge \mathsf{frappeur}(h) \neq \mathsf{cible}(h)\}$.
Étant donné un état s des Frappes de Processus, nous définissons $[\![s]\!]$ le marquage correspondant : $[\![s]\!] = [s[a] \mid a \in \Sigma]$.

Partant d'un marquage encodant un état des Frappes de Processus, le lemme 4.1 ci-dessous montre que tous les marquages accessibles correspondent à un état des Frappes de Processus ; le théorème 4.2 établit alors la relation de bisimulation entre $(\Sigma, L, \mathcal{H})$ et $\mathbf{RP}(\Sigma, L, \mathcal{H})$.

Lemme 4.1. *Étant donnés des Frappes de Processus* $(\Sigma, L, \mathcal{H})$ *et le Réseau de Petri* $\mathbf{RP}(\Sigma, L, \mathcal{H})$*,* $\forall s \in L, [\![s]\!] \to_{RP} M' \Longrightarrow \exists s' \in L, [\![s']\!] = M'$.

Démonstration. Par construction de W, toute transition échange deux processus de même sorte. □

Théorème 4.2. *Étant donnés des Frappes de Processus* $(\Sigma, L, \mathcal{H})$ *et le réseau de Petri* $\mathbf{RP}(\Sigma, L, \mathcal{H})$, $\forall s, s' \in L, s \rightarrow_{FP} s' \iff [\![s]\!] \rightarrow_{RP} [\![s']\!]$.

Démonstration. Par construction, une transition du réseau de Petri est tirable dans $[\![s]\!]$ si et seulement si l'action en Frappes de Processus correspondante est jouable dans s ($\mathsf{pre}(h) = \{a_i\} \Leftrightarrow h = a_i \rightarrow a_i \Rsh a_j$ et $\mathsf{pre}(h) = \{b_k, a_i\} \Leftrightarrow h = b_k \rightarrow a_i \Rsh a_j, b_k \neq a_i$) ; et son application donne un marquage équivalent à l'état obtenu dans les Frappes de Processus ($^\bullet h = \mathsf{cible}(h) \wedge h^\bullet = \mathsf{bond}(h)$). □

La figure 4.2(d) donne la traduction en Réseau de Petri des Frappes de Processus de la figure 4.2(a). Les places sont représentées par des cercles ; les transitions par des carrés ; les arcs ordinaires sont orientés, contrairement aux arcs de lecture. Un jeton est placé sur une place pour chaque exemplaire de cette place dans le marquage.

4.3 Augmenter l'Expressivité des Frappes de Processus

Afin d'assurer une traduction des Réseaux Discrets et des Automates Finis Communicants en Frappes de Processus préservant une relation de bisimilarité faible, nous étendons le formalisme des Frappes de Processus, défini à la page 25, avec la séparation des actions en classes de priorité (sous-section 4.3.1).

Nous montrons alors une traduction des Réseaux Discrets en Frappes de Processus avec 2 Priorités possédant un coût exponentiel selon l'arité des interactions et polynomial selon leur nombre. Cette traduction assure une relation de bisimilarité faible entre un Réseau Discret et sa traduction en Frappes de Processus (sous-section 4.3.2).

Cette première traduction est alors généralisée pour encoder les Automates Finis Communicants en Frappes de Processus avec 3 Priorités. Cette traduction possède également une complexité exponentielle selon l'arité des interactions et polynomiale selon leur nombre. Nous montrons également la présence d'une relation de bisimilarité entre les Automates Finis Communicants et leur traduction en Frappes de Processus (sous-section 4.3.3).

4.3.1 Les Frappes de Processus avec Priorités

Les Frappes de Processus formalisées par la définition 3.1 page 25 sont généralisées par les Frappes de Processus avec k Priorités de la définition 4.6 : les actions sont réparties au sein de k ensembles $\mathcal{H}^1, \ldots, \mathcal{H}^k$. Les actions appartenant à $\mathcal{H}^n$, $n \in [1; k]$ sont dites de priorité n ; les priorités sont classées par ordre décroissant : 1 est la classe de priorité la plus forte et k la plus faible. La sémantique des transitions est modifiée en accord avec la définition 4.7 : une action de priorité n est jouable dans s seulement si aucune action de priorité $< n$ est jouable. Les Frappes de Processus classiques correspondent exactement aux Frappes de Processus avec 1 Priorité.

Définition 4.6 (Frappes de Processus avec k Priorités)**.** Les *Frappes de Processus avec k Priorités* sont définies par un $2+k$-uplet $(\Sigma, L, \mathcal{H}^1, \ldots, \mathcal{H}^k)$, où :
- $\Sigma = \{a, b, \ldots\}$ est l'ensemble fini et dénombrable des sortes,
- $L = \prod_{a\in\Sigma} L_a$ est l'ensemble de états, où $L_a = \{a_0 \ldots a_{l_a}\}$ est l'ensemble fini et dénombrable des processus de sorte $a \in \Sigma$, avec l_a un entier positif. On pose $a \neq b \Rightarrow a_i \neq b_j \; \forall (a_i, b_j) \in L_a \times L_b$,
- $\forall n \in [1; k], \mathcal{H}^n = \{a_i \to b_j \Rsh b_k, \cdots \mid (a, b) \in \Sigma^2 \wedge (a_i, b_j, b_k) \in L_a \times L_b \times L_b \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ est l'ensemble fini des *actions de priorité n*.

Définition 4.7 (Sémantique des Frappes de Processus avec k Priorités)**.** Étant donnés des Frappes de Processus avec k Priorités $(\Sigma, L, \mathcal{H}^1, \ldots, \mathcal{H}^k)$ et deux états $s, s' \in L$:

$$\begin{aligned} s \to_{FPP} s' \Longleftrightarrow & \exists n \in [1; k] \wedge \exists h \in \mathcal{H}^n, s \cdot h = s' \\ & \wedge \forall h' \in \mathcal{H}^1 \cup \cdots \cup \mathcal{H}^{n-1}, \mathsf{frappeur}(h') \notin s \vee \mathsf{cible}(h') \notin s \ . \end{aligned}$$

En notant $\mathbf{Sce}(\Sigma, L, \mathcal{H}^1, \ldots, \mathcal{H}^k)$ l'ensemble des scénarios (définition 3.2 page 26) possibles pour les Frappes de Processus avec k Priorités $(\Sigma, L, \mathcal{H}^1, \ldots, \mathcal{H}^k)$, nous obtenons la propriété 4.1 : en augmentant le nombre de classes de priorités, nous réduisons (pas nécessairement strictement) les scénarios possibles.

Propriété 4.1. $\mathbf{Sce}(\Sigma, L, \mathcal{H}^1, \ldots, \mathcal{H}^k) \subseteq \mathbf{Sce}(\Sigma, L, \mathcal{H}^1 \cup \cdots \cup \mathcal{H}^k)$

Démonstration. Pour tout état $s \in L$, si une action est jouable pour des Frappes de Processus avec k Priorités, alors cette action est jouable dans les Frappes de Processus regroupant toutes les frappes $\mathcal{H}^1, \ldots, \mathcal{H}^k$. □

4.3.2 Bisimulation Faible des Réseaux Discrets

Les Réseaux Discrets décrivent l'évolution discrète d'un nombre fini de composants, et sont notamment utilisés pour la modélisation des RRB (voir chapitre 2). La définition 4.8 résume leur formalisation avec une dynamique asynchrone. Un Réseau Discret regroupe un nombre fini de composants i munis d'une fonction de mise à jour f^i. Un composant i possède un domaine discret fini $\mathbb{F}^i$, typiquement $\mathbb{F}^i = [0; l_i]$, l_i étant la valeur maximale de i. La fonction de mise à jour f^i associe à l'état global du réseau la valeur suivante du composant i. Cette valeur dépend généralement d'un sous-ensemble de composants que nous notons $\mathsf{dep}(f^i)$. Enfin, une transition entre deux états du réseau $x \to_{RD} x'$ est gouvernée par l'application d'une et une seule fonction de mise à jour.

Définition 4.8 (Réseau Discret $(\mathbb{F}, \langle f^1, \ldots, f^n\rangle)$)**.** Un Réseau Discret $(\mathbb{F}, \langle f^1, \ldots, f^n\rangle)$ est une collection de n applications $f^i : \mathbb{F} \mapsto \mathbb{F}^i$ associant à un état du réseau la valeur suivante de la i^{e} composante, où $\mathbb{F}^i = [0; l_i]$ est le domaine de la i^{e} composante et $\mathbb{F} = \mathbb{F}^1 \times \cdots \times \mathbb{F}^n$ le domaine du réseau. La transition $\to_{RD}$ entre deux états $x, x' \in \mathbb{F}$ est définie de la façon suivante :

$$x \to_{RD} x' \Longleftrightarrow \exists i \in [1; n], f^i(x) = x'[i] \wedge \forall j \in [1; n], j \neq i, x[j] = x'[j] \ ,$$

où $x[i]$ est la valeur de la i^{e} composante de x. Nous notons $\mathsf{dep}(f^i) \subseteq \{1, \ldots, n\}$ l'ensemble des composantes dont dépend la valeur de f^i (alors, $\forall x, x' \in \mathbb{F}$ tel que $\forall j \in \mathsf{dep}(f^i), x[j] = x'[j]$, nous avons $f^i(x) = f^i(x')$).

Nous notons $\mathbf{PH}(\mathbb{F}, \langle f^1, \ldots, f^n \rangle)$ la traduction d'un Réseau Discret $(\mathbb{F}, \langle f^1, \ldots, f^n \rangle)$ en Frappes de Processus avec 2 Priorités (définition 4.9). Cette traduction est similaire à celle proposée en section 3.5 à la différence de la séparation des actions en deux classes de priorité : les actions mettant à jour les sortes coopératives sont de priorité 1, les actions mettant à jour les sortes des composants sont de priorité 2. Une sorte correspond ainsi soit à un composant a^i soit à la fonction associée f^i. Les sortes des composants possèdent un processus par état possible du composant. Les sortes des fonctions possèdent un processus par état ς possible du m-uplet regroupant les composants dont dépend f^i : nous notons $\prod_{j \in \mathsf{dep}(f^i)} L_{a^j}$ l'ensemble de tous ces états, et f^i_ς un tel processus ; $m = |\mathsf{dep}(f^i)|$ est l'arité de la fonction. Un état s des Frappes de processus est alors traduit en un état $x = [\![s]\!]$ du Réseau Discret en copiant uniquement les états des sortes des composants.

Deux classes d'action sont ensuite définies : $\mathcal{H}^1$ regroupe les frappes des sortes des composants vers les sortes des fonctions afin de mettre à jour le processus reflétant leur processus courant : si f^i dépend de a^j, a^j_k frappe tous les processus de f^i où a^j_k n'est pas présent pour les faire bondir vers les processus où a^j_k est présent. $\mathcal{H}^2$ liste les actions appliquant les fonctions : f^i_ς frappe les processus de sorte a^i pour les faire bondir vers le processus $a^i_{k'}$, où k' est la valeur de $f^i([\![\varsigma]\!])$; nous utilisons ici un petit abus de notation qui consiste à donner à f^i uniquement les valeurs des composants dont il dépend.

Enfin, un état x du Réseau Discret est traduit en un état $s = (\!|x|\!)$ des Frappes de Processus en copiant les états des composants et en affectant aux sortes coopératives les processus correspondant à l'état des composants coopérants.

Définition 4.9 ($\mathbf{PH}(\mathbb{F}, \langle f^1, \ldots, f^n \rangle)$)**.** Nous définissons $\mathbf{PH}(\mathbb{F}, \langle f^1, \ldots, f^n \rangle) = (\Sigma, L, \mathcal{H}^1, \mathcal{H}^2)$ les Frappes de Processus avec 2 Priorités correspondant au Réseau Discret $(\mathbb{F}, \langle f^1, \ldots, f^n \rangle)$ comme suit :
- $\Sigma = \{a^1, \ldots, a^n\} \cup \{f^1, \ldots, f^n\}$;
- $L = \prod_{i \in [1;n]} L_{a^i} \times \prod_{i \in [1;n]} L_{f^i}$, où $L_{a^i} = \{a^i_0, \ldots, a^i_{l_i}\}$, et $L_{f^i} = \{f^i_\varsigma \mid \varsigma \in \prod_{j \in \mathsf{dep}(f^i)} L_{a^i}\}$ si $\mathsf{dep}(f^i) \neq \emptyset$, sinon $L_{f^i} = \{f^i_\emptyset\}$;
- $\mathcal{H}^1 = \{a^j_k \to f^i_\varsigma \Rsh f^i_{\varsigma'} \mid i \in [1;n] \wedge j \in \mathsf{dep}(f^i) \wedge a^j_k \in L_{a^j} \wedge f^i_\varsigma \in L_{f^i} \wedge \varsigma[a^j] \neq a^j_k \wedge \varsigma'[a^j] = a^j_k \wedge (\varsigma'[a^l] = \varsigma[a^l], \forall l \in [1;n], l \neq j)\}$, l'ensemble des frappes de priorité 1 mettant à jour les sortes coopératives.
- $\mathcal{H}^2 = \{f^i_\varsigma \to a^i_k \Rsh a^i_{k'} \mid i \in [1;n] \wedge f^i_\varsigma \in L_{f^i} \wedge a^i_k \in L_{a^i} \wedge k \neq k' \wedge f^i([\![\varsigma]\!]) = k'\}$, l'ensemble des frappes de priorité 2 mettant à jour les sortes des composants en fonction de la valeur de la fonction f^i associée, $[\![\varsigma]\!]$ étant défini ci-dessous.

Étant donné un état $s \in L$ des Frappes de Processus, nous définissons $[\![s]\!] = x$ l'état du réseau correspondant : $\forall i \in [1;n], s[a^i] = a^i_k \Rightarrow x[i] = k$. Cette définition se généralise aisément à un état partiel des Frappes de Processus.
Étant donné un état $x \in \mathbb{F}$ du Réseau Discret, nous définissons $(\!|x|\!) = s$ l'état correspondant des Frappes de Processus : $\forall i \in [1;n], x[i] = k \Rightarrow s[a^i] = a^i_k$ et $\forall i \in [1;n], s[f^i] = f^i_\varsigma$ avec $f^i_\varsigma \in L_{f^i}$ et $\forall j \in \mathsf{dep}(f^i), \varsigma[j] = s[a^j]$.

La traduction ainsi définie possède une complexité exponentielle en l'arité des interactions (due aux processus des sortes coopératives, c.-à-d. celles des fonctions). Toutefois, comme discuté dans la section 3.5, l'explosion combinatoire des processus des sortes coopératives peut être limitée en factorisant la fonction f^i (voir figure 3.3 page 32).

Le théorème 4.3 établit une relation de bisimulation faible entre un Réseau Discret et sa tra-

duction en Frappes de Processus avec 2 Priorités. Intuitivement, les actions mettant à jour les sortes coopératives étant de priorité supérieure à celles mettant à jour les sortes des composants, ces dernières ne sont actionnées que lorsque toutes les sortes coopératives sont à jour, assurant la cohérence de l'application des fonctions discrètes.

Théorème 4.3 $((\mathbb{F}, \langle f^1, \ldots, f^n \rangle) \approx \mathbf{PH}(\mathbb{F}, \langle f^1, \ldots, f^n \rangle))$**.** *Étant donnés un Réseau Discret* $(\mathbb{F}, \langle f^1, \ldots, f^n \rangle)$ *et les Frappes de Processus avec* 2 *Priorités correspondantes* $\mathbf{PH}((\mathbb{F}, \langle f^1, \ldots, f^n \rangle)) = (\Sigma, L, \mathcal{H}^1, \mathcal{H}^2)$*, alors :*

1. $\forall x, x' \in \mathbb{F},\ x \to_{RD} x' \Longrightarrow (\!|x|\!) \to^*_{FPP} (\!|x'|\!)$*, où* $\to^*_{FPP}$ *est une suite finie de transitions* $\to_{FPP}$*.*
2. $\forall s, s' \in L,\ s \to_{FPP} s' \Longrightarrow [\![s]\!] = [\![s']\!] \vee [\![s]\!] \to_{RD} [\![s']\!]$.

Démonstration. (1) D'après la définition 4.8 $x \to_{RD} x' \Rightarrow \exists i \in [1;n], f^i(x) = x'[i] \wedge \forall j \in [1;n], i \neq j, x[j] = x'[j]$. Posons $f^i(x) = k'$, $x[i] = k$ et $\varsigma \in \prod_{j \in \mathsf{dep}(f^i)} L_{a^j}$ tel que $\forall j \in \mathsf{dep}(f^i), \varsigma[j] = a^j_{x[j]}$. D'après la définition 4.9, $h = f^i_\varsigma \to a^i_k \nearrow a^i_{k'} \in \mathcal{H}^2$. D'après la définition de $(\!|x|\!)$, $a^i_k \in (\!|x|\!)$ et $f^i_\varsigma \in (\!|x|\!)$; de plus il n'existe aucune frappe de $\mathcal{H}^1$ jouable dans $(\!|x|\!)$, donc h est jouable dans $(\!|x|\!)$. Nous notons cette transition $(\!|x|\!) \to_{FPP} (\!|x|\!) \cdot h$. Ensuite, seules les actions dont $a^i_{k'}$ est le frappeur sont jouables : elles sont toutes de priorité 1 et modifient uniquement les processus des sortes coopératives, donnant une suite de transitions toujours finie vers l'état où toutes les sortes coopératives reflètent l'état courant des composants de F : l'état $(\!|x'|\!)$.

(2) $s \to_{FPP} s'$ implique qu'il existe une action h jouable dans s telle que $s \cdot h = s'$. Si cette action est de priorité 1, alors par définition de $\mathcal{H}^1$, $[\![s]\!] = [\![s']\!]$. Si cette action est de priorité 2, alors $\forall i \in [1;n]$, si $s[f^i] = f^i_\varsigma$, alors, $\forall j \in \mathsf{dep}(f^i), \varsigma[a^j] = s[a^j]$. Soit $i \in [1;n]$ tel que $s[a^i] \neq s'[a^i]$ (i est unique pour cette transition). D'après la définition 4.9, si $s'[a^i] = a^i_{k'}$, nous avons nécessairement $f^i([\![s]\!]) = k'$, d'où $[\![s]\!] \to_{RD} [\![s']\!]$. □

Si nous reprenons l'exemple de la figure 4.1(droite) en affectant la priorité 1 aux actions vers la sorte coopérative $a \wedge c$, et la priorité 2 aux autres actions, nous obtenons le graphe de transitions suivant :

$$\begin{array}{ccccc} \langle a_1, b_0, c_1, ac_{11} \rangle & \longrightarrow & \langle a_0, b_0, c_1, ac_{11} \rangle & \longrightarrow & \langle a_0, b_0, c_1, ac_{01} \rangle \\ \downarrow & & & & \\ \langle a_1, b_1, c_1, ac_{11} \rangle & & & & \end{array}$$

montrant ici la faible bisimilarité avec le graphe des états obtenu avec le réseau discret de la figure 4.1(gauche).

4.3.3 Bisimulation Faible des Automates Finis Communicants

En s'inspirant de la traduction présentée dans la sous-section précédente, nous montrons ici une traduction des Automates Finis Communicants en Frappes de Processus avec 3 priorités.

La différence substantielle entre un Réseau Discret et des Automates Finis Communicants est la capacité de ces derniers d'opérer un changement synchrone d'état dans au moins 2 automates différents.

Nous pourrions résumer le rôle des classes de priorités des actions résultant de la traduction d'un Réseau Discret comme ceci :

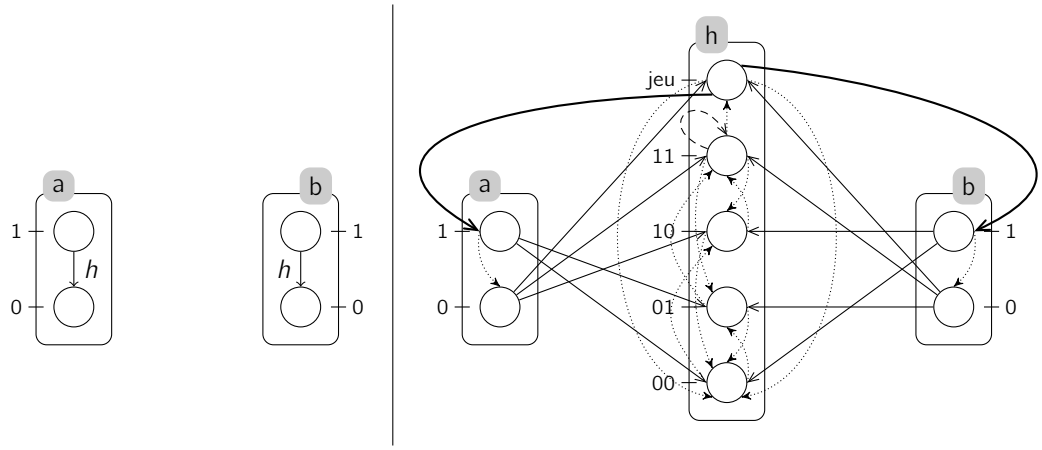


Figure 4.3 – (droite) Traduction des Automates Finis Communicants de (gauche) en Frappes de Processus avec 3 Priorités : les actions de priorité 1 ont leur frappe dessinée en ligne épaisse, celles de priorité 2 en ligne normale, et celle de priorité 3 en ligne discontinue (voir auto-frappe du processus h_{11}).

1. les actions de priorité 1 mettent à jour les sortes coopératives ;
2. les actions de priorité 2 choisissent et appliquent une transition.

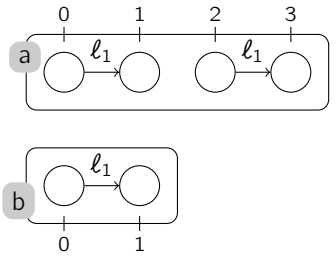
Notre traduction des Automates Finis Communicants nécessite une troisième classe de priorité :

1. les actions de priorité 1 appliquent une transition ;
2. les actions de priorité 2 mettent à jour les sortes coopératives ;
3. les actions de priorité 3 choisissent la transition à appliquer.

La figure 4.3 illustre ce procédé par la traduction d'une interaction entre a_1 et b_1 modifiant simultanément l'état des 2 automates. h est la sorte coopérative : le processus h_{11} correspond à a_1 présent et b_1 présent, le processus h_{10} à a_1 présent et b_1 absent, et de façon similaire pour les processus h_{01} et h_{00}. Le processus h_{jeu} est présent uniquement lorsque la transition h doit être jouée. Les actions partant de h_{jeu} sont ainsi de priorité 1, celles frappant h depuis a ou b de priorité 2 et celle bondissant en h_{jeu} de priorité 3.

Nous notons $\mathbf{PH}(\mathcal{A}, S, \mathcal{L}, T)$ les Frappes de Processus résultant de la traduction des Automates Finis Communicants $(\mathcal{A}, S, \mathcal{L}, T)$ (définition 4.10). Chaque automate est associé à une sorte unique comprenant un processus pour chaque état de l'automate. Étant donné un libellé de synchronisation ℓ, nous notons $\Sigma(\ell)$ l'ensemble des automates utilisant ce libellé. Nous créons alors une sorte (coopérative) pour chaque « groupe » $\upsilon^\ell \in \mathrm{g}(\mathcal{L})$, où υ est un m-uplet de processus de sortes différentes pouvant se synchroniser sur ℓ, m étant l'arité maximum de l'interaction ($m = |\Sigma(\ell)|$). La figure 4.4 illustre ce procédé : si $T = \{a_0 \xrightarrow{\ell_1} a_1, a_2 \xrightarrow{\ell_1} a_3, b_0 \xrightarrow{\ell_1} b_1\}$, alors $\mathrm{g}(\ell_1) = \{\langle a_0, b_0\rangle, \langle a_2, b_0\rangle\}$. Une telle sorte υ^ℓ possède alors $2^m + 1$ processus : 2^m processus pour noter l'absence/présence de chaque processus dans υ, notés υ^ℓ_ς avec $\varsigma \in \{0,1\}^m$; et 1 processus υ^ℓ_{jeu} dont la présence indique que la synchronisation entre les processus υ selon ℓ est en cours. L'encodage/décodage entre les états des Frappes de Processus et des Automates Finis Communicants est similaire à celui défini avec les Réseaux Discrets.

Les actions sont alors définies comme suit : (priorité 1) υ^ℓ_{jeu} frappe tous les processus contenus dans υ pour les faire bondir vers les processus résultant d'une synchronisation selon ℓ. (priorité 2)



$\Longrightarrow \mathrm{g}(\ell_1) = \{\langle a_0, b_0\rangle, \langle a_2, b_0\rangle\}$

Figure 4.4 – Exemple d'Automates Finis Communicants : illustration des groupes de libellés séparant les processus de même sorte ne pouvant se synchroniser sur un même libellé.

les processus des sortes des automates frappent les processus des sortes coopératives afin de mettre à jour leurs états reflétés ; les processus de sorte dans $\Sigma(\ell)$ frappent également les processus υ^ℓ_{jeu} pour désactiver l'application (en les faisant bondir vers le processus $\upsilon^\ell_{0^m}$, où 0^m est le vecteur de m 0. (priorité 3) les actions jouent une transition interne, ou font bondir un processus υ^ℓ_ς en υ^ℓ_{jeu} seulement si ς contient deux composantes à 1 (au moins deux processus à synchroniser).

Définition 4.10 (**PH**$(\mathcal{A}, S, \mathcal{L}, T)$)**.** Nous définissons **PH**$(\mathcal{A}, S, \mathcal{L}, T) = (\Sigma, L, \mathcal{H}^1, \mathcal{H}^2, \mathcal{H}^3)$ les Frappes de Processus avec 3 Priorités correspondant aux Automates Finis Communicants $(\mathcal{A}, S, \mathcal{L}, T)$ comme suit, où $\Sigma(\ell) = \{a \in \mathcal{A} \mid \exists a_i \xrightarrow{\ell} a_j \in T_a\}$:
- $\Sigma = \mathcal{A} \cup \mathrm{g}(\mathcal{L})$, où $\mathrm{g}(\mathcal{L}) = \{\upsilon^\ell \mid \ell \in \mathcal{L} \wedge \upsilon \in \prod_{a\in\Sigma(\ell)} {}^\bullet T_a(\ell)\}$,
  avec ${}^\bullet T_a(\ell) = \{a_i \in S_a \mid \exists a_j \in S_a, a_i \xrightarrow{\ell} a_j \in T_a\}$.
- $L = \prod_{a\in\mathcal{A}} S_a \times \prod_{\upsilon^\ell\in\mathrm{g}(\mathcal{L})}\{\upsilon^\ell_\varsigma \mid \varsigma \in \{0,1\}^{|\Sigma(\ell)|}\} \cup \{\upsilon^\ell_{jeu}\}$.
- $\mathcal{H}^1 = \{\upsilon^\ell_{jeu} \to a_i \Rsh a_j \mid \upsilon^\ell \in \mathrm{g}(\mathcal{L}) \wedge a \in \Sigma(\ell) \wedge \upsilon[a] = a_i \wedge a_i \xrightarrow{\ell} a_j \in T_a\}$, l'ensemble des frappes de priorité 1 appliquant une transition libellée ℓ.
- $\mathcal{H}^2 = \{a^i \to \upsilon^\ell_\varsigma \Rsh \upsilon^\ell_{\varsigma'} \mid \upsilon^\ell \in \mathrm{g}(\mathcal{L}) \wedge a \in \Sigma(\ell) \wedge \varsigma \in \{0,1\}^{|\Sigma(\ell)|} \wedge \varsigma[a] = 1 - \varsigma'[a] \wedge \upsilon[a] = a_i \Rightarrow \varsigma'[a] = 1 \wedge \upsilon[a] \neq a_i \Rightarrow \varsigma'[a] = 0\} \cup \{a^i \to \upsilon^\ell_{jeu} \Rsh \upsilon^\ell_{\varsigma 0} \mid \upsilon^\ell \in \mathrm{g}(\mathcal{L}) \wedge a \in \Sigma(\ell) \wedge \varsigma[a] = a_i \wedge \varsigma 0 = 0^{|\Sigma(\ell)|}\}$, l'ensemble des frappes de priorité 2 mettant à jour les sortes coopératives.
- $\mathcal{H}^3 = \{\upsilon^\ell_\varsigma \to \upsilon^\ell_\varsigma \Rsh \upsilon^\ell_{jeu} \mid \upsilon^\ell \in \mathrm{g}(\mathcal{L}) \wedge \varsigma \in \{0,1\}^{|\Sigma(\ell)|} \wedge \exists a, b \in \Sigma(L), a \neq b, \varsigma[a] = \varsigma[b] = 1\} \cup \{a_i \to a_i \Rsh a_j \mid a_i \xrightarrow{\epsilon} a_j \in T_a\}$, l'ensemble des frappes de priorité 3 sélectionnant la transition à jouer.

Étant donné un état $s \in L$ des Frappes de Processus, nous définissons $[\![s]\!] = x$ l'état correspondant des automates : $\forall a \in \mathcal{A}, x[a] = s[a]$.
Étant donné un état $x \in S$ des Automates Finis Communicants, nous définissons $(\!|x|\!) = s$ l'état correspondant des Frappes de Processus : $\forall a \in \mathcal{A}, x[a] = s[a]$ et $\forall \upsilon^\ell \in \mathrm{g}(\mathcal{L}), s[\upsilon^\ell] = \upsilon^\ell_\varsigma$ avec $\varsigma \in \{0,1\}^{|\Sigma(\ell)|} \wedge \forall a_i \in \upsilon, x[a] = a_i \Rightarrow \varsigma[a] = 1 \wedge x[a] \neq a_i \Rightarrow \varsigma[a] = 0$.

Intuitivement, lorsque le processus jeu d'une sorte coopérative υ^ℓ devient présent, il frappe tous les processus pouvant se synchroniser selon ℓ afin d'appliquer les transitions correspondantes. Puis, les nouveaux processus désactiveront ce processus jeu et mettront à jour toutes les autres sortes coopératives. De plus, la définition des actions de priorité 3 assure qu'au plus un processus coopératif est de type jeu.

Le théorème 4.4 prouve l'existence d'une bisimulation faible entre les Automates Finis Communicants et leur traduction en Frappes de Processus avec 3 Priorités.

Théorème 4.4 ($(\mathcal{A}, S, \mathcal{L}, T) \approx \mathbf{PH}(\mathcal{A}, S, \mathcal{L}, T)$)**.** *Étant donnés des Automates Finis Communicants* $(\mathcal{A}, S, \mathcal{L}, T)$ *et les Frappes de Processus avec* 3 *Priorités correspondantes* $\mathbf{PH}(\mathcal{A}, S, \mathcal{L}, T) = (\Sigma, L, \mathcal{H}^1, \mathcal{H}^2, \mathcal{H}^3)$*, alors :*

1. $\forall x, x' \in S$, $x \rightarrow_{AFC} x' \Longrightarrow (\!|x|\!) \rightarrow^*_{FPP} (\!|x'|\!)$, *où* $\rightarrow^*_{FP}$ *est une suite finie de transitions* $\rightarrow_{FPP}$.
2. $\forall x \in S$, *si* $\exists s' \in L$, $(\!|x|\!) \rightarrow^3_{FPP} \rightarrow^{1*}_{FPP} s' \wedge s' \not\rightarrow^1_{FPP} \Longrightarrow x \rightarrow_{AFC} [\![s']\!]$, *où* $\rightarrow^k_{FPP}$ *est une transition due à une frappe de priorité* k *et* $s' \not\rightarrow^k_{FPP}$ *signifie qu'aucune transition* $\rightarrow^k_{FPP}$ *n'est possible depuis* s'.

Démonstration. Tout d'abord, d'après la définition 4.10, pour tout $x \in S$, aucune action de priorité 1 ou 2 n'est jouable dans $(\!|x|\!)$; dans un tel cas, l'application d'une action de priorité 3 peut rendre possible des actions de priorité 1. De même, après l'application de de toutes les actions de priorité 1 possibles, des actions de priorité 2 peuvent être applicables. Au contraire, l'application d'une action de priorité 2 ne peut pas rendre possible une action de priorité 1.

(1) Si $x \rightarrow_{AFC} x'$ est due à une transition interne $a_i \xrightarrow{\epsilon} a_i$, alors $a_i \rightarrow a_i \Rsh a_j \in \mathcal{H}^3$ est jouable dans $(\!|x|\!)$ et nous avons $(\!|x|\!) \rightarrow^3_{FPP} (\!|x'|\!)$. Si $x \rightarrow_{AFC} x'$ est due à une synchronisation libellée ℓ entre $k \geq 2$ automates $a^1, \ldots, a^k$, alors $\exists \upsilon^\ell \in \mathbf{g}(\mathcal{L})$ tel que $(\!|x|\!)[\upsilon^\ell] = \upsilon^\ell_\varsigma$ avec $\forall i \in [1; k], \varsigma[k] = 1$. Ainsi l'action $\upsilon^\ell_\varsigma \rightarrow \upsilon^\ell_\varsigma \Rsh \upsilon^\ell_{jeu} \in \mathcal{H}^3$ est jouable dans $[\![x]\!]$ et son application active les actions de $\mathcal{H}^1$ frappant les processus $x[a^i]$ pour les faire bondir en $x'[a^i]$, $\forall i \in [1; k]$. Après le jeu de toutes ces actions, aucune autre action de priorité 1 n'est jouable, et les actions de $\mathcal{H}^2$ mettant à jour l'état des sortes coopératives sont jouées, résultant en $(\!|x'|\!)$. D'où $(\!|x|\!) \rightarrow^*_{FPP} (\!|x'|\!)$.

(2) Découle de (1) en remarquant que $\forall s, s', s \rightarrow^2_{FPP} s'$, $[\![s]\!] = [\![s']\!]$. □

4.4 Discussion

Dans ce chapitre, nous avons comparé formellement les Frappes de Processus avec différents formalismes classiques de concurrence.

Il apparaît clairement que les Frappes de Processus peuvent être traduites dans ces formalismes standards de façon simple et directe, comme nous l'avons montré pour les Automates Finis Communicants, le π-Calcul et les Réseaux de Petri. En revanche, la traduction de formalismes concurrents en Frappes de Processus préservant une relation de bisimulation (faible) s'avère plus complexe à réaliser avec un coût raisonnable de construction.

Nous avons montré la traduction des Réseaux Discrets et, plus généralement, des Automates Finis Communicants en Frappes de Processus augmentées par la notion de classes de priorité pour les actions : une action de priorité n n'est jouable que si aucune action de priorité $< n$ n'est jouable. Alors, nos traductions préservent une relation de bisimulation faible et sont de coût raisonnable (exponentiel selon l'arité des interactions, polynomial selon leur nombre).

Il est important de remarquer que les relations de bisimulation obtenues reposent sur l'ajout d'une notion de priorité aux Frappes de Processus tout en ignorant une telle capacité sur le formalisme original. Toutefois, la traduction de ces mêmes formalismes dotés de classes de priorité peut être dérivée des traductions proposées : le choix des actions à jouer étant toujours dans la classe de priorité la plus faible, il suffit de scinder cette dernière classe au regard des priorités des actions du modèle original. Ainsi, nous obtenons notamment une équivalence d'expressivité entre les Automates Finis Communicants avec k Priorités et les Frappes de Processus avec $k + 2$ Priorités.

La traduction du π-Calcul et des Réseaux de Petri en Frappes de Processus n'est pas traitée par ce chapitre, et pourrait constituer un travail futur intéressant. Concernant le π-Calcul, cette traduction requerrait l'extraction d'un nombre fixe de composantes agissant en parallèle et possédant un nombre fini d'états. Il existe une traduction d'un π-Calcul restreint vers le langage PRISM (Hinton, Kwiatkowska, Norman & Parker, 2006) proposée par Norman, Palamidessi, Parker & Wu (2009) et abordée dans le chapitre 5. PRISM est un langage basé sur les Modules Réactifs (Alur & Henzinger, 1999) et possède de fortes similitudes avec la définition des Automates Finis Communicants proposée dans ce chapitre, donnant ainsi une piste sérieuse pour la traduction du π-Calcul en Frappes de Processus. La traduction des Réseaux de Petri en Frappes de Processus, elle, semble plus simple à aborder, la similitude avec les Automates Finis Communicants étant assez proche sous réserve d'imposer l'unicité des places dans un marquage.

Dans le reste de ce manuscrit, nous ne considérerons que les Frappes de Processus classiques, sans classes de priorités. Ainsi, les traductions présentées dans la section précédente résultent en une approximation supérieure du comportement du modèle original. Nous notons cependant que ces priorités pourraient être approximées à l'aide de paramètres stochastiques ou temporels tels que présentés dans la partie II. Enfin, la gestion de cette notion de priorité au sein des analyses statiques développées dans le chapitre 9 constitue une ouverture pour les travaux de cette thèse, discutée dans le chapitre 12.

Deuxième partie

Introduction de Paramètres Temporels et Stochastiques

Cette partie développe et démontre des techniques permettant l'introduction conjointe de paramètres temporels et stochastiques dans la modélisation des systèmes complexes en général. Ces paramètres se traduisent par l'apparition de durées lors des transitions effectuées lors de l'exécution d'un modèle, impactant son comportement discret (des transitions deviennent impossibles) et quantitatif (durées, probabilités).
Dans le chapitre 5, nous introduisons le *facteur d'absorption de stochasticité* afin de contrôler la variance temporelle des durées aléatoires, et permettant ainsi la spécification d'intervalles de temps bornant les durées des transitions. Cette étude se place dans le cadre général du π-calcul stochastique et de la vérification formelle de propriétés quantitatives avec le logiciel PRISM.
Dans le chapitre 6, nous traitons de la simulation des modèles stochastiques en proposant une machine abstraite générique facilitant la définition et la démonstration de la simulation markovienne et non-markovienne des calculs de processus.
Enfin, nous appliquons dans le chapitre 7 les résultats obtenus dans cette partie à l'introduction de paramètres temporels et stochastiques dans les Frappes de Processus.

Chapitre **5**

Le Raffinement du Temps dans le Cadre Général du π-Calcul Stochastique

Le π-calcul stochastique est un formalisme utilisé pour modéliser des systèmes dynamiques complexes, où l'aléatoire et le délai des transitions sont des caractéristiques importantes, comme, par exemple, les réactions biochimiques. Généralement, les durées des transitions en π-calcul stochastique suivent une loi de distribution exponentielle. La dynamique sous-jacente de tels modèles peut alors être exprimée par des chaînes de Markov à temps continu, pour lesquelles il existe des méthodes efficaces de simulation et de vérification formelle. Cependant, la loi exponentielle possède une très grande variance, rendant difficile la modélisation de systèmes avec des contraintes temporelles précises. Dans ce chapitre, nous montrons une technique pour raffiner le temps en π-calcul stochastique. Cette méthode introduit un facteur d'*absorption de stochasticité* en remplaçant la distribution exponentielle par la distribution d'Erlang, qui suit la loi d'une somme de variables aléatoires exponentielles. Nous présentons une construction de cette absorption en π-calcul stochastique distribué exponentiellement. Des outils pour manipuler l'absorption de stochasticité et son lien avec les intervalles de temps sont également discutés. Enfin, nous aborderons la vérification formelle de tels systèmes en montrant une traduction vers PRISM, un vérificateur de modèles probabilistes.

5.1 Préliminaires

En introduisant des aspects temporels et stochastiques dans les modèles, leurs concepteurs aspirent à reproduire avec plus de précision le comportement des systèmes réels. L'incorporation d'une telle complexité dans les modèles est souvent le résultat d'un compromis entre une spécification temporelle précise et des méthodes efficaces d'analyse.

Le π-calcul (Milner, 1989) est une algèbre de processus concurrents adaptée pour la modélisation d'entités, définies indépendamment, communicant entre elles par des canaux mobiles. En particulier, le π-calcul est largement utilisé pour l'analyse de protocoles de communication (Abadi & Gordon, 1999). Parmi les différentes extensions de ce formalisme, le π-calcul stochastique (Priami, 1995) introduit des dimensions temporelles et stochastiques dans les modèles. Le π-calcul stochastique est notamment utilisé pour modéliser les systèmes biologiques (Priami, Regev, Shapiro & Silverman, 2001; Kuttler & Niehren, 2006; Blossey, Cardelli & Phillips, 2008; Cardelli, Caron,

Gardner, Kahramanogullari & Phillips, 2009). Ces dimensions de temps et d'aléatoire sont cruciales pour l'analyse et la prédiction du comportement de systèmes complexes, tels que les réactions biochimiques.

En π-calcul stochastique, le temps et l'aléatoire sont incorporés en imposant une durée aléatoire pour effectuer une transition. Habituellement, cette durée suit une distribution exponentielle. L'unique paramètre de cette distribution est généralement appelé *taux* et, de manière informelle, définit le nombre de fois que la transition peut être utilisée en l'espace d'une unité de temps. La sémantique sous-jacente du π-calcul stochastique distribué exponentiellement peut s'exprimer avec des Chaînes de Markov à Temps Continu (CMTC). En exploitant l'absence de mémoire de la loi exponentielle, des algorithmes de simulation très efficaces ont pû être établis — par exemple, *l'algorithme de Gillespie* (Gillespie, 1977), et les algorithmes implémentés dans *BioSpi* (Priami et coll., 2001) et *SPiM* (Phillips & Cardelli, 2007). La vérification formelle du π-calcul stochastique est également possible en utilisant, par exemple, sa traduction vers PRISM, un vérificateur symbolique de modèles probabilistes (Hinton et coll., 2006; Norman et coll., 2009).

Bien que les taux des transitions apportent une dimension temporelle aux modèles, la grande variance imposée par la distribution exponentielle prévient une modélisation précise de contraintes temporelles. Par exemple, on peut se demander comment modéliser en π-calcul stochastique une transition prenant place seulement dans un intervalle de temps donné, à la façon des Automates Temporisés (Alur & Dill, 1992) ou des Réseaux de Petri temporisés (Merlin, 1974). Ce type de spécification temporelle est très courant, intuitif, et nécessaire pour modéliser des systèmes où le temps détermine leurs dynamiques.

Dans ce chapitre, nous présentons une technique pour raffiner la dimension temporelle dans les modèles en π-calcul stochastique, et nous proposons une méthode pour analyser de tels modèles avec PRISM. Ce raffinement permet la modélisation conjointe d'aspects temporels et stochastiques d'un système d'une manière plus souple que dans le π-calcul stochastique classique. Une telle modélisation repose sur l'attribution aux transitions, en supplément d'un taux d'utilisation, un *facteur d'absorption de stochasticité*. Plus ce facteur d'absorption est élevé, plus la variance temporelle de la durée est réduite autour d'une moyenne imposée par le taux d'utilisation. L'absorption de stochasticité des transitions est obtenue en remplaçant la distribution exponentielle par la distribution d'Erlang, qui est la distribution de la somme de variables aléatoires exponentielles.

Nous apportons ici trois contributions originales :

- *La traduction du π-calcul stochastique avec des distributions d'Erlang vers le π-calcul stochastique avec des distributions exponentielles.* Cette traduction rend possible la simulation et l'analyse des modèles en π-calcul stochastique distribué avec Erlang par les très nombreux outils se reposant sur la distribution exponentielle. La construction est purement syntaxique, et n'utilise pas une construction explicite des CMTC.
- *Des estimateurs pour les paramètres de la distribution d'Erlang correspondant à un intervalle de temps.* Ainsi, similairement aux automates et réseaux de Petri temporisés, une transition peut être paramétrée en spécifiant un intervalle de confiance pour sa durée, que nous appelons *intervalle de tir*. La transition sera effective dans cet intervalle de temps avec un degré de confiance donné.
- *La vérification formelle du π-calcul stochastique distribué par Erlang en utilisant* PRISM, en adaptant une traduction existante du π-calcul stochastique markovien vers PRISM établie par Norman et coll. (Norman et coll., 2009) pour prendre en compte le facteur d'absorption de stochasticité. De plus, la construction en PRISM du facteur d'absorption de stochasticité que nous proposons peut être appliquée à des modèles encore plus génériques que ceux résultant de la traduction depuis le π-calcul stochastique.

La complexité de la traduction du π-calcul stochastique avec les distributions d'Erlang vers le π-calcul stochastique avec les distributions exponentielles est linéaire avec la taille du modèle ; de même concernant l'incorporation du facteur d'absorption de stochasticité dans la traduction en PRISM. Alors que la simulation ne souffre pas de l'ajout des absorptions, nous observons que la vérification formelle de tels modèles est confrontée à une explosion combinatoire de l'espace des états à explorer lorsque de forts facteurs d'absorption de stochasticité sont employés.

L'applicabilité de nos techniques est illustrée par la modélisation d'un phénomène biologique de segmentation (François, Hakim & Siggia, 2007). Cette étude de cas biologique repose sur une spécification temporelle assez précise, difficilement modélisable avec un π-calcul purement markovien. Nous proposons ici un modèle en π-calcul stochastique distribué avec Erlang, et nous étudions l'influence du facteur d'absorption de stochasticité sur la reproduction de comportements particuliers du modèle.

Ce chapitre est structuré ainsi : La section 5.2 présente une variante du π-calcul stochastique sur laquelle les résultats de ce chapitre s'appliquent. La section 5.3 introduit le facteur d'absorption de stochasticité à travers la distribution d'Erlang. Une construction du π-calcul stochastique avec les distributions d'Erlang en distributions exponentielles est fournie. La section 5.4 établit le parallèle entre l'intervalle de tir d'une transition et le couple du taux d'utilisation et du facteur d'absorption de stochasticité. La section 5.5 adapte la traduction du π-calcul stochastique standard vers PRISM pour ajouter la gestion du facteur d'absorption de stochasticité. L'applicabilité de cette approche globale est discutée et illustrée dans la section 5.6. Enfin, les sections 5.7 et 5.8 discutent des travaux relatifs et des contributions apportées par ce chapitre.

Ces résultats sont publiés dans (Paulevé, Magnin & Roux, 2010).

5.2 Le π-Calcul Stochastique

Le π-calcul est une algèbre de processus concurrents, où deux processus communiquent via un canal commun (Milner, 1989; Milner, 1999). Pour établir une communication, un processus émetteur écrit sur un canal et un processus récepteur lit sur ce même canal. Le processus émetteur peut écrire des *valeurs* sur le canal afin de transmettre des données au récepteur, une valeur pouvant être également un canal. Ainsi, le π-calcul permet de modéliser une mobilité des canaux de communication entre des processus concurrents.

L'extension stochastique du π-calcul attache à chaque *action* (entrée/sortie sur un canal, ou transition interne) une distribution de probabilité qui détermine la durée de l'action (Priami, 1995; Priami, 1996). La distribution exponentielle est la plus usitée. Le seul paramètre de cette distribution s'appelle le *taux d'utilisation* de l'action, et, informellement, correspond au nombre de fois que l'action peut être tirée en une unité de temps.

La syntaxe du π-calcul stochastique utilisée dans cet article est présentée dans la définition 5.1. Elle est similaire à la définition utilisée dans (Phillips et coll., 2006).

Définition 5.1 (π-calcul stochastique)**.** En désignant les termes par C, C_i, P, Q, les définitions des compétitions par A, les identifiants des transitions internes par t, les actions par π, les canaux par a, y, et les valeurs par $m, z_1, \ldots, z_n$:

$$A(z_1, \ldots, z_n) ::= \sum_{i \in I} C_i \qquad \text{avec } \mathrm{fn(C_i)} \subseteq \{\mathrm{z_1}, \ldots, \mathrm{z_n}\}$$
$$C, C_i ::= \nu a\ C \mid [cond]\ C \mid \pi.P$$
$$\pi ::= \tau_t \mid am \mid \bar{a}m$$
$$P, Q ::= P|Q \mid A(z_1, \ldots, z_n) \mid \nu x\ P \mid [cond]\ P \mid \mathbf{0}$$

où I est un ensemble fini d'indices, et $cond$ est une expression booléenne entre des valeurs ; fn(C) est l'ensemble des canaux qui sont libres dans C.
$A()$ est abrégé par A et $z_1, \ldots, z_n$ est abrégé par $\tilde{z}$; $C_1 + C_2$ signifie $\sum_{i \in \{1,2\}} C_i$; am (resp. $\bar{a}m$) est abrégé par a (resp. $\bar{a}$) si m n'est pas utilisé.

Une action π d'un processus est soit une transition interne (τ_t), une entrée (ou lecture) de m sur un canal a (am), ou une sortie (ou écriture) de m sur un canal a ($\bar{a}m$) ; m est une valeur : soit un canal y, une liste (de taille variable) de valeurs, ou un n-uplet de valeurs $\tilde{m}$. Comme dans (Phillips et coll., 2006), m peut également désigner un nouveau canal νy lors de l'écriture ; nous parlerons alors d'écriture liée. L'action π effectuée, le processus peut évoluer vers P, ce qui s'écrit $\pi.P$. Le terme $\mathbf{0}$ dénote le processus nul, $P|Q$ la composition parallèle des processus P et Q, $[cond]P$ l'activation du processus P si la condition $cond$ est satisfaite (généralement, $cond$ est une conjonction de tests sur des valeurs), et νaP dénote la création d'un canal a restreint au processus P. $\sum_{i \in I} C_i$ est une compétition entre les processus C_i : seule la première action tirée est considérée. Une compétition est associée à une définition de la forme $A(\tilde{z}) = \sum_{i \in I} C_i$, où $\tilde{z}$ représente les paramètres de la définition A. Nous supposons que I est fini, et que sa valeur peut dépendre des valeurs dans $\tilde{z}$. Si I est vide, la compétition est équivalente au processus nul.

Un canal est *lié* à un processus P si il existe une restriction νa dans la définition de P. Dans le cas contraire, le canal a est dit libre (ou non lié). L'ensemble des canaux liés à un processus P est noté bn(P) et l'ensemble des canaux libres fn(P). $\mathrm{n(P)} = \mathrm{fn(P)} \cup \mathrm{bn(P)}$ est l'ensemble des canaux présents dans le terme P.

La première action exécutée par un processus C en compétition est dénotée par $\Pi(C)$, définie ci-dessous. Dans la suite de ce chapitre, nous supposons que si la première action exécutée par C est une entrée ou une sortie sur un canal, ce canal est libre dans C.

$$\Pi(\tau_t.P) = \tau_t \quad \Pi(\nu x\ C) = \Pi(C) \text{ si } \Pi(C) \notin \{x, \bar{x}\}$$
$$\Pi(ay.P) = a \quad \Pi([cond]\ C) = \Pi(C)$$
$$\Pi(\bar{a}y.P) = \bar{a}$$

La figure 5.1 précise la sémantique opérationnelle du π-calcul stochastique, similairement à Phillips et coll. (2006). Les transitions sont libellées par l'action à exécuter : τ_t dénote une transition interne identifiée par t, $\dot{a}$ une communication sur un canal a, c.-à-d. une synchronisation entre une entrée et une sortie sur un même canal a. Les canaux libres et liés des actions π sont définis dans le tableau 5.1.

Nous dénotons par $S\pi_e$ le π-calcul stochastique dont la durée de chaque action selon a (transition interne τ_a ou entrée/sortie sur un canal a) est une variable aléatoire suivant la loi exponentielle avec un taux d'utilisation $r_a \in \mathbb{R}+^*$. La probabilité qu'une action suivant un taux r soit tirée dans

π	description	$\mathsf{fn}(\pi)$	$\mathsf{bn}(\pi)$
τ_t	transition interne	$\emptyset$	$\emptyset$
am	lecture de la valeur m	$\{a, m\}$	$\emptyset$
$\bar{a}m$	écriture de la valeur m	$\{a, m\}$	$\emptyset$
$\bar{a}\nu y$	écriture liée du canal y	$\{a\}$	$\{y\}$
$\dot{a}$	communication	$\{a\}$	$\emptyset$

Tableau 5.1 – Canaux libres et liés d'une action π.

$$\mathsf{TAU}\ \frac{}{\tau_t.P \xrightarrow{\tau_t} P} \qquad \mathsf{PRE_{IN}}\ \frac{}{ay.P \xrightarrow{ay} P} \qquad \mathsf{PRE_{OUT}}\ \frac{}{\bar{a}y.P \xrightarrow{\bar{a}y} P} \qquad \mathsf{SUM}\ \frac{P_j \xrightarrow{\pi} P'_j}{(\sum_{i\in I} P_i) \xrightarrow{\pi} P'_j}\, j \in I$$

$$\mathsf{PRE_{BIND}}\ \frac{P \xrightarrow{\bar{a}y} P'}{\nu y P \xrightarrow{\bar{a}\nu y} P'}\, a \neq y \qquad \mathsf{COM}\ \frac{P \xrightarrow{ay} P' \quad Q \xrightarrow{\bar{a}z} Q'}{P|Q \xrightarrow{\dot{a}} P'\{z/y\}|Q'} \qquad \mathsf{BCOM}\ \frac{P \xrightarrow{ay} P' \quad Q \xrightarrow{\bar{a}\nu z} Q'}{P|Q \xrightarrow{\dot{a}} \nu z(P'\{z/y\}|Q')}\, z \notin n(P)$$

$$\mathsf{RES}\ \frac{P \xrightarrow{\pi} P'}{\nu a P \xrightarrow{\pi} \nu a P'}\, a \notin n(\pi) \qquad \mathsf{MATCH}\ \frac{P \xrightarrow{\pi} P'}{[cond]P \xrightarrow{\pi} P'}\, cond \qquad \mathsf{PAR}\ \frac{P \xrightarrow{\pi} P'}{P|Q \xrightarrow{\pi} P'|Q}\, bn(\pi) \cap \mathsf{fn(Q)} = \emptyset$$

Figure 5.1 – Sémantique opérationnelle du π-calcul stochastique (définition 5.1).

un délai de t unités de temps est donnée par $1 - e^{-r.t}$. La durée moyenne d'une telle action est de r^{-1} unités de temps avec une variance de r^{-2}. Étant donnés x actions ayant respectivement des taux d'utilisation $r_1, \ldots, r_x$, la probabilité que la y^{e} action soit tirée est $\frac{r_y}{r_1+\cdots+r_x}$. Nous considérons également les actions avec un taux infini $r_a = \infty$, c.-à-d. les actions instantanées. Une telle action est toujours tirée en premier. Si deux actions instantanées sont tirables, l'action à tirer est choisie de façon non-déterministe.

5.3 Absorption de Stochasticité

La loi de distribution exponentielle impose un lien fort entre la durée moyenne et la variance temporelle d'une transition. Par exemple, plus la durée moyenne est élevée (c.-à.d, plus le taux d'utilisation est faible), plus la variance est forte. Le *facteur d'absorption de stochasticité* a pour but d'apporter plus de souplesse entre ces dimensions temporelles et stochastiques.

Au lieu d'affecter à la durée d'une action une variable aléatoire exponentielle suivant un taux d'utilisation r, nous proposons une durée suivant la somme de sa variables aléatoires exponentielles avec un taux $r.sa$. Ceci n'affecte pas la durée moyenne d'une action, mais réduit sa variance d'un facteur sa ; sa étant donc le facteur d'absorption de stochasticité. La distribution de probabilité obtenue est connue comme étant la *distribution d'Erlang*. Ainsi, le raffinement du temps dans le π-calcul stochastique se traduit par la spécification, pour la durée de chaque action, d'une distribution d'Erlang déterminée par un taux d'utilisation et un facteur d'absorption de stochasticité. Nous dénotons par $S\pi_{Er}$ le π-calcul stochastique distribué par Erlang.

Cette section présente tout d'abord la distribution d'Erlang et des fonctions pour calculer quelques probabilités courantes. La traduction de $S\pi_{Er}$ vers $S\pi_e$ est ensuite présentée, montrant que les processus $S\pi_{Er}$ peuvent être simulés par des algorithmes classiques, basés sur la loi exponentielle. Enfin, un exemple simple de $S\pi_{Er}$ est étudié.

5.3.1 La Distribution d'Erlang

La distribution d'Erlang est usuellement définie par deux paramètres : la forme $k \in \mathbb{N}^*$ et le taux $\lambda \in \mathbb{R}_+^*$. La distribution d'Erlang est la distribution de la somme de k variables aléatoires indépendantes distribuées exponentiellement suivant un taux d'utilisation λ. La distribution d'Erlang est un cas particulier de la *distribution gamma*, où le paramètre de forme peut être un nombre réel positif quelconque.

Pour une raison de cohérence, nous définissons la distribution d'Erlang comme étant la distribution de la somme de sa variables aléatoires exponentielles suivant un taux d'utilisation $r.sa$, où sa est le taux d'absorption de stochasticité, et r le taux d'utilisation de la variable aléatoire exponentielle non-absorbée (c.-à-d, $sa = 1$). Cette définition est bien entendu équivalente au regard des relations $k = sa$ et $\lambda = r.sa$.

La densité de probabilité et la fonction de répartition d'une distribution d'Erlang suivant un taux r et une absorption de stochasticité sa sont définies par les équations (5.1) et (5.2), respectivement. Elles correspondent à la distribution d'Erlang usuelle avec une forme sa et un taux $r.sa$ (Evans, Hastings & Peacock, 2000). Des exemples de densités de probabilité et de fonctions de répartition sont dessinées dans la figure 5.2.

$$f_{r,sa}(t) = \frac{(r.sa)^{sa} t^{sa-1} e^{-r.sa.t}}{(sa-1)!} \tag{5.1}$$

$$F_{r,sa}(t) = 1 - e^{-r.sa.t} \sum_{n=0}^{sa-1} \frac{(r.sa.t)^n}{n!} \quad . \tag{5.2}$$

Soit une action dont la durée suit une distribution d'Erlang avec un taux r et une absorption de stochasticité sa. La durée moyenne de cette action est r^{-1} avec une variance de $r^{-2}sa^{-1}$. $F_{r,sa}(t)$ donne la probabilité de tirer l'action avant un temps t. Étant donné x actions aux taux d'utilisation respectifs $r_1, \ldots, r_x$ et facteurs d'absorption de stochasticité $sa_1, \ldots, sa_x$, la probabilité que la y^{e} action soit tirée est donnée par l'équation (5.3) (Priami, 1996).

$$\int_0^\infty f_{r_y,sa_y}(t) \prod_{w \neq y} (1 - F_{r_w,sa_w}(t)) dt \quad . \tag{5.3}$$

5.3.2 Construction en π-Calcul Stochastique

Dans cette section, une construction des processus $S\pi_{Er}$ en $S\pi_e$ est proposée. Cette construction est purement syntaxique : nous n'utilisons aucune construction explicite de CMTC ; et sa complexité est linéaire suivant la taille de l'expression $S\pi_{Er}$. Cette correspondance permet également l'analyse des processus $S\pi_{Er}$ par les outils classiques (simulateurs, vérificateurs formels, etc.) qui supposent une distribution exponentielle des durées des transitions.

Sémantique de $S\pi_{Er}$. Dans le cadre de la présentation de la sémantique de $S\pi_{Er}$, nous considérons que toute action possible est de la forme π^w où w est l'indice globalement unique de l'action π (voir la section 6.4 page 93 pour le traitement détaillé des indices des actions en π-calcul stochastique). Alors, au cours d'une exécution, au plus une action π^w est executée, pour un w fixé.

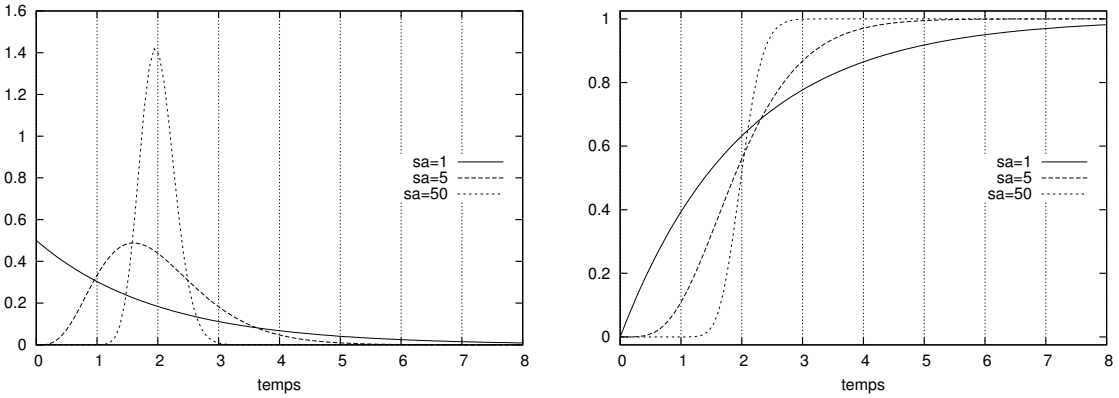


Figure 5.2 – Densité de probabilité (gauche) et fonction de répartition (droite) de la distribution d'Erlang pour différents facteurs d'absorption de stochasticité (sa) quand le taux (r) est $\frac{1}{2}$.

Nous notons activé(P) l'ensemble des actions possibles dans le processus P. En particulier :

$$\frac{P \xrightarrow{\pi^w} P'}{\pi^w \in \text{activé}(P)} \qquad \frac{\sum_{i\in I} C_i \xrightarrow{\pi^w} P'}{\text{activé}(\sum_{i\in I} C_i) \cap \text{activé}(P') = \emptyset}$$

Ainsi, nous considérons ici que tout processus P exécutant une action au sein d'une compétition désactive toutes les actions de cette compétition, même si P est redémarré (p. ex., si $P \triangleq \tau_1.P + \tau_2.P'$ et que la transition interne τ_1 est exécutée, l'action τ_2 sera totalement réinitialisée). Nous notons toutefois qu'un tel comportement peut être modifié facilement dans la sémantique et la construction proposée dans cette section.

Nous définissons la sémantique de $S\pi_{Er}$ de manière similaire à la sémantique du π-calcul stochastique avec des distributions générales proposée par Priami (1996) : soit $\xi_0 = P_0 \xrightarrow{\pi_0^{w_0}} P_1 \xrightarrow{\pi_1^{w_1}} \ldots \xrightarrow{\pi_n^{w_n}} P_{n+1}$ une exécution possible (voir figure 5.1). L'*exécution stochastique* correspondante est donnée par $\xi_{n+1} = P_0 \xrightarrow{\pi_0^{w_0},\tilde{F}_0} P_1 \xrightarrow{\pi_1^{w_1},\tilde{F}_1} \ldots \xrightarrow{\pi_n^{w_n},\tilde{F}_n} P_{n+1}$, où $\tilde{F}_i = \mathbb{P}\left[\delta_i \leq t + \sum_{h=j+1}^{i-1} \delta_h \;\middle|\; \delta_i > \sum_{h=j+1}^{i-1} P_h\right]$ est la distribution de la variable aléatoire δ_i décrivant le délai associé à la transition $P_i \xrightarrow{\pi_i^{w_i}} P_{i+1}$ et avec $j = \max\{k \in [0; i-1] \mid \pi_i^{w_i} \notin \text{activé}(P_k)\}$.

Construction du facteur d'absorption de stochasticité. Au préalable, à tout couple de paramètres (r_a, sa_a) des actions des processus $S\pi_{Er}$, un paramètre $r'_a = r_a.sa_a$ est attaché aux actions des processus $S\pi_e$ résultants. Nous supposons par la suite que $sa_a > 1$, c.-à-d que l'action ne suit pas déjà une distribution exponentielle. Notre objectif est alors d'exécuter sa_a fois l'action exponentielle a (une transition interne τ_a ou une communication sur un canal a) avant d'appliquer la transition (c.-à-d, activer le processus suivant). La construction est effectuée via un opérateur $(\!|\cdot|\!)_e$, où $\cdot$ est le terme en $S\pi_{Er}$. Si P est un terme en $S\pi_{Er}$, $(\!|P|\!)_e$ est un terme en $S\pi_e$ possédant un comportement similaire à P en utilisant seulement des distributions exponentielles (ceci sera formellement détaillé dans cette section). La traduction complète entre les termes en $S\pi_{Er}$ vers des termes en $S\pi_e$ est formalisée dans la définition 5.2. Nous soulignons le fait que cette construction peut être appliquée à tout processus $S\pi_{Er}$.

Le principe général de la construction est l'association d'un compteur pour chaque action a

présente dans la définition des processus. Lorsque ce compteur atteint $sa_a - 1$, la prochaine exécution de l'action a sera appliquée normalement (la transition a lieu). Dans le cas contraire, l'exécution de l'action a relancera le processus, en incrémentant d'une unité la valeur du compteur. Alors que ce principe peut être appliqué directement aux actions internes (équation (5.8)), son application aux entrées/sorties sur un canal demande un peu plus d'astuce. Considérons le processus $S\pi_{Er}$ suivant, A_1, présent dans l'expression de A :

$$\begin{aligned} &A_1(a) \triangleq \bar{a}.\mathbf{0} \qquad A_2(a) \triangleq a.\mathbf{0} \\ &A(a) \triangleq A_1(a) \mid A_2(a) \mid A_2(a) \ . \end{aligned} \tag{5.4}$$

En comptant le nombre d'exécutions de l'écriture $\bar{a}$, une différenciation doit être faite entre les instances des processus lecteurs A_2.

La procédure employée dans la définition 5.2 repose alors sur l'identification des processus lors d'une communication : lors de l'instanciation d'un processus, un canal unique (noté id) lui est ajouté dans ses paramètres (équation (5.6)). La communication sur le canal a entre deux processus A et B, où A est identifié id peut se résumer par les étapes suivantes (équations (5.9) et (5.10)) :

1. A écrit sur le canal a le triplet $\langle \nu a', \nu a'', id \rangle$: deux nouveau canaux privés et son identifiant.

2. B lit sur le canal a le triplet $\langle a', a'', pid \rangle$: si pid est déjà connu (une communication est déjà en cours), les canaux a' et a'' sont ignorés ; si pid est inconnu de B, alors les lectures sur a' et a'' sont ajoutées aux actions à effectuer en compétition par B.

3. A écrit $sa_a - 2$ fois sur le canal a', B lit ; les deux processus redémarrent.

4. A écrit finalement sur a'' et lorsque B lit sur a'', la transition est effectuée.

Ainsi, une communication n'est pas littéralement répétée sa_a fois sur le canal a, mais sur les canaux a, a' et a''.

Afin d'enregistrer les canaux privés a', a'' et les identifiants connus, la définition d'un processus mettant en compétition m actions se voit ajouter, en supplément de son canal identifiant, une liste de paramètres notée $\tilde{\rho}$ (équation (5.5)) représentant quatre types de liste ; les listes $c_1, \ldots, c_m$ représentant les compteurs associés aux actions en compétition ; les listes $a'_1, \ldots, a'_m$ enregistrant les canaux a' reçus ou émis ; les listes $a''_1, \ldots, a''_m$ enregistrant les canaux a'' reçus ou émis ; les listes $k_1, \ldots, k_m$ enregistrant les identifiants pid reçus. Dans le cas où l'action indicée i est une simple action interne, c_i est alors un entier positif, et les listes associées a'_i, a''_i et k_i ne sont pas utilisées. Dans le cas où l'action indicée i est une écriture sur le canal a, plusieurs communications peuvent être démarrées selon le nombre de processus parallèles lisant sur a : à chaque initiation de communication les canaux privés envoyés sont ajoutés aux listes a'_i et a''_i afin de les mémoriser lors de la ré-itération du processus. Il en est de même pour le processus récepteur, qui ajoute les canaux a' et a'' reçus aux listes a'_i et a''_i seulement si le canal pid reçu n'est pas présent dans la liste k_i.

En ignorant les actions d'écriture et de lecture sur les canaux identifiants, la transformation des processus $S\pi_{Er}$ définis dans l'équation (5.4) avec l'opérateur $(\!|\cdot|\!)_e$ résulte en les processus $S\pi_e$

Définition 5.2 ($(\!|\cdot|\!)_e$)**.** Nous dénotons par $(\!|\cdot|\!)_e$ la transformation d'un processus $S\pi_{Er}$ en processus $S\pi_e$. Cette transformation est définie récursivement par les règles exposées ci-dessous.
Étant une définition de processus $A(\tilde{z}) \triangleq \sum_{i\in I} C_i$ nous définissons $\tilde{\rho}$ la suite de paramètres $c_1, \ldots, c_m, a'_1, \ldots, a'_m, a''_1, \ldots, a''_m, k_i \ldots, k_m$, où $m = |I|$. Le paramètre c_i dénote le ou les compteur(s) associé(s) au i^e choix C_i : si C_i effectue une action interne, alors c_i est un entier positif ; si C_i effectue une lecture ou écriture sur un canal, alors c_i est une liste d'entiers positifs. Les paramètres a'_i, a''_i et k_i dénotent des listes de canaux privés. Le n^e élément de le la liste a'_i est dénoté par $a'_{i/n}$. $\tilde{\rho}\{c_i{+}{=}1\}$ représente l'incrémentation du compteur c_i ; $\tilde{\rho}\{\langle 1, a', a'', pid\rangle :: \tilde{\rho}_i\}$ représente l'ajout respectif de 1, a', a'' et pid dans les listes c_i, a'_i, a''_i et k_i. Enfin, $\overline{a}\langle pid, a', a''\rangle$ (resp. $a\langle pid, a', a''\rangle$)) dénote l'écriture (resp. la lecture) sur le canal a du triplet de canaux pid, a', a''.
Nous posons les transformations n'impliquant pas d'action ou la définition d'un processus :

$$(\!|\mathbf{0}|\!)_e = \mathbf{0} \qquad (\!|\nu x P|\!)_e = \nu x (\!|P|\!)_e$$
$$(\!|P \mid Q|\!)_e = (\!|P|\!)_e \mid (\!|Q|\!)_e \qquad (\!|[cond]P|\!)_e = [cond](\!|P|\!)_e$$

De même pour les actions dont le facteur d'absorption est 1 :

$$(\!|\tau_t.P|\!)_e = \tau_t.(\!|P|\!)_e \qquad (\!|\bar{a}y.P|\!)_e = \bar{a}y.(\!|P|\!)_e \qquad (\!|ay.P|\!)_e = ay.(\!|P|\!)_e$$

Dans la suite, le facteur d'absorption sa_a d'une action a est considéré > 1 ; nous posons le nouveau taux de l'action $r'_a = r_a.sa_a$; nous supposons $A(\tilde{z}) = \sum_{i\in I} C_i$ et $i \in I$.

$$(\!|A(\tilde{z}) \triangleq \sum_{i\in I} C_i|\!)_e = \tag{5.5}$$
$$A(\tilde{z}, id, \tilde{\rho}) \triangleq \sum_{i\in I} (\!|C_i|\!)_e + id\langle pid, a', s\rangle.\begin{pmatrix} [s = \text{stop}] & A(\tilde{z}, id, \tilde{\rho} \setminus \tilde{\rho}_{j/n}) \\ \mid [s = \text{ajout}] & A(\tilde{z}, id, \tilde{\rho}\{k_{j/n} = pid\}) \end{pmatrix}$$

où $\tilde{\rho} \cap fn(A(\tilde{z})) = \emptyset$, et $a'_{j,n} = a'$. $\tilde{\rho} \setminus \tilde{\rho}_{j/n}$ supprime le n^e élément de chaque liste indicée j.

$$(\!|A(\tilde{z})|\!)_e = A(\tilde{z}, \nu id, \tilde{\rho}^0) \tag{5.6}$$

où $r_{id} = \infty$ et $\tilde{\rho}^0 = c^0_1, \ldots c^0_{|I|}, \underbrace{\emptyset, \ldots, \emptyset}_{|I|}, \underbrace{\emptyset, \ldots, \emptyset}_{|I|}, \underbrace{\emptyset, \ldots, \emptyset}_{|I|}$, avec $c^0_i = \begin{cases} 0 & \text{si } \Pi(C_i) = \tau_a \\ \emptyset & \text{sinon.} \end{cases}$

$$(\!|P|\!)^{stop}_e = \left((\!|P|\!)_e \mid \prod_{i\in I} \prod_{n=1}^{|k_i|} \overline{k_{i_n}}\langle id, a'_{i_n}, \text{stop}\rangle\right) \tag{5.7}$$

$$(\!|\tau_a.P|\!)_e = \tau_a.\begin{pmatrix} [c_i < sa_a - 1] & A(\tilde{z}, \tilde{\rho}\{c_i{+}{=}1\}) \\ \mid [c_i = sa_a - 1] & (\!|P|\!)^{stop}_e \end{pmatrix} \tag{5.8}$$

$$(\!|\bar{a}y.P|\!)_e = \bar{a}\langle \nu a', \nu a'', id\rangle.A(\tilde{z}, id, \tilde{\rho}\,\{\langle 1, a', a'', \_\rangle :: \rho_i\}) \tag{5.9}$$
$$+ \sum_{n=1}^{|a'_i|} \begin{matrix} [c_{i/n} < sa_a - 1] & \overline{a'_{i/n}}.A(\tilde{z}, id, \tilde{\rho}\,\{c_{i/n}{+}{=}1\}) \\ +[c_{i/n} = sa_a - 1] & \overline{a''_{i/n}}y.(\!|P|\!)^{stop}_e \end{matrix}$$

$$(\!|ay.P|\!)_e = \sum_{n=1}^{|a'_i|} a'_{i/n}.A(\tilde{z}, id, \tilde{\rho}) + a''_{i/n}y.(\!|P|\!)^{stop}_e \tag{5.10}$$
$$+ a\langle a', a'', pid\rangle.\begin{pmatrix} [pid \in k_i] & (A(\tilde{z}, id, \tilde{\rho}) \mid \overline{pid}\langle id, a', \text{stop}\rangle) \\ \mid [pid \notin k_i] & (A(\tilde{z}, id, \tilde{\rho}\,\{\langle \_, a', a'', pid\rangle :: \rho_i\}) \mid \overline{pid}\langle id, a', \text{ajout}\rangle) \end{pmatrix}$$

où $r_{a'} = r_{a''} = r'_a$ et $\{a', a'', pid\} \cap (fn(A) \cup fn(P) \cup n(\tilde{z}) \cup n(\tilde{\rho})) = \emptyset$.

suivants :

$$
\begin{aligned}
A_1(a, id, c_1, a'_1, a''_1, k_1) \triangleq \\
&\langle a', a'', pid\rangle.A_1(a, id, 1::c_1, a'::a'_1, a''::a''_1, \_::k_1) \\
&+\sum_{n=1}^{|a'_1|} \begin{array}{ll} [c_{1/n} < sa_a - 1] & \overline{a'_{1/n}}.A_1(a, id, c_1\left\{c_{1/n}+=1\right\}, a'_1, a''_1, k_1) \\ +[c_{1/n} = sa_a - 1] & \overline{a''_{1/n}}.\mathbf{0} \end{array}
\end{aligned}
$$

$$
\begin{aligned}
A_2(a, id, c_1, a'_1, a''_1, k_1) \triangleq \\
&\sum_{n=1}^{|a'_1|} a'_{1/n}.A_2(a, id, c_1, a'_1, a''_1, k_1) + a''_{1/n}y.\mathbf{0} \\
&+a\langle a', a'', pid\rangle.\left(\begin{array}{ll} [pid \in k_1] & A_2(a, id, c_1, a'_1, a''_1, k_1) \\ |\ [pid \notin k_1] & A_2(a, id, \_::c_1, a'::a'_1, a''::a''_1, pid::k_1) \end{array}\right)
\end{aligned}
$$

$$A(a) \triangleq A_1(a, \nu id, \emptyset, \emptyset, \emptyset, \emptyset) \mid A_2(a, \nu id, \emptyset, \emptyset, \emptyset, \emptyset) \mid A_2(a, \nu id, \emptyset, \emptyset, \emptyset, \emptyset)$$

La définition 5.2 ajoute également des actions instantanées sur les canaux identifiant les processus. Comme nous le verrons dans la sous-section suivante, ces actions ne sont pas nécessaires pour la correction de la transformation proposée. Nous avons toutefois introduit ces actions pour supprimer explicitement des listes les canaux devenus obsolètes au cours de l'exécution des communications. En effet, nous pouvons remarquer que les canaux introduits lors de la construction peuvent devenir inutiles dans les cas suivants :

- Lorsqu'un processus A initie une deuxième communication avec le même processus B, ce dernier le détecte et l'ignore (grâce à l'identifiant reçu) ; cependant le processus A enregistre tout de même les canaux privés envoyés, oubliés par B.
- Lorsqu'un processus A initie une communication avec B et que B devient un autre processus avant la fin de la communication, A garde en mémoire les canaux privés partagés avec B.

Afin d'optimiser la mémoire utilisée pour stocker les canaux privés, nous proposons une méthode de nettoyage explicite. À tout processus, nous ajoutons une action de lecture sur son canal identifiant attendant un triplet $\langle pid, a', s\rangle$ (équation (5.5)) : pid est un canal identifiant d'un autre processus, a' est un canal privé supposé connu par notre processus ($\exists i, n, a'_{i/n} = a'$) ; lorsque s vaut stop, les n^e éléments des listes c_i, a'_i, a''_i et k_i sont supprimés ; lorsque s vaut ajout, pid est enregistré dans le n^e élément de la liste k_i. Les actions d'écriture suivantes sont alors ajoutées :

- Lorsqu'un processus effectue une transition, il écrit sur tous les canaux identifiants connus (référencés dans ses listes k) son identifiant avec le canal a' partagé et la valeur stop (équation (5.7)).
- Lorsqu'un processus récepteur ignore l'initiation d'une communication, ce dernier écrit sur le canal identifiant de l'émetteur son identifiant avec le canal reçu a' et la valeur stop.
- Lorsqu'un processus récepteur accepte l'initiation d'une communication, ce dernier écrit sur le canal identifiant de l'émetteur son identifiant avec le canal reçu a' et la valeur ajout.

Enfin, nous remarquons que la transformation présentée repose sur la capacité à avoir un nombre d'actions en compétition dépendant des paramètres de l'instanciation du processus. Toutefois, si le nombre de processus agissant en concurrence sur un même canal est toujours borné, et si cette borne est connue, la taille des listes employées peut alors être constante, donnant une compétition entre un nombre fixe d'actions. Cette restriction est similaire à celle imposée sur les modèles en π-calcul stochastique pour leur vérification formelle en PRISM telle que proposée par Norman et coll. (2009) et discutée en section 5.5.

$$\mathsf{TAU}_{\mathsf{wait}}\ \frac{A(\tilde{z}) \xrightarrow{\tau_a}_E B(\tilde{w})}{A_e(\tilde{z},id,\tilde{\rho}) \xrightarrow{\tau_a}_e A_e(\tilde{z},id,\tilde{\rho}\{c_i+\!\!=1\})}\, c_i<sa_a-1 \qquad \mathsf{TAU}_{\mathsf{eff}}\ \frac{A(\tilde{z}) \xrightarrow{\tau_a}_E B(\tilde{w})}{A_e(\tilde{z},id,\tilde{\rho}) \xrightarrow{\tau_a}_e B_e(\tilde{w},\nu id,\tilde{\varrho}^0)}\, c_i=sa_a-1$$

$$\mathsf{OUT}_{\mathsf{init}}\ \frac{A(\tilde{z}) \xrightarrow{\bar{a}y}_E B(\tilde{w})}{A_e(\tilde{z},id,\tilde{\rho}) \xrightarrow{\bar{a}\langle \nu a',\nu a'',id\rangle}_e A_e(\tilde{z},id,\tilde{\rho}\{\langle 1,a',a'',\_\rangle{:}\tilde{\rho}_i\})}$$

$$\mathsf{IN}_{\mathsf{init}}\ \frac{A(\tilde{z}) \xrightarrow{ay}_E B(\tilde{w})}{A_e(\tilde{z},id,\tilde{\rho}) \xrightarrow{a\langle a',a'',pid\rangle}_e A_e(\tilde{z},id,\tilde{\rho}\{\langle \_,a',a'',pid\rangle{:}\tilde{\rho}_i\})}\, pid\notin k_i$$

$$\mathsf{OUT}_{\mathsf{wait}}\ \frac{A(\tilde{z}) \xrightarrow{\bar{a}y}_E B(\tilde{w})}{A_e(\tilde{z},id,\tilde{\rho}) \xrightarrow{\bar{a'}}_e A_e(\tilde{z},id,\tilde{\rho}\{c_{i/n}+\!\!=1\})}\, \exists a'_{i/n}=a', c_{i/n}<sa_a-1 \qquad \mathsf{IN}_{\mathsf{wait}}\ \frac{A(\tilde{z}) \xrightarrow{ay}_E B(\tilde{w})}{A_e(\tilde{z},id,\tilde{\rho}) \xrightarrow{a'}_e A_e(\tilde{z},id,\tilde{\rho})}\, \exists a'_{i/n}=a'$$

$$\mathsf{OUT}_{\mathsf{eff}}\ \frac{A(\tilde{z}) \xrightarrow{\bar{a}y}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho}) \xrightarrow{\overline{a''y}}_e B_e(\tilde{w},\tilde{\varrho}^0)}\, \exists a''_{i/n}=a'', c_{i/n}=sa_a-1 \qquad \mathsf{IN}_{\mathsf{eff}}\ \frac{A(\tilde{z}) \xrightarrow{ay}_E B(\tilde{w})}{A_e(\tilde{z},id,\tilde{\rho}) \xrightarrow{a''y}_e B_e(\tilde{w},\tilde{\varrho}^0)}\, \exists a''_{i/n}=a''$$

Figure 5.3 – Transitions dérivées de la définition 5.2 pour les actions internes et les communications. i est l'indice de l'action dans la compétition du processus $S\pi_{Er}$. $\tilde{\varrho}$ est défini comme $\tilde{\rho}$.

5.3.3 Correction de la construction

Cette sous-section établit les lemmes nécessaires montrant la préservation du comportement quantitatif par la construction en $S\pi_e$ des processus $S\pi_{Er}$. Les actions agissant sur les canaux identifiant les processus sont ici ignorées : il est facile de voir qu'elles n'affectent pas les transitions entre les processus et n'engendrent jamais de comportements cycliques infinis.

Ci-après, les processus A, B, C, D sont supposés distincts. Le processus $S\pi_e$ résultant de la transformation par $(\!|\cdot|\!)_e$ du processus $S\pi_{Er}$ $A(\tilde{z})$ est noté $A_e(\tilde{z}, id, \tilde{\rho})$ (nous généralisons cette notation aux autres processus). Les transitions entre les processus $S\pi_e$ (resp. $S\pi_{Er}$) sont dénotées par $\rightarrow_e$ (resp. $\rightarrow_E$) ; $\rightarrow_e^*$ est une suite finie de transitions $\rightarrow_e$.

Partant de la définition 5.2, nous établissons dans un premier temps les règles de transitions pour les processus $S\pi_e$ en fonction du processus $S\pi_{Er}$ original, listées dans la figure 5.3. Les lemmes 5.1 et 5.2 établissent la correction du comportement qualitatif entre les processus $S\pi_{Er}$ et $S\pi_e$: une transition de A vers B est possible si et seulement si une transition est possible de A_e vers B_e. Enfin, les lemmes 5.3 et 5.4 établissent la correction du comportement quantitatif obtenu pour les actions internes et les communications, respectivement.

Lemme 5.1. $A_e(\tilde{z}, id^1, \tilde{\rho}) \rightarrow_e B_e(\tilde{w}, id^2, \tilde{\varrho}) \Rightarrow A(\tilde{z}) \rightarrow_E B(\tilde{w})$.

Démonstration. Évident étant donnée la définition 5.2. □

Lemme 5.2. $A(\tilde{z}) \rightarrow_E B(\tilde{w}) \Longrightarrow A_e(\tilde{z}, id^1\tilde{\rho}^0) \rightarrow_e^* B_e(\tilde{w}, id^2\tilde{\varrho}^0)$.

Démonstration. D'après les transitions référencées dans la figure 5.3, il existe toujours une suite finie de transitions annotées $init$ ou $wait$ menant aux transitions annotées eff correspondantes. □

Lemme 5.3. *La durée de* $A_e(\tilde{z}, id^1, \tilde{\rho}^0) \xrightarrow{\tau_a}_e^* B_e(\tilde{w}, id^2, \tilde{\varrho}^0)$ *suit la même distribution que la durée de la transition* $A(\tilde{z}) \xrightarrow{\tau_a}_E B(\tilde{w})$.

Démonstration. Au regard des transitions $\mathsf{TAU}_{\mathsf{wait}}$ et $\mathsf{TAU}_{\mathsf{eff}}$, il existe une et une seule suite de transitions selon τ_a menant de $A_e(\tilde{z}, id^1, \tilde{\rho}^0)$ à $B_e(\tilde{w}, id^2, \tilde{\varrho}^0)$: $\underbrace{\mathsf{TAU}_{\mathsf{wait}}, \ldots, \mathsf{TAU}_{\mathsf{wait}}}_{sa_a - 1}, \mathsf{TAU}_{\mathsf{eff}}$. La durée de chaque transition suivant une variable exponentielle avec un taux $r_a.sa_a$, la durée de cette suite de transitions est bien une variable aléatoire distribuée selon Erlang avec un taux r_a et un facteur d'absorption de stochasticité sa_a. □

Lemme 5.4. *La durée de* $A_e(\tilde{z}, id^1, \tilde{\rho}_1) \mid C_e(\tilde{z}', id^2, \tilde{\rho}'_1) \xrightarrow{\dot{a}}_e^* B_e(\tilde{w}, id^3\tilde{\varrho}_1) \mid D_e(\tilde{w}', id^4, \tilde{\varrho}'_1)$ *suit la même distribution que la durée de la transition* $A(\tilde{z}) \mid C(\tilde{z}') \xrightarrow{\dot{a}}_E B(\tilde{w}) \mid D(\tilde{w}')$.

Démonstration. Au regard des transitions $\mathsf{OUT}_{\mathsf{init}}$, $\mathsf{IN}_{\mathsf{init}}$, $\mathsf{OUT}_{\mathsf{wait}}$, $\mathsf{IN}_{\mathsf{wait}}$, $\mathsf{OUT}_{\mathsf{eff}}$ et $\mathsf{IN}_{\mathsf{eff}}$, la séquence suivante existe : $\mathsf{OUT}_{\mathsf{init}} \mid \mathsf{IN}_{\mathsf{init}}, \underbrace{\mathsf{OUT}_{\mathsf{wait}} \mid \mathsf{IN}_{\mathsf{wait}}, \ldots, \mathsf{OUT}_{\mathsf{wait}} \mid \mathsf{IN}_{\mathsf{wait}}}_{sa_a - 2}, \mathsf{OUT}_{\mathsf{eff}} \mid \mathsf{IN}_{\mathsf{eff}}$. La durée de chaque transition étant une variable exponentielle avec un taux $r_a.sa_a$, la durée de cette suite de transitions est bien une variable aléatoire distribuée selon Erlang avec un taux r_a et un facteur d'absorption de stochasticité sa_a.

Nous remarquons finalement que la séquence présentée ci-dessus est la seule séquence possible menant de $A_e(\tilde{z}, id^1, \tilde{\rho}_1) \mid C_e(\tilde{z}', id^2, \tilde{\rho}'_1)$ à $B_e(\tilde{w}, id^3\tilde{\varrho}_1) \mid D_e(\tilde{w}', id^4, \tilde{\varrho}'_1)$. En effet, la transition $\mathsf{IN}_{\mathsf{init}}$ ne peut s'effectuer qu'une seule fois, empêchant toute séquence superflue de transitions entre ces deux processus. □

5.3.4 Exemple Simple

Nous appliquons finalement les résultats obtenus dans cette section à un exemple simple : un processus se répétant infiniment (équation (5.11)).

$$A(l) \triangleq \tau_a.A(1-l) \qquad l \in \{0,1\} \tag{5.11}$$

Démarrant avec $A(0)$, nous nous attendons ainsi à observer une séquence infinie de changements de valeur de l $(0, 1, 0, 1, \ldots)$. Entre chaque transition, une transition interne τ_a est exécutée.

Le résultat de la traduction du processus $S\pi_{Er}$ défini par l'équation (5.11) vers un processus $S\pi_e$ est donné par l'équation (5.12), où le nettoyage des canaux a été omis.

$$A(l, id, c_1, a'_1, a''_1, k_1) \triangleq \tau_a.\left(\begin{array}{ll} [c_1 < sa_a - 1] & A(l, id, c_1 + 1, a'_1, a''_1, k_1) \\ \mid [c_1 = sa_a - 1] & A(1-l, \nu id, 0, \emptyset, \emptyset, \emptyset) \end{array}\right) \tag{5.12}$$

La figure 5.4 montre l'évolution au cours du temps de la valeur de l'argument l du processus A au cours de différentes simulations du processus $[\![A(0)]\!]_e = A(0, \nu id, 0, \emptyset, \emptyset, \emptyset)$ avec SPiM (Phillips & Cardelli, 2007; Phillips, SPiM). Pour chaque simulation, un même taux d'utilisation a été spécifié pour différents facteurs d'absorption de stochasticité.

Sans absorption de stochasticité, une forte variance de la durée des actions internes est observée, conformément à la loi de distribution exponentielle. En augmentant le facteur d'absorption de stochasticité, cette variance est réduite. Des oscillations régulières sont observées avec un facteur élevé d'absorption.

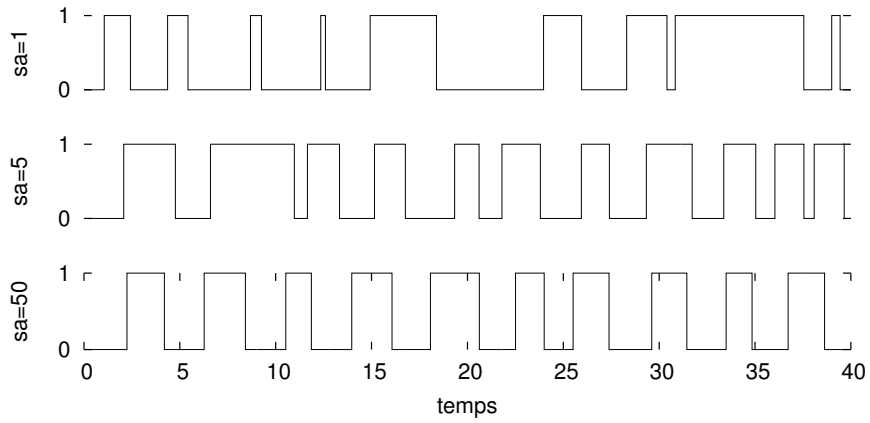


Figure 5.4 – Évolution de la valeur l au cours des simulations du processus $S\pi_{Er}$ défini par l'équation (5.11) avec un taux d'utilisation $\frac{1}{2}$ et différents facteurs d'absorption de stochasticité sa.

5.4 Intervalles de Tir

Dans cette section, nous mettons en avant une relation entre le taux d'utilisation et le facteur d'absorption de stochasticité d'une action et l'intervalle de temps durant lequel cette action sera tirée avec une certaine confiance. Cet intervalle de temps est appelé l'*intervalle de tir* (définition 5.3). Cette section donne également un ensemble d'outils statistiques pour aider la modélisation et la vérification des systèmes temporels et stochastiques dont la durée des transitions suit une distribution d'Erlang.

Définition 5.3 (Intervalle de tir)**.** Étant donné un taux d'utilisation r et un facteur d'absorption de stochasticité sa, l'*intervalle de tir* pour un niveau de confiance $1-\alpha$ est noté $FI_\alpha(r, sa) = [d; D]$ où les probabilités de tirer une action avant le temps d et D sont données par $F_{r,sa}(d) = \frac{\alpha}{2}$ et $F_{r,sa}(D) = 1 - \frac{\alpha}{2}$, respectivement ; $F_{r,sa}$ est la fonction de répartition définie par l'équation (5.2) (section 5.3.1 page 64).

Lors de la phase de conception d'un modèle, ou, plus spécifiquement, lors de sa paramétrisation, la spécification des durées des transitions par un intervalle de tir peut être plus naturelle que la spécification de la paire taux d'utilisation et absorption de stochasticité. Nous proposons alors des estimateurs et des fonctions d'approximation pour convertir un intervalle de tir en paramètres stochastiques.

Ainsi, le facteur d'absorption de stochasticité peut avoir plusieurs fonctions, selon la nature du système modélisé. D'un côté, l'absorption de stochasticité exprime la confiance en la durée précise de l'action : plus cette confiance est haute, plus le facteur d'absorption peut être important, et plus la variance de la durée sera réduite. D'un autre côté, l'absorption de stochasticité permet de reproduire des actions intrinsèquement aléatoires, mais dont les bornes de durée sont connues.

Nous soulignons l'existence d'autres outils de manipulation des distributions d'Erlang (et plus généralement, des distributions gamma), tels que des méthodes pour inférer le paramètre de forme (Zaigraev & Podraza-Karakulska, 2008) et de taux (Mi & Naranjo, 2003) d'une distribution gamma à partir d'un ensemble de données temporelles (provenant, par exemple, d'expériences). La conver-

sion des paramètres d'une distribution gamma en paramètres d'une distribution d'Erlang est discutée dans la suite de cette section.

5.4.1 Des Paramètres Stochastiques aux Intervalles de Tir

Étant donnés le taux et l'absorption de stochasticité d'une action, nous nous intéressons au calcul de l'intervalle de confiance du temps auquel l'action sera tirée. Soit $1-\alpha$ le niveau de confiance recherché pour l'intervalle de tir. Nous cherchons d et D tels que $F_{r,sa}(d) = \frac{\alpha}{2}$ et $F_{r,sa}(D) = 1-\frac{\alpha}{2}$, où $F_{r,sa}$ est la fonction de répartition de la distribution d'Erlang avec un taux r et un facteur d'absorption de stochasticité sa (équation (5.2)). La fonction qui associe à $d, 0 \leq d \leq 1$ le temps t tel que $F_{r,sa}(t) = d$ est la fonction *quantile*, notée F^{-1}.

De par sa relation avec la *fonction gamma incomplète*, la fonction quantile de la distribution gamma, et donc de la distribution d'Erlang, ne possède pas de forme analytique simple, et ne peut pas être utilisée directement (Wilkinson, 2006). Cependant, des algorithmes efficaces d'approximation de F^{-1} existent (Best & Roberts, 1975; DiDonato & Morris, 1986). L'outil statistique populaire *R* (R Development Core Team, 2009) possède une implémentation d'un tel algorithme. *R* étant distribué avec une bibliothèque de fonctions programmées en langage *C*, l'accès à l'implémentation depuis des programmes indépendants est aisé. Le calcul des quantiles de la distribution d'Erlang avec *R* se fait via la fonction `qgamma`. Voici un exemple d'une session *R* qui calcule l'intervalle de tir d'une action avec un taux d'utilisation r et une absorption de stochasticité sa :

$$d \leftarrow \texttt{qgamma}(\alpha/2, \texttt{shape} = sa, \texttt{rate} = r * sa)$$
$$D \leftarrow \texttt{qgamma}(1-\alpha/2, \texttt{shape} = sa, \texttt{rate} = r * sa)$$

La figure 5.5 montre l'influence du taux d'utilisation et du facteur d'absorption de stochasticité sur l'intervalle de tir associé.

5.4.2 Des Intervalles de Tir vers les Paramètres Stochastiques

Étant donné un intervalle de tir $[d; D]$ pour un niveau de confiance $1-\alpha$, nous cherchons le taux r et le facteur d'absorption de stochasticité sa tels que $FI_\alpha(r, sa) = [d, D]$. Pour résoudre ce problème, nous avons construit des estimateurs de r et sa (respectivement $\hat{r}_\alpha$ et $\hat{sa}_\alpha$) en fonction des bornes inférieure et supérieure de l'intervalle de tir, d et D, respectivement.

Dans un premier temps, nous relâchons la contrainte d'avoir un entier comme facteur d'absorption de stochasticité. Nous considérons alors la distribution gamma avec un taux d'utilisation $r \in \mathbb{R}_+^*$ et un facteur d'absorption de stochasticité $sa \in \mathbb{R}_+^*$. Comme il n'existe pas d'expression analytique de la fonction quantile de la distribution gamma, nous ne pouvons pas exprimer analytiquement r et sa en fonction de l'intervalle de confiance. Des estimateurs ont alors été obtenus par une régression sur un ensemble (généré automatiquement) de taux et d'absorption de stochasticité pour lesquels l'intervalle de tir à été calculé. Pour cette section, d et D ont été calculés pour des niveaux de confiance de 90%, 95% et 99% et pour tout $\log sa$ entre 0 et 4.5 (par pas de 0.1) et pour tout $\log r$ entre -8 et 2 (par pas de 0.1). La figure 5.6 montre la carte des valeurs des paramètres r et sa en fonction des bornes d et D des intervalles de tir pour la génération obtenue.

À partir des courbes en 3-D (figure 5.6), nous avons manuellement déterminé les régressions correspondant aux données. Les paramètres numériques des régressions obtenues pouvant dépendre

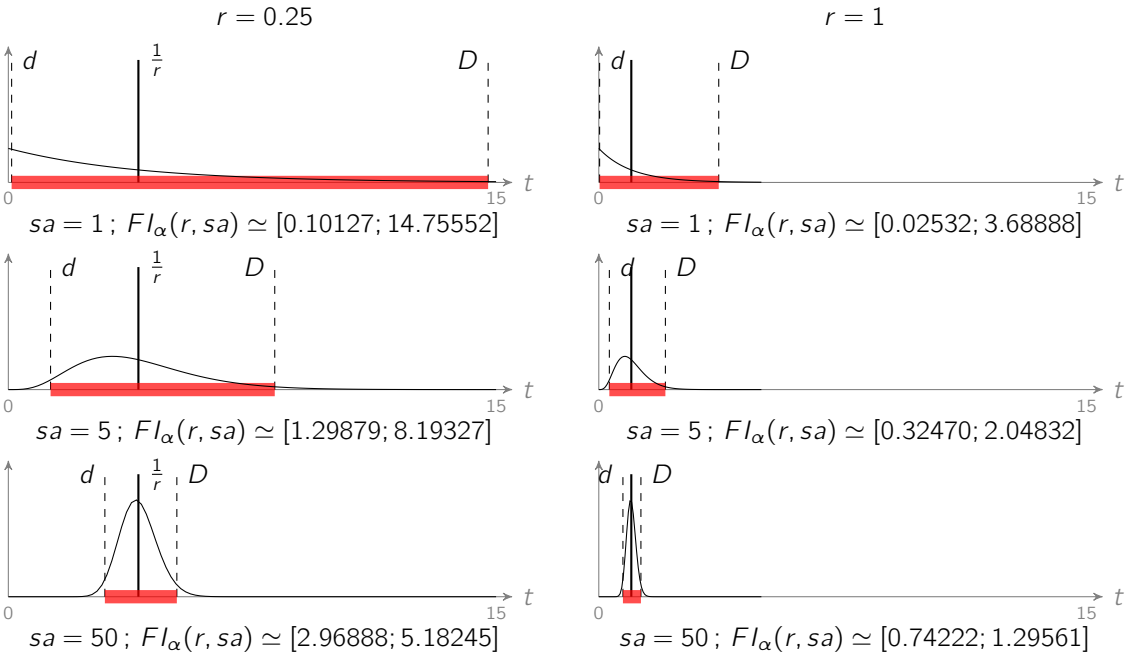


Figure 5.5 – Évolution de l'intervalle de tir quand l'absorption de stochasticité sa augmente. La ligne verticale en trait plein représente la durée moyenne obtenue pour chaque couple de paramètres. Le niveau de confiance a été fixé à 95% ($\alpha = 0.05$).

du niveau de confiance choisi, nous les avons abstraits. L'estimateur que nous proposons pour le taux d'utilisation, noté $\hat{r}_\alpha$, est donné par l'équation (5.13) ; celui que nous proposons pour le facteur d'absorption de stochasticité, noté $\hat{sa}_\alpha$ est donné par l'équation (5.14) (nous supposons $d > 0$).

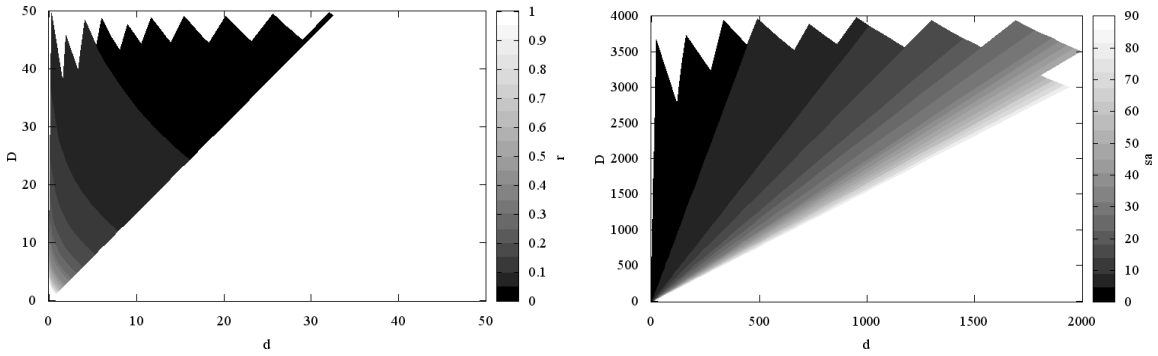
$$\hat{r}_\alpha = (w + xe^{-y.d})(d + D)^{-1} \tag{5.13}$$

$$\hat{sa}_\alpha = e^{u\left(\frac{D}{d}\right)^v}, \tag{5.14}$$

où u, v, w, x, y sont les paramètres dépendant du niveau de confiance $1 - \alpha$ choisi.

Les paramètres de ces expressions ont alors été estimés par l'outil *R*, qui ajuste les équations (5.13) et (5.14) aux données générées. Le tableau 5.2 résume les estimateurs trouvés pour r et sa à différents niveaux de confiance. Le manque d'expressions analytiques utilisables concernant la distribution gamma rend difficile l'évaluation de tels estimateurs. La figure 5.7 tente d'établir la qualité de ces estimateurs en comparant, pour d'autres données générées, la valeur attendue et la valeur estimée. Ce type d'évaluation est comparable aux matrices de confusion, utilisées notamment en apprentissage automatique (Kohavi & Provost, 1998). Il est important de remarquer que les estimateurs obtenus sont des fonctions standards, c.-à-d. qu'à chaque intervalle de tir correspond un seul taux d'utilisation et un seul facteur d'absorption de stochasticité.

Reste à traiter la contrainte qui impose des facteurs d'absorption de stochasticité *entiers*. De par la nature des estimateurs, si l'absorption estimée n'est pas un nombre entier, il n'existe pas de distribution d'Erlang correspondant à l'intervalle de tir donné. Partant de cette observation, la recherche d'une approximation des paramètres de la distribution d'Erlang est à la discrétion du concepteur du modèle. Nous soulignerons toutefois qu'arrondir le facteur d'absorption estimé à l'entier supérieur (resp. inférieur) résultera en un intervalle de tir inclus par (resp. contenant)



$r_{0.05}$ $sa_{0.05}$

Figure 5.6 – Carte du taux d'utilisation r (gauche) et du facteur d'absorption de stochasticité sa (droite) en fonction des bornes d et D d'un intervalle de tir $FI_{0.05} = [d; D]$ pour des données générées.

Tableau 5.2 – Estimateurs des paramètres r et sa obtenus pour un intervalle de tir $[d; D]$ avec différents niveaux de confiance.

	$\hat{r}_\alpha$	$\hat{sa}_\alpha$
$1-\alpha = 0.90$	$(2.13 + 2.89e^{-44.46d})(d+D)^{-1}$	$e^{6.39\left(\frac{d}{D}\right)^{-0.66}}$
$1-\alpha = 0.95$	$(2.06 + 1.93e^{-36.49d})(d+D)^{-1}$	$e^{6.41\left(\frac{d}{D}\right)^{-0.87}}$
$1-\alpha = 0.99$	$(2.03 + 1.39e^{-33.33d})(d+D)^{-1}$	$e^{6.41\left(\frac{d}{D}\right)^{-1.04}}$

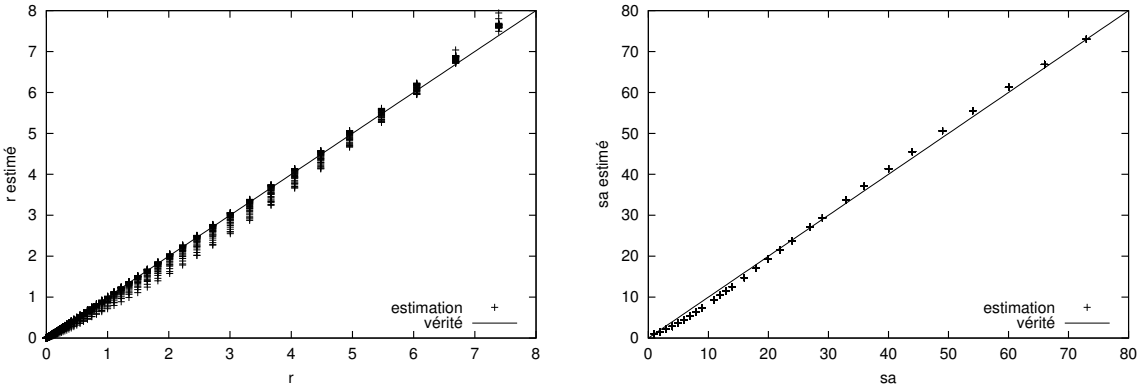
l'intervalle de tir original. Ainsi découle une approximation supérieure et inférieure des couples de paramètres stochastiques correspondant à un intervalle de tir arbitraire. Enfin, nous observons que l'estimation des paramètres stochastiques d'un intervalle de tir étant une simple évaluation de la fonction exponentielle, elle est très rapide à calculer.

5.4.3 Sur la Séquence des Actions

En conclusion de cette section, nous discutons brièvement de la distribution d'une séquence d'actions et de sa relation avec les intervalles de tir.

Soit k actions avec respectivement des taux d'utilisation $r_1, \ldots, r_k$ et des facteurs d'absorption de stochasticité $sa_1, \ldots, sa_k$. En considérant que ces actions sont tirées successivement, quel est l'intervalle de tir de la somme de ces actions ?

Malheureusement, nous pouvons vérifier simplement que l'intervalle de tir $[d; D]$ d'une séquence d'actions distribuées selon Erlang n'est pas la somme des intervalles de tir des actions individuelles — c.-à-d. $d \neq d_1 + \cdots + d_k$ et $D \neq D_1 + \cdots + D_k$ où d_i (resp. D_i) est la borne inférieure (resp. supérieure) de l'intervalle de tir de la i^{e} action. Cependant, la distribution de la somme de variables aléatoires suivant la distribution d'Erlang avec des paramètres différents a été étudiée par Amari & Misra (1997) et Nadarajah (2008), et Favaro & Walker (2008) donnent une expression aisément calculable de la fonction de répartition d'une telle distribution. En ce sens, l'intervalle de tir de la somme de variables aléatoires suivant la distribution d'Erlang pourrait être calculé en utilisant des



évaluation de $\hat{r}_{0.05}$ évaluation de $\hat{sa}_{0.05}$

Figure 5.7 – Évaluation des estimateurs obtenus pour le taux d'utilisation r et le facteur d'absorption de stochasticité sa avec un niveau de confiance de 95% sur des données générées. Les points représentent la valeur estimée. Un estimateur parfait placerait tous les points sur la ligne de vérité.

techniques d'approximation standards de la fonction quantile (comme la recherche par bissection). L'opération duale consistant à inférer les paramètres de la somme d'Erlang à partir d'un intervalle de tir se heurte à plusieurs difficultés, comme la perte de l'unicité des solutions. Nous ne considérons pas ce problème dans le reste de ce chapitre. Il devrait toutefois être l'objet de travaux futurs.

5.5 Vérification Formelle avec PRISM

Le vérificateur de modèles probabilistes PRISM (Hinton et coll., 2006) apporte une vérification formelle efficace des CMTC. En PRISM, les transitions sont spécifiées par des modules PRISM. Chaque module possède un ensemble fini de variables locales. L'union des variables locales de tous les modules donne un état global du modèle, dénoté par V. Une transition est le résultat d'une action spécifiée comme ceci :

$$[act]\ guard \to r : (x_1' = u_1)\ \&\ \ldots\ \&\ (x_k' = u_k)$$

où act est le libellé optionnel de l'action, $guard$ est un prédicat sur V, x_i est une variable locale, et u_i une fonction sur V. $r \in \mathbb{R}_+^*$ est le taux d'utilisation de l'action, et est supposé à 1 lorsqu'il n'est pas spécifié. Pour être applicable, une action libellée doit être synchronisée avec une action d'un autre module possédant le même libellé. Le taux obtenu d'une telle action synchronisée est le produit des taux des deux actions. x_i' représente la valeur de x_i une fois que l'action a été appliquée.

Une traduction efficace du π-calcul distribué exponentiellement ($S\pi_e$) vers PRISM a été proposée par Norman et coll. (2009). Leur traduction requiert que la structure générale d'un processus soit ré-arrangeable dans la forme $P = \nu x_1 \ldots \nu x_k (P_1 | \ldots | P_n)$ où aucun P_i ne contient d'opérateur ν, ni d'appel récursif contenant l'opérateur $|$, principalement afin d'assurer un nombre fini d'états. Nous remarquons cependant, qu'étant donné un processus P en $S\pi_{Er}$ respectant ces contraintes, le processus $(\!|P|\!)_e$ construit en $S\pi_e$ ne respecte pas cette limitation. En effet, la construction proposée de l'action d'écriture en $S\pi_e$ (équation (5.9)) utilise une génération récursive de nouveaux canaux a', a''. Une solution est alors la traduction directe d'un processus en $S\pi_{Er}$ vers PRISM.

Dans cette section, la traduction des processus $S\pi_e$ vers PRISM proposée par Norman et coll. (2009) est adaptée à la traduction des processus $S\pi_{Er}$. Ceci permet la vérification formelle efficace des processus $S\pi_{Er}$ en PRISM, ce qui est un nouveau résultat. En fin de section, nous illustrons cette traduction sur un exemple simple.

5.5.1 Construction du Facteur d'Absorption de Stochasticité en PRISM

Soit P une expression en π-calcul stochastique de la forme $P = \nu x_1 \ldots \nu x_k(P_1| \ldots |P_n)$ où aucun P_i ne contient l'opérateur ν ou $|$. Ainsi, chaque processus P_i décrit une suite de compétitions et peut être représenté par un graphe de transitions où les nœuds sont les compétitions successives, annotées par $Q_i, R_i, \ldots$ (Norman et coll., 2009). Chaque $Q_i, R_i, \ldots$ représente donc un état du processus P_i. En utilisant la traduction détaillée dans (Norman et coll., 2009), le modèle PRISM correspondant à P peut être calculé. Il résulte n modules PRISM, un par P_i, chacun possédant une variable s_i pour l'état courant du processus P_i. Les variables représentant les canaux à envoyer ou à recevoir sont également attachées aux modules. Les actions résultantes ont trois formes possibles :

$$[]\ (s_i = Q_i)\ \&\ M \rightarrow r_t : (s_i' = R_i) \tag{5.15}$$

$$[a\_P_i\_P_j\_y]\ (s_i = Q_i)\ \&\ M \rightarrow r_a : (s_i' = R_i) \tag{5.16}$$

$$[a\_P_j\_P_i\_y]\ (s_i = Q_i)\ \&\ M \rightarrow (s_i' = R_i)\ \&\ (z' = y) \tag{5.17}$$

Chacune de ces formes représente le processus P_i à l'état Q_i appliquant une certaine action sous la condition M et changeant alors en l'état R_i. Ces actions sont respectivement la transition interne (équation (5.15)), l'écriture (libre ou liée) de y sur un canal a vers P_j (équation (5.16)), et la lecture de y en tant que z sur un canal a depuis P_j (équation (5.17)). En fonction de la portée du canal y envoyé, des conditions supplémentaires sont ajoutées à M (nous ne détaillons pas cette partie). Dans le reste de cette section, nous supposons que les actions libellées identiquement ont des gardes disjointes, c.-à-d. qu'elles ne font jamais partie de la même compétition.

La gestion de l'absorption de stochasticité dans cette traduction est ajoutée d'une manière similaire à la construction du facteur d'absorption de stochasticité en $S\pi_e$ présentée en section 5.3 : un compteur est lié à chaque action. Quand ce compteur atteint la valeur du facteur d'absorption de stochasticité, l'action est appliquée. Les actions correspondant aux transitions internes sont libellées par P_i_t, avec τ_t la transition interne de P_i. L'ensemble des libellés des transitions internes et des écritures par le processus P_i est dénoté par $\mathcal{L}_{P_i\Diamond} = \{a\_P_i\_P_j\_y, \ldots, P_i\_t, \ldots\}$, et l'ensemble des libellés des lectures par le processus P_i est dénoté par $\mathcal{L}_{\Diamond P_i} = \{a\_P_j\_P_i\_y, \ldots\}$. Essentiellement, nous aurons un compteur pour chaque action possédant un libellé $l \in \mathcal{L}_{P_i\Diamond}$. Ce compteur est défini en tant que variable locale c_l à l'intérieur du module PRISM du processus P_i. Chaque fois qu'une écriture ou une transition interne est exécutée, le compteur correspondant est incrémenté d'une unité. L'application de l'action (c.-à-d. la mise à jour des variables d'état) n'est exécutée que lorsque ce compteur atteint le facteur d'absorption de stochasticité escompté.

Quand P_i change d'état, chaque compteur lié à son nouvel état doit être initialisé proprement. Cependant, comme PRISM interdit la mise à jour de variables appartenant aux autres modules, P_i ne peut pas directement mettre à jour les compteurs qu'il ne possède pas (liés aux actions libellées par $l \in \mathcal{L}_{\Diamond P_i}$). Pour contourner cette limitation, une variable booléenne d_l est définie dans le module P_i pour tout $l = a\_P_j\_P_i\_y \in \mathcal{L}_{\Diamond P_i}$, et est mise à vrai quand la réinitialisation de c_l est requise. Le module possédant c_l doit alors réinitialiser (à 1) le compteur quand d_l est vrai.

Nous soulignons que cet encodage proposé des transitions suivant Erlang en PRISM peut être appliqué à d'autres modèles que ceux résultant de la traduction d'une expression en π-calcul stochas-

tique. Le point délicat de cette construction est la réinitialisation de l'absorption de stochasticité dès que la synchronisation correspondante n'est plus possible. À chaque mise à jour, l'ensemble des synchronisations rendues impossibles doit être complètement caractérisé. Bien que cela demande une gestion précise des gardes des synchronisations, cela devrait être applicable à bon nombre de modèles PRISM.

Le reste de cette sous-section décrit les transformations nécessaires pour ajouter la gestion de l'absorption de stochasticité aux modules PRISM des processus P_i résultants de la traduction de Normal et coll. (Norman et coll., 2009).

Variables locales supplémentaires

Pour chaque libellé $l \in \mathcal{L}_{P_i\Diamond}$, la variable c_l représente le compteur de l'absorption de stochasticité de l'action.

$c\_l : [1..sa\_l]\ init\ 1;$

Pour chaque libellé $l \in \mathcal{L}_{\Diamond P_i}$, la variable booléenne d_l est vraie si P_i a changé son état alors que l'absorption de l'action libellée par l a déjà commencé. En d'autres termes, d_l est vrai si le compteur associé c_l doit être réinitialisé.

$d\_l : bool\ init\ false;$

Par la suite, la mise à jour PRISM correspondant à la réinitialisation de tous les compteurs d'absorption de stochasticité est dénotée par $R_{P_i\Diamond}$ (équation (5.18)). La mise à jour des variables d_l, $l \in \mathcal{L}_{\Diamond P_i}$, est dénotée par $S_{\Diamond P_i}$ (équation (5.19)). Essentiellement, d_l est changé à vrai si et seulement si le compteur associé c_l est différent de sa valeur initiale (1).

$$R_{P_i\Diamond} \stackrel{def}{=} \mathop{\&}_{l \in \mathcal{L}_{P_i\Diamond}} (c\_l' = 1) \tag{5.18}$$

$$S_{\Diamond P_i} \stackrel{def}{=} \mathop{\&}_{l \in \mathcal{L}_{\Diamond P_i}} (d\_l' = c\_l > 1) \tag{5.19}$$

où $\mathop{\&}_{i \in \{1,\ldots,k\}} u_i = u_1\ \&\ \cdots\ \&\ u_k$.

Transition interne

Soit $l = P_i\_t$ le libellé de l'action résultante de la traduction d'une transition interne τ_t. Le taux d'utilisation et le facteur d'absorption de stochasticité de cette transition interne sont respectivement r_t et sa_t. Cette action traduite en PRISM a la forme suivante :

$[]\ G \rightarrow r_t : U;$

où G est la garde de l'action et U est les mises à jour à effectuer. L'absorption de stochasticité de l'action est acquise en remplaçant la règle précédente par les actions ci-dessous :

$[]\ G\ \&\ (c\_l < sa_t) \rightarrow r_t * sa_t : (c\_l' = c\_l + 1);$
$[]\ G\ \&\ (c\_l = sa_t) \rightarrow r_t * sa_t : U\ \&\ R_{P_i\Diamond}\ \&\ S_{\Diamond P_i};$

Ainsi, les mises à jour U sont appliquées après exactement sa_t transitions à un taux $r_t \cdot sa_t$ depuis le dernier changement d'état de P_i.

Écriture sur un canal

Soit $l = a\_P_i\_P_j\_y$ le libellé de l'action résultant de la traduction d'une écriture d'un canal y sur un canal a. Le taux d'utilisation et le facteur d'absorption de stochasticité de cette action sont respectivement r_a et sa_a. Cette action traduite en PRISM est de la forme suivante :

$$[l]\ G \rightarrow r_a : U;$$

où G est la garde de l'action et U est les mises à jours à exécuter. L'absorption de stochasticité de l'action est acquise en remplaçant la règle précédente par les actions ci-dessous :

$$[l\_wait]\ G\ \&\ d\_l \rightarrow r_a * sa_a : (c\_l' = 2);$$
$$[l\_wait]\ G\ \&\ !d\_l\ \&\ (c\_l < sa_a) \rightarrow r_a * sa_a : (c\_l' = c\_l + 1);$$
$$[l]\ G\ \&\ !d\_l\ \&\ (c\_l = sa_a) \rightarrow r_a * sa_a : U\ \&\ R_{P_i\Diamond}\ \&\ S_{\Diamond P_i};$$

Pour exécuter les mises à jours U, P_i doit au préalable exécuter $sa_a - 1$ synchronisations sur le libellé l_wait, et finalement une synchronisation sur le libellé l. La règle où d_l est vrai correspond à la remise à 1 du compteur c_l. Ainsi, la valeur de ce compteur devient 2 une fois cette règle exécutée.

Lecture sur un canal

Soit $l = a\_P_j\_P_i\_y$ le libellé de l'action résultant de la traduction d'une lecture d'un canal y sur un canal a. Cette action traduite en PRISM est de la forme suivante :

$$[l]\ G \rightarrow U;$$

où G est la garde de l'action et U est les mises à jour à exécuter. L'absorption de stochasticité de l'action est acquise en remplaçant la règle précédente par les actions ci-dessous :

$$[l\_wait]\ G \rightarrow (d\_l' = false);$$
$$[l]\ G \rightarrow U\ \&\ R_{P_i\Diamond}\ \&\ S_{\Diamond P_i};$$

Pour exécuter les mises à jour U, P_i doit exécuter au préalable $sa_a - 1$ synchronisations sur le libellé l_wait, et finalement une synchronisation sur le libellé l. Au regard de la transformation correspondant à l'écriture sur canal par P_j, le compteur c_l est réinitialisé après une synchronisation sur l_wait. Ainsi, d_l doit être mis à faux dans la mise à jour pour que c_l puisse s'incrémenter lors des synchronisations futures.

En appliquant ces transformations, il est assuré que chaque mise à jour U est exécutée après une durée suivant la distribution de la somme de sa variables aléatoires de distribution exponentielle avec un taux $r \cdot sa$.

5.5.2 Exemple Simple

En tant qu'application de la vérification formelle avec PRISM de processus $S\pi_{Er}$, et en tant qu'illustration de l'approche générale présentée dans ce chapitre, nous proposons une étude du processus P défini par l'équation (5.20).

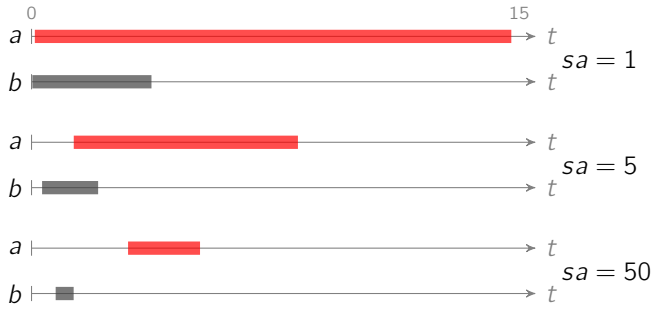


Figure 5.8 – Intervalles de tir avec un niveau de confiance de 95% pour a et b (équation (5.20)) avec différents facteurs d'absorption de stochasticité mais des taux d'utilisation constants $r_a = 0.25$ et $r_b = 1$.

$$
\begin{aligned}
A_1(a,b,c) &\triangleq a.A_0(c) + \overline{b}.A_1(a,b,c) & A_0(c) &\triangleq c.\mathbf{0} \\
B_1(a,b,c) &\triangleq b.B_0(c) + \overline{a}.B_1(a,b,c) & B_0(c) &\triangleq c.\mathbf{0} \\
P &\triangleq \nu a \nu b \nu c (A_1(a,b,c) | B_1(a,b,c)) & & \qquad (5.20)
\end{aligned}
$$

Intuitivement, deux scénarios sont possibles : soit A_1 écrit en premier sur b et B_1 devient B_0 et le système termine en étant bloqué ; soit B_1 écrit en premier sur a et A_1 devient A_0 et le système termine en étant bloqué.

Pour ce simple exemple , nous nous intéressons à réduire au plus proche de zéro la probabilité que A_1 devienne A_0, c.-à-d. la probabilité que B_1 écrive sur a. En tant que contraintes supplémentaires, les durées moyennes d'utilisation des canaux a et b sont fixées respectivement à 4 et 1 unité(s) de temps (c.-à-d., $r_a = 0.25$ et $r_b = 1$). Par exemple, de telles contraintes auraient pu avoir été imposées par des observations sur le système réel modélisé par P. La propriété à vérifier est exprimée en PRISM par `P=? [F (a=0)]`, ce qui signifie la probabilité que A_0 finisse par être exécuté.

Nous étudions d'abord ce modèle en tant que processus $S\pi_e$. Le listing 5.1 montre le résultat de la traduction de P en PRISM en suivant la méthode de Norman et coll. (2009). La vérification par PRISM de la propriété énoncée précédemment évalue la probabilité d'exécuter A_0 à 0.2.

Considérons maintenant P comme un processus $S\pi_{Er}$. Pour une raison de simplicité, nous attachons à a et b le même facteur d'absorption de stochasticité.

Au regard des intervalles de tir, nous cherchons un facteur d'absorption de stochasticité pour lequel l'intervalle de tir de b soit entièrement avant l'intervalle de tir de a. En calculant les intervalles de tir de ces deux actions avec différents facteurs d'absorption — comme montré dans la figure 5.8 — nous observons que la probabilité de tirer a avant b devrait être considérablement réduite avec un facteur d'absorption de stochasticité de 5. En utilisant un facteur d'absorption de stochasticité de 50, nous nous attendons alors à une probabilité de tir de a proche de 0, avec un niveau de confiance de 95%.

Afin de confirmer ces résultats, nous nous tournons vers la vérification formelle du processus P en $S\pi_{Er}$ avec PRISM. Le listing 5.2 montre la traduction de P en PRISM en utilisant la construction présentée précédemment. Avec une absorption de stochasticité de 5, la probabilité d'exécuter A_0

```
module proc_A
    a: [0..1] init 1;
    [a_B1_A1] (a=1) -> (a'=0);
    [b_A1_B1] (a=1) -> r_b: (a'=1);
endmodule

module proc_B
    b: [0..1] init 1;
    [b_A1_B1] (b=1) -> (b'=0);
    [a_B1_A1] (b=1) -> r_a: (b'=1);
endmodule
```

Listing 5.1 – Traduction en PRISM du modèle exemple en π-calcul stochastique distribué exponentiellement (équation (5.20)).

```
module proc_A
    a: [0..1] init 1;
    c_b_A1_B1: [1..sa_b] init 1;
    d_a_B1_A1: bool init false;

    [a_B1_A1_wait] (a=1) -> (d_a_B1_A1'=false);
    [a_B1_A1] (a=1) -> (a'=0) & (d_a_B1_A1'=false) & (c_b_A1_B1'=1);

    [b_A1_B1_wait] (a=1) & d_b_A1_B1 -> r_b: (c_b_A1_B1'=2);
    [b_A1_B1_wait] (a=1) & !d_b_A1_B1 & (c_b_A1_B1<sa_b) -> r_b:
        (c_b_A1_B1'=c_b_A1_B1+1);
    [b_A1_B1] (a=1) & !d_b_A1_B1 & (c_b_A1_B1=sa_b) -> r_b: (a'=1) &
        (c_b_A1_B1'=1) & (d_a_B1_A1'=c_a_B1_A1>1);
endmodule

module proc_B
    b: [0..1] init 1;
    c_a_B1_A1: [1..sa_a] init 1;
    d_b_A1_B1: bool init false;

    [b_A1_B1_wait] (b=1) -> (d_b_A1_B1'=false);
    [b_A1_B1] (b=1) -> (b'=0) & (d_b_A1_B1'=false) & (c_a_B1_A1'=1);

    [a_B1_A1_wait] (b=1) & d_a_B1_A1 -> r_a: (c_a_B1_A1'=2);
    [a_B1_A1_wait] (b=1) & !d_a_B1_A1 & (c_a_B1_A1<sa_a) -> r_a:
        (c_a_B1_A1'=c_a_B1_A1+1);
    [a_B1_A1] (b=1) & !d_b_A1_B1 & (c_a_B1_A1=sa_a) -> r_a: (b'=1) &
        (c_a_B1_A1'=1) & (d_b_A1_B1'=c_b_A1_B1>1);
endmodule
```

Listing 5.2 – Traduction en PRISM du modèle exemple en π-calcul stochastique distribué par Erlang (équation (5.20)).

est divisée par 100 (environ 0.02) comparé aux cas sans absorption de stochasticité. Augmenter cette absorption à 50 réduit cette probabilité à approximativement 10^{-11}.

5.6 Applicabilité

Dans cette section, nous discutons de l'applicabilité des résultats présentés dans ce chapitre. En premier lieu, la complexité et le passage à l'échelle de la traduction des modèles distribués par Erlang en modèles distribués exponentiellement sont abordés. Enfin, une étude de cas démontrant les bénéfices de l'utilisation du facteur d'absorption de stochasticité conclut cette section.

5.6.1 Complexités et Passage à l'Échelle

Les traductions proposées (vers le π-calcul stochastique ou vers PRISM) étant de simples réécritures sans duplication de termes, elles sont linéaires en la taille du modèle. L'ajout du facteur d'absorption de stochasticité dans les modèles en π-calcul stochastique devrait être gérable par les outils de simulations, les actions étant simplement décomposées en plusieurs étapes. Cependant, la vérification formelle des modèles avec de nombreuses actions ayant un fort facteur d'absorption de stochasticité peut souffrir d'une explosion combinatoire de l'espace des états à explorer. Des techniques pouvant permettre de surpasser cette difficulté sont discutées dans la section sur les travaux liés. Enfin, la conversion d'un intervalle de temps en paramètres de distribution d'Erlang étant une simple évaluation de la fonction exponentielle, elle est très efficace.

5.6.2 Étude de Cas

Nous montrons les bénéfices des contributions apportées par ce chapitre à travers l'étude d'un modèle biologique des processus de segmentation chez les métazoaires. Ce système a été étudié en utilisant des équations différentielles par François et coll. (2007). Nous proposons d'étendre l'étude du système en introduisant de la stochasticité dans les réactions, que nous modélisons en π-calcul stochastique. L'utilisation de l'absorption de stochasticité permet alors de reproduire et de vérifier des propriétés mettant en jeu le temps.

Le processus de segmentation est modélisé par un marqueur, appelé A, qui devient périodiquement actif ou inactif. L'activité du marqueur est contrôlée par une horloge C et un interrupteur global F. Tant que F est actif, l'horloge change régulièrement de niveau (actif ou inactif), et le marqueur, si désactivé, peut devenir actif. Enfin, lorsque l'horloge est active, elle peut désactiver le marqueur. Après une certaine durée, l'interrupteur se désactive, empêchant une nouvelle activation de l'horloge et du marqueur. Nous proposons une modélisation de ce système par l'expression en π-calcul stochastique suivante :

$$
\begin{aligned}
F_1(h) &::= \tau_f.F_0(h) \qquad F_0(h) ::= \bar{h}.\mathbf{0} \\
C_0(a,d,h) &::= \bar{a}.C_0(a,d,h) + \tau_c.C_1(a,d,h) + h.\mathbf{0} \\
C_1(a,d,h) &::= \bar{d}.C_1(a,d,h) + \tau_c.C_0(a,d,h) \\
A(l,a,d) &::= a.A(1,a,d) + d.A(0,a,d) \\
&\nu a \nu d \nu h(F_1(h)|C_0(a,d,h)|A(0,a,d))
\end{aligned}
$$

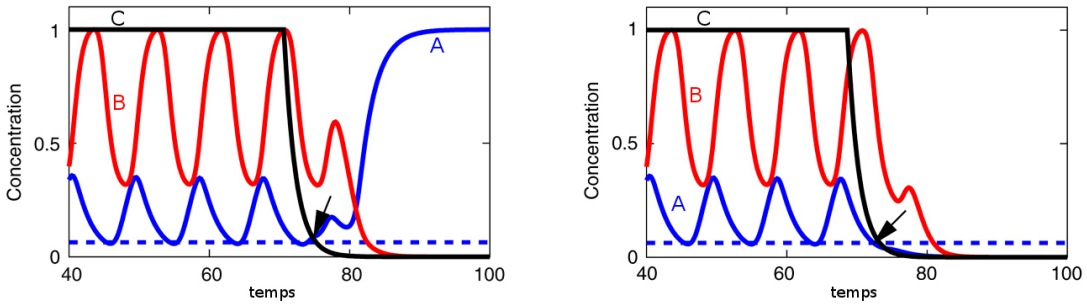


Figure 5.9 – Évolution du modèle de la segmentation au cours du temps tel présenté dans (François et coll., 2007) : en noir est la concentration de F, en rouge celle de C et en bleu celle de A. Deux destins sont possibles pour A : terminer avec un niveau élevé (gauche) ou faible (droite).

L'interrupteur, initialement actif (F_1), attend un moment avant de devenir inactif (F_0) ; F_0 écrit sur le canal h (pour *halt – arrêt*), indiquant que le système doit s'arrêter. Si l'horloge est inactive (C_0 est exécuté), elle écrit sur a pour activer le marqueur (A change son état l à 1 lors d'une lecture sur a). De manière concurrente, l'horloge inactive effectue une transition interne τ_c pour s'activer (C_1). Dès que C_0 lit sur h (c.-à-d. l'interrupteur est inactif), il devient le processus nul, désactivant la totalité du système. Enfin, l'horloge active écrit sur d pour désactiver le marqueur (A change son état l à 0 lors d'une lecture sur d) ; de manière concurrente, il effectue une transition interne τ_c pour se désactiver. La figure 5.9 illustre ce comportement tel que modélisé par François et coll..

Le défi de cette modélisation est d'obtenir un système produisant un nombre fixe de segments, c.-à-d. que le marqueur devienne actif un nombre fixe de fois. Nous utilisons dans un premier temps des actions avec une durée distribuée exponentiellement, et nous fixons les taux d'utilisation suivants : $r_f = 0.02$ (en moyenne, l'interrupteur reste actif pendant 50 unités de temps) ; $r_c = 0.1$ (en moyenne, l'horloge change de niveau toutes les 10 unités de temps) ; $r_a = r_d = 1$ (l'activation et la désactivation du marqueur durent 1 unité de temps en moyenne) ; et $r_h = 10$. Ainsi, en moyenne, nous espérons observer 3 activations du marqueur.

En utilisant PRISM, nous calculons la probabilité d'observer exactement 3 activations du marqueur, et obtenons 0.15, ce qui est plutôt faible. Ce résultat n'est pas surprenant à la vue de la grande variance de la distribution exponentielle. En réduisant la variance des délais τ_f et τ_c, nous avons l'intention de réduire la variance du nombre d'activations de l'horloge, et donc, par transitivité, la variance du nombre d'activations du marqueur. Cette réduction de variance est acquise par l'utilisation du facteur d'absorption de stochasticité. De façon arbitraire, nous fixons le facteur d'absorption de stochasticité $sa_c = 15$ (donnant un intervalle de tir pour τ_c entre 4.6 et 17.9 unités de temps, avec un niveau de confiance de 99%) ; et nous utilisons les estimateurs de la section 5.4.2 pour obtenir les paramètres stochastiques correspondant à un intervalle de tir entre 45 et 55 unités de temps (ce qui donne $r_f = 0.0202$ et $sa_f = 608$). Ces paramètres étant renseignés, et en utilisant la traduction de la distribution d'Erlang en PRISM proposée en section 5.5 (listing 5.3), la probabilité d'observer exactement 3 activations de marqueur croît à 0.86.

Le temps de calcul de la probabilité par PRISM augmente de quelques millisecondes à quelques minutes, montrant une large croissance de l'espace des états causée par l'introduction du facteur d'absorption de stochasticité. La figure 5.10 montre la sensibilité de la probabilité recherchée sur les facteurs d'absorption de stochasticité sa_c et sa_f.

```
module proc_a
    a: [0..1] init 0; // state
    marks: [0..10] init 0;
    [act] a=0 & marks<10 -> (a'=1) & (marks'=marks+1);
    [inh] a=1 -> (a'=0);
endmodule

module proc_c
    c: [0..2] init 0; // state
    c_wait: [1..sa_c] init 1;
    [] c=0 & c_wait<sa_c -> r_c*sa_c: (c_wait'=c_wait+1);
    [] c=0 & c_wait=sa_c -> r_c*sa_c: (c'=1) & (c_wait'=1);
    [] c=1 & c_wait<sa_c -> r_c*sa_c: (c_wait'=c_wait+1);
    [] c=1 & c_wait=sa_c -> r_c*sa_c: (c'=0) & (c_wait'=1);
    [halt] c=0 -> (c'=2);
    [act] c=0 -> r_a: (c'=0);
    [inh] c=1 -> r_a: (c'=1);
endmodule

module proc_f
    f: [0..1] init 1; // state
    f_wait: [1..sa_f] init 1;
    [] f=1 & f_wait<sa_f -> r_f*sa_f: (f_wait'=f_wait+1);
    [] f=1 & f_wait=sa_f -> r_f*sa_f: (f'=0) & (f_wait'=1);
    [halt] f=0 -> r_h: (f'=0);
endmodule
```

Listing 5.3 – Modèle PRISM obtenu par la traduction du modèle en π-calcul stochastique distribué selon Erlang de la segmentation des métazoaires.

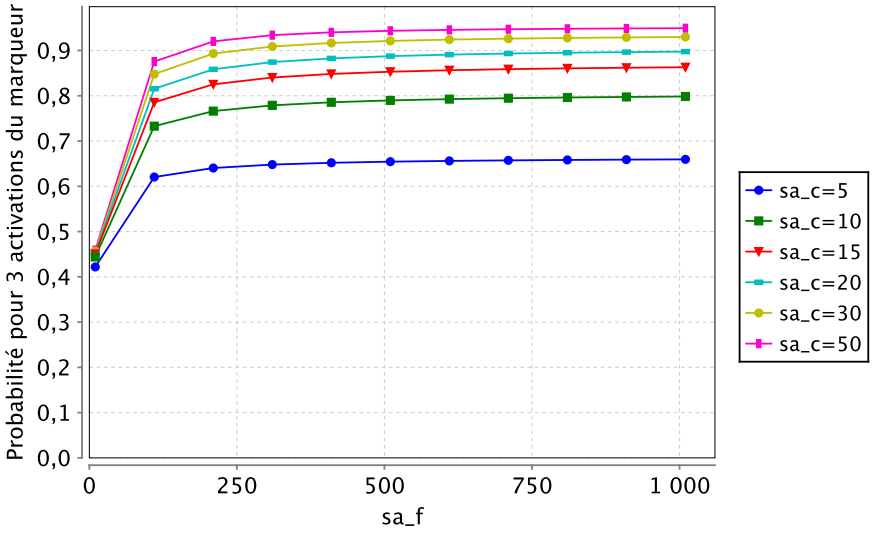


Figure 5.10 – Évolution de la probabilité d'observer exactement 3 activations du marqueur en fonction des facteurs d'absorption de stochasticité sa_c et sa_f.

5.7 Contributions et Travaux Liés

Les contributions principales de ce chapitre sont la construction du π-calcul stochastique distribué par Erlang en utilisant le π-calcul stochastique markovien, avec la construction de la distribution d'Erlang dans les modèles PRISM traduits depuis le π-calcul stochastique. Ce chapitre apporte également des outils établissant le lien entre les intervalles de temps et les paramètres stochastiques des distributions d'Erlang.

Nous notons que la simulation du π-calcul stochastique avec des distributions de probabilité arbitraires a été étudié dans (Priami, 1996). Cette simulation fonctionne en recalculant la distribution des transitions après qu'une transition a été choisie, pour prendre en compte le temps écoulé. Basée sur cette méthode, une implémentation de la simulation des processus non-markoviens pour le langage BlenX, proche du π-calcul stochastique, a été récemment proposée (Mura, Prandi, Priami & Romanel, 2009). Une autre méthode proposée dans (Gibson & Bruck, 2000), aborde la simulation de réactions chimiques non-markovienne, en identifiant précisément chaque molécule réagissant et en calculant *a priori* l'heure de la réaction, sélectionnant alors la plus faible. Le chapitre 6 proposera notamment une simulation non-markovienne du π-calcul stochastique basée sur cette méthode.

Les méthodes présentées dans ce chapitre permettent la vérification formelle des modèles en π-calcul stochastiques distribué sur Erlang, sous la restriction imposée par la traduction vers PRISM (section 5.5). Cette vérification est acquise grâce à la traduction d'un modèle utilisant les distributions d'Erlang en un modèle markovien. Alors que la vérification formelle par PRISM est gérable avec quelques transitions suivant une distribution d'Erlang, elle souffre de l'explosion combinatoire de l'espace des états lorsque de grands facteurs d'absorption de stochasticité sont spécifiés (section 5.6). Cependant, nous notons que les CMTC obtenues sont particulièrement structurées, suggérant que des approches telles que la réduction par symétrie (Kwiatkowska, Norman & Parker, 2006) ou les abstractions de séquences de transitions (Katoen, Klink, Leucker & Wolf, 2008) pourraient produire une vérification formelle plus efficace.

Peu de travaux appliquent la vérification formelle à des modèles utilisant des distributions de probabilités non exponentielles. Donatelli, Haddad & Moreaux (1998) proposent une caractérisation efficace de l'espace des états associé à un réseau de Petri avec des lois phase-types. López, Hermanns & Katoen (2001) étudient la vérification formelle de chaînes semi-markoviennes, où le temps passé dans les états suit une distribution de probabilité quelconque. Malgré quelques bon résultats sur la vérification de quelques propriétés, ils tirent une conclusion négative du fait que la vérification devient calculatoirement très complexe. Bryans, Bowman & Derrick (2003) apportent une vérification formelle efficace des automates stochastiques avec des distributions quelconques en utilisant une logique temporelle probabiliste simplifiée.

Bobbio & Horváth (2001) proposent de réduire la différence entre les extensions temporelles et stochastiques des réseaux de Petri. Ils définissent une nouvelle façon de décrire la structure des réseaux par des lois phase-type discrètes, et l'utilisent pour construire à la fois un modèle fonctionnel au sens des réseau de Petri temporisés ou stochastiques. Ces lois phase-type discrètes peuvent représenter des choix probabilistes ou non-déterministes.

5.8 Discussion

Dans ce chapitre, nous avons présenté une technique pour raffiner la notion de temps en π-calcul stochastique. Ce raffinement est acquis au travers d'un facteur d'absorption de stochasticité

qui réduit la variance autour de la durée moyenne d'une transition. Ceci permet la spécification des transitions par un intervalle de temps de tir pour un niveau de confiance donné. L'absorption de stochasticité est produite en remplaçant la distribution exponentielle des délais d'action par la distribution d'Erlang. Des estimateurs permettant le calcul des paramètres stochastiques des transitions correspondant à un intervalle de temps de tir voulu ont été produits pour ce chapitre. *Nous prétendons qu'une telle approche rend possible une spécification temporelle plus précise dans des systèmes stochastiques.*

Nous avons présenté une traduction du π-calcul stochastique distribué par Erlang vers le π-calcul distribué exponentiellement. En ce sens, les modèles ainsi raffinés peuvent être simulés et analysés avec les nombreux outils standards se basant sur la distribution exponentielle. La vérification formelle du π-calcul stochastique distribué par Erlang est possible en utilisant PRISM : nous avons étendu la traduction du π-calcul stochastique vers PRISM de Norman et coll. (2009) pour y construire le facteur d'absorption de stochasticité. Nous avons soulevé le fait que, bien que la simulation des modèles raffinés soit toujours efficace, leur vérification formelle peut souffrir de l'explosion combinatoire de l'espace des états. Des travaux futurs pourraient étudier les techniques de réduction de l'espace des états pour surmonter cette explosion.

L'application de notre approche à la modélisation d'un système biologique a également été montrée, démontrant son utilité pour la modélisation des systèmes informatiques en général. Le raffinement de la dimension temporelle dans les modèles stochastiques apporte une solution pour les modèles dans lesquels le temps et l'aléatoire sont deux caractéristiques importantes.

Chapitre **6**

La Simulation dans le Cadre Générique des Calculs de Processus †

De nombreux langages de programmation basés sur les calculs de processus ont été développés, notamment pour la modélisation en biologie, dont la plupart peuvent générer un nombre potentiellement non-borné d'espèces moléculaires et de réactions. Ainsi, la simulation de tels langages ne peut pas utiliser directement les techniques standards basées sur la simulation de réactions, et est généralement implémentée avec des algorithmes spécifiques de simulation stochastique. Dans ce chapitre, nous proposons une alternative reposant sur une machine abstraite générique qui peut être instanciée pour simuler un grand nombre de calculs de processus en utilisant diverses méthodes de simulation. La machine abstraite fonctionne comme un compilateur « à la volée », qui met à jour dynamiquement l'ensemble possible des réactions et choisit la prochaine réaction dans un processus itératif. Nous montrons l'instanciation de la machine abstraite générique pour deux méthodes de simulation markovienne et une non-markovienne ; ainsi que pour trois calculs de processus stochastiques représentatifs : le π-calcul, basé sur les agents ; le calcul bioambiant, basé sur les compartiments, et κ, basé sur les règles. Nous présentons une méthode générique pour prouver la correction de l'instanciation d'un calcul de processus arbitraire dans la machine abstraite. Enfin, nous montrons comment cette machine abstraite peut être utilisée pour simuler des modèles hétérogènes composés de modules écrits dans différents langages. Notre approche forme les bases d'un environnement générique de simulation markovienne et non-markovienne des modèles biologiques hétérogènes.

6.1 Préliminaires

Les systèmes biologiques mettent en jeu un grand nombre de composants engendrant des interactions complexes, fortement parallèles et possédant une stochasticité intrinsèque. Afin de reproduire cette complexité, de nombreux langages de programmation basés sur les calculs de processus ont été développés, dont notamment des variantes du π-calcul stochastique (Priami et coll., 2001; Regev, Silverman & Shapiro, 2001; Phillips & Cardelli, 2007), BlenX (Dematté, Priami & Romanel, 2008*a*), κ (Danos, Feret, Fontana, Harmer & Krivine, 2007), LBS (Pedersen & Plotkin, 2008) ; des vari-

†. Travail effectué en collaboration avec Andrew Phillips (Microsoft Research, Cambridge, Royaume-Uni), Matthew Lakin (Microsoft Research, Cambridge, Royaume-Uni), et Simon Youssef (Ludwig-Maximilians-Universität, Munich, Allemagne).

antes du calcul bioambiant (Regev, Panina, Silverman, Cardelli & Shapiro, 2004; Phillips, 2009) ; et DSD (Phillips & Cardelli, 2009). La plupart de ces calculs sont assez expressifs pour générer un nombre potentiellement infini d'espèces moléculaires et de réactions. De ce fait, ils ne peuvent pas se reposer sur les méthodes de simulation classiques basées sur les réactions telles que celles proposées par Gillespie (1977) et Gibson & Bruck (2000) qui demandent un nombre fixe d'espèces et de réactions. Ainsi, un algorithme de simulation spécifique au langage est généralement développé. Le choix de l'algorithme dépend de la nature du système biologique sous-jacent qui peut impliquer la nécessité d'une simulation stochastique exacte (Gillespie, 1977; Gibson & Bruck, 2000), la spécification de réactions opérant à plusieurs échelles de temps (Gillespie, 2001; Tian & Burrage, 2004), ou l'utilisation de réactions non-markoviennes (Gibson & Bruck, 2000; Bratsun, Volfson, Tsimring & Hasty, 2005).

Au lieu d'implémenter un algorithme de simulation stochastique dédié à chaque calcul de processus, nous proposons d'utiliser une machine abstraite générique qui peut encoder un grand nombre de calculs de processus et qui peut être instanciée avec plusieurs algorithmes de simulation stochastique basés sur les réactions. La machine abstraite procède « à la volée » en générant dynamiquement l'ensemble des réactions possibles et en choisissant alors la prochaine réaction à exécuter. Ainsi, la machine abstraite calcule seulement les espèces et les réactions qui sont nécessaires à la simulation. La machine abstraite est instanciée pour un calcul donné en définissant deux fonctions : une pour convertir un processus du calcul en un ensemble d'espèces ; et une pour calculer l'ensemble des réactions possibles entre les espèces. La machine abstraite est instanciée pour un algorithme de simulation donné en définissant trois fonctions : une sélectionnant la prochaine réaction à exécuter ; une calculant l'activité d'une réaction en fonction de la population des espèces ; et une mettant à jour cette activité en fonction de l'évolution de la population au cours du temps.

Bien que l'idée d'intégrer différents modèles et méthodes de simulations dans un socle commun n'est pas nouvelle (Ewald, Himmelspach, Jeschke, Leye & Uhrmacher, 2010), notre approche est la première qui tente de définir formellement un cadre générique de simulation pour un large panel de calculs de processus avec une méthode arbitraire de simulation basée sur les réactions. Assurer une séparation claire entre l'algorithme de simulation et le langage de spécification nous permet d'instancier simplement la machine pour des calculs de processus différents et d'ajouter de nouvelles fonctionnalités sur l'algorithme de simulation (comme la simulation non-markovienne) qui seront alors partagées pour les calculs instanciés. De plus, cette approche peut être utilisée pour simuler différents calculs de processus interagissant simultanément, fournissant un environnement de simulation biologique multi-langage.

Nous utilisons en premier lieu notre machine générique pour simuler une variante du π-calcul stochastique (Phillips & Cardelli, 2007) que nous étendons avec une primitive élémentaire de complexation en utilisant les écritures bornées. Nous instancions la machine abstraite de manière à stocker un complexe comme une seule espèce, ce qui permet une simulation efficace des systèmes impliquant des complexations. Nous montrons également comment le calcul bioambiant et κ peuvent être encodés dans notre machine abstraite. Chacun de ces calculs est instancié en définissant les fonctions appropriées à la traduction entre les espèces et les processus du calcul et au calcul de l'ensemble des réactions entre les espèces. Nous établissons une preuve générique de la correction de la simulation d'un calcul de processus arbitraire avec une méthode markovienne choisie. Cette approche peut être utilisée pour prouver de manière succincte l'instanciation d'un langage choisi dans la machine abstraite. Nous montrons l'instanciation de notre machine abstraite avec des méthodes classiques de simulation markovienne basées sur les réactions, dont une que nous étendons à la simulation non-markovienne. Ceci apporte naturellement une implémentation formelle de la simulation non-markovienne du π-calcul stochastique et du calcul bioambiant.

La simulation de calculs de processus spécialisés dans différents domaines (agents, compartiments, règles, etc.) est importante car elle permet le choix du langage le plus adapté à la modélisation du système d'intérêt. Cependant, si nous considérons de très grands systèmes biologiques complexes, l'utilisation d'un seul langage de modélisation peut représenter une limitation : les systèmes biologiques de grande échelle tendent à être composés de modules bien définis réalisant des fonctions très variées. Une approche naturelle de modélisation de tels systèmes est la construction de sous-modèles communicant entre eux, où chacun de ces sous-modèles représente une unité fonctionnelle particulière. Cette modélisation compositionnelle est une pratique très répandue apportant de nombreux avantages, dont une possibilité de ré-utilisation très simple. Notre machine abstraite est suffisamment générique pour gérer une telle flexiblité car elle permet la simulation de modèles récursivement composés de sous-modèles écrits dans différents langages, et ceci sans modification des définitions.

Ce chapitre est structuré de la manière suivante. Dans la section 6.2, nous définissons la machine abstraite générique. Dans la section 6.3, nous instancions la méthode de simulation de la machine abstraite à la *méthode directe* (ou *Direct Method*) (Gillespie, 1977) et à la *méthode par réaction suivante* (ou *Next Reaction Method*) (Gibson & Bruck, 2000). Dans la section 6.4, nous instancions la machine abstraite pour le π-calcul. Dans la section 6.5, nous présentons une méthode générique pour prouver la correction de la machine abstraite instanciée avec un calcul de processus arbitraire et une méthode de simulation markovienne. Nous présentons alors les preuves des instances spécifiques de notre machine dans la section 6.4. Dans la section 6.6, nous étendons la méthode par réaction suivante pour la simulation non-markovienne et donnons une méthode générique afin de montrer la correction de cette instanciation. Dans la section 6.7, nous montrons comment les modèles impliquant plusieurs calculs de processus différents peuvent être implémentés à l'aide de notre formalisme. L'instanciation et la correction de la machine pour le calcul bioambiant et κ sont présentées dans l'appendice A.

Hormis la sous-section 6.6.2 traitant la correction du cas non-markovien, les résultats présentés dans ce chapitre résultent de travaux effectués en collaboration avec Andrew Phillips et Matthew Lakin (Microsoft Research, Cambridge, UK) et Simon Youssef (Ludwig-Maximilians-Universität, Munich, Allemagne) ; dont une partie est publiée dans (Paulevé, Youssef, Lakin & Phillips, 2010) et (Phillips, Lakin & Paulevé, 2010).

6.2 La Machine Abstraite Générique

6.2.1 Syntaxe et Sémantique

La syntaxe de la machine générique abstraite est donnée par la définition 6.1. Un terme T de la machine est un triplet (t, S, R), où t est le temps courant, S est une association d'une espèce I vers sa population i, et R une association d'une réaction O vers son *activité* A, utilisée pour calculer la prochaine réaction à exécuter. La structure de l'activité A dépend du choix de l'algorithme de simulation. Chaque réaction est représentée par un tuple $(\overline{I}, r, \overline{I'})$, où $\overline{I}$ dénote le multi-ensemble des espèces réactantes, $\overline{I'}$ dénote le multi-ensemble des espèces produites, et r dénote le taux de la réaction. La syntaxe des espèces I est spécifique au choix du calcul de processus.

Définition 6.1 (Syntaxe de la machine générique abstraite)**.** Une machine est définie par un terme T comprenant le temps courant t, une association des espèces S et une association des réactions R. Étant donné $\overline{I}$ dénotant un multi-ensemble d'espèces $[I_1, .., I_N]$:

$T ::=$	(t, S, R)	Temps t ; association d'espèces S ; de réactions R
$S ::=$	$\{I_1 \mapsto i_1, .., I_N \mapsto i_N\}$	Association d'une espèce I à sa population i
$R ::=$	$\{O_1 \mapsto A_1, .., O_N \mapsto A_N\}$	Association d'une réaction O à son activité A
$O ::=$	$(\overline{I}, r, \overline{I'})$	Réaction $\overline{I} \xrightarrow{r} \overline{I'}$ avec un taux r

La structure d'un terme de la machine abstraite peut se résumer sous forme de tableau comme suit :

Terme T de la machine				
Temps t	Association d'espèces S		Association de réactions R	
	Espèce	Population	Réaction	Activité
	I_1	i_1	$\overline{I}_1 \xrightarrow{r_1} \overline{I'}_1$	A_1
	...	...	...	...
	I_N	i_N	$\overline{I}_M \xrightarrow{r_M} \overline{I'}_M$	A_M

Pour instancier la machine abstraite avec un calcul de processus donné, nous définissons simplement une fonction espèces(P) pour transformer un processus P du calcul en un multi-ensemble d'espèces, et une fonction réactions$(I, \widetilde{I'})$ pour calculer le multi-ensemble de réactions entre une nouvelle espèce I et un ensemble existant d'espèces $\widetilde{I'}$. La fonction espèces est utilisée pour initialiser la machine abstraite au début de la simulation, alors que la fonction réactions est utilisée pour mettre dynamiquement à jour l'ensemble des réactions possibles. *Ce développement technique permet la simulation de systèmes ayant un nombre potentiellement non-borné d'espèces et de réactions, ce qui n'est pas possible en utilisant les algorithmes de simulation stochastique standards.*

Pour instancier la machine abstraite avec une méthode de simulation donnée, nous définissons une fonction suite(T) pour choisir la prochaine réaction apparaissant dans un terme T, une fonction init$(\overline{O}, T)$ pour initialiser un terme T avec un multi-ensemble de réactions $\overline{O}$, et une fonction m-à-j(I, T) pour rafraîchir l'activité des réactions du terme T affectées par une espèce donnée I. La machine abstraite exécute la méthode de simulation donnée en appliquant itérativement la règle :

$$\frac{(\overline{I}, r, \overline{I'}), a, t' = \text{suite}(t, S, R)}{(t, S, R) \xrightarrow{a,(\overline{I},r,\overline{I'})} \overline{I'} \oplus ((t', S, R) \ominus \overline{I})} \tag{6.1}$$

Cette règle sélectionne une réaction en utilisant la fonction suite qui renvoie la réaction choisie, sa *propension* a et le nouveau temps de simulation t'. La réaction choisie est exécutée en supprimant les réactants $\overline{I}$, en ajoutant les produits $\overline{I'}$ et en remplaçant le temps courant de la simulation dans le terme de la machine.

L'ajout et la suppression des espèces d'un terme sont donnés dans la définition 6.2. Un processus P est ajouté à un terme T en calculant le multi-ensemble d'espèces $[I_1, .., I_N]$ correspondant à P puis en ajoutant chacune de ces espèces au terme. Si une nouvelle espèce I est déjà présente dans le terme, alors sa population est incrémentée dans S et l'activité des réactions concernées est mise à jour. Si l'espèce est nouvelle dans le terme, alors sa population est initialisée dans S et les nouvelles réactions impliquant ces espèces sont calculées, avec leur activité respective. L'opération $T \ominus \overline{I}$ supprime les espèces $\overline{I}$ du terme T en décrémentant les populations des espèces correspondantes, et en rafraîchissant l'activité des réactions concernées.

Définition 6.2 (Ajout et suppression des espèces dans la machine abstraite générique)**.** Nous dénotons par $\overline{O}$ un multi-ensemble de réactions $[O_1, .., O_N]$; Si $\overline{I}$ est le multi-ensemble $\{(I_1, i_1), .., (I_N, i_N)\}$ nous écrivons $\overline{I} \oplus T$ pour $(I_1, i_1) \oplus .. \oplus (I_N, i_N) \oplus T$, et $T \ominus \overline{I}$ pour $T \ominus (I_1, i_1) \ominus .. \ominus (I_N, i_N)$ (l'ordre est sans importance).

$$
\begin{array}{rcl}
P \oplus T & \triangleq & \text{espèces}(P) \oplus T \\
(I, i) \oplus (t, S, R) & \triangleq & (t, S', R\{R'\}) \quad \text{si } \widetilde{I'} = \text{dom}(S) \wedge I \notin \widetilde{I'} \wedge S' = S\{I \mapsto i\} \\
 & & \wedge \overline{O} = \text{réactions}(I, \widetilde{I'}) \wedge R' = \text{init}(\overline{O}, (t, S', R)) \\
(I, i) \oplus (t, S, R) & \triangleq & (t, S', R\{R'\}) \quad \text{si } S(I) = i' \wedge S' = S\{I \mapsto i' + i\} \\
 & & \wedge R' = \text{m-à-j}(I, (t, S', R)) \\
(t, S, R) \ominus (I, i) & \triangleq & (t, S', R\{R'\}) \quad \text{si } S(I) = i' \wedge S' = S\{I \mapsto i' - i\} \\
 & & \wedge R' = \text{m-à-j}(I, (t, S', R)) \wedge i' \geq i
\end{array}
$$

Pour les méthodes de simulation markovienne, pour lesquelles les délais r des réactions sont supposés distribués exponentiellement avec la forme $\exp(\lambda)$, nous pouvons calculer directement la Chaîne de Markov à Temps Continu (CMTC) de la machine abstraite à partir de l'équation (6.1). Dans un premier temps, nous dérivons une sémantique en CMTC d'une relation de réduction $T \xrightarrow{a,O} T'$ en utilisant la règle suivante :

$$
\frac{a = \left(\sum_{\{b,O \mid T \xrightarrow{b,O} T'\}} b\right) > 0}{T \xrightarrow{a} T'} \tag{6.2}
$$

Cette règle additionne les propensions b de toutes les réactions $T \xrightarrow{b,O} T'$ qui donnent le même terme T'. Nous dérivons ensuite une CMTC pour laquelle les transitions pour un terme T donné sont définies par l'ensemble $\{T \xrightarrow{a} T'\}$, pour chaque terme distinct T'.

6.3 Instanciation de la Machine Abstraite avec une Méthode de Simulation

Cette section décrit comment la machine abstraite peut être instanciée avec une méthode choisie de simulation en définissant les fonctions appropriées suite, init et m-à-j. Nous commençons par présenter une instanciation avec la *Méthode Directe* (ou *Direct Method*) (Gillespie, 1977), puis une instanciation avec la *Méthode par Réaction Suivante* (ou *Next Reaction Method*) (Gibson & Bruck, 2000).

6.3.1 La Méthode Directe de Gillespie

Une instanciation de la machine abstraite générique avec la Méthode Directe (Gillespie, 1977) est détaillée dans la définition 6.3. Chaque réaction O_i dans R est associée à son activité, qui, dans ce cas, est simplement la propension a de la réaction. La fonction $\text{propension}((\overline{I}, r, \overline{I'}), S)$ calcule la propension de la réaction $(\overline{I}, r, \overline{I'})$ en multipliant le nombre de combinaisons distinctes des réactants $\overline{I}$ en exploitant le taux exponentiel λ de la réaction, en supposant que tous les délais des réactions sont distribués exponentiellement selon $\exp(\lambda)$ (cas markovien). Le nombre de combinaisons distinctes des réactants est calculé avec le coefficient binomial à l'aide de la population de chaque réactant associée aux espèces dans S. La fonction $\text{init}(\overline{O}, T)$ calcule la propension de chaque réaction dans

le multi-ensemble $\overline{O}$ avec les populations initiales des espèces dans T. Pour fusionner des copies multiples d'une même réaction, le taux est multiplié par le nombre d'occurrences de la réaction. Il est à noter que cette fusion est uniquement appliquable dans le cas d'une simulation markovienne. La fonction m-à-j(I, T) recalcule les propensions de toutes les réactions dans T dont I est un réactant. Enfin, la fonction suite(T) sélectionne une réaction O_μ de T avec une probabilité proportionnelle à la propension de la réaction a_μ, et calcule le délai t' de la réaction en accord avec (Gillespie, 1977).

Définition 6.3 (Instanciation de la machine abstraite avec la Méthode Directe)**.**

$$
\begin{aligned}
\mathsf{suite}(t, S, R) &\triangleq (O_\mu, a_\mu, t + t') \quad \text{si } a_0 = \textstyle\sum_{O_i \in \mathsf{dom}(R)} R(O_i) > 0, \\
&\quad t' = (\tfrac{1}{a_0}) \ln(\tfrac{1}{n_1}) \text{ et } \textstyle\sum_{i=1}^{\mu-1} a_i < n_2 a_0 \leq \sum_{i=1}^{\mu} a_i \\
&\quad \text{où } n_1 \text{ et } n_2 \text{ sont deux nombres aléatoires suivant la} \\
&\quad \text{distribution uniforme } U(0,1). \\
\mathsf{init}(\overline{O}, (t, S, R)) &\triangleq \{O_i \mapsto \mathsf{propension}(O_i, S) \mid O_i \in \mathsf{fusion}(\overline{O})\} \\
\mathsf{fusion}(\overline{O}) &\triangleq \{(\overline{I}, \exp(\lambda \times o_i), \overline{I'}) \mid ((\overline{I}, \exp(\lambda), \overline{I'}), o_i) \in \overline{O}\} \\
\mathsf{m\text{-}à\text{-}j}(I, (t, S, R)) &\triangleq \{O_i \mapsto \mathsf{propension}(O_i, S) \mid O_i \in \mathsf{dom}(R) \\
&\quad \wedge O_i = (\overline{J}, r, \overline{J'}) \wedge I \in \overline{J}\}
\end{aligned}
$$

$$
\mathsf{propension}(\{(I_1, i_1), ..(I_N, i_N)\}, \exp(\lambda), \overline{I'}), S) \triangleq \lambda \times \tbinom{S(I_1)}{i_1} \times .. \times \tbinom{S(I_N)}{i_N}
$$

La notation $\binom{n}{k}$ dénote le coefficient binomial calculant le nombre de sous-ensembles distincts de taille k d'un ensemble de taille n. Les délais r des réactions sont supposés distribués exponentiellement selon la forme $\exp(\lambda)$, où λ est un nombre réel.

6.3.2 La Méthode par Réaction Suivante

Une instance de la machine abstraite générique avec la Méthode par Réaction Suivante (MRS) de (Gibson & Bruck, 2000) est détaillée dans la définition 6.4. Chaque réaction O_i de R est associée à son activité, ici une paire (a, t), où a est la propension de la réaction et t le temps présumé de l'application de la réaction. Les fonctions propension et fusion sont celles définies dans la définition 6.3. La réaction suivante est celle associée au temps présumé le plus petit, comme défini par la fonction suite(T) qui renvoie la réaction choisie $(\overline{J}, r, \overline{J'})$ avec sa propension a et son temps présumé t' (équation (6.3)).

Quant une nouvelle réaction est créée, MRS calcule son temps présumé en accord avec sa propension et sa distribution de probabilité (équation (6.4)). MRS fournit également un moyen pour mettre à jour les temps présumés des réactions markoviennes quand leur propension change, sans générer une nouvelle variable aléatoire, mais en redimensionnant le temps présumé (équation (6.5). Si l'ancienne propension est 0, ce redimensionnement ne peut pas s'appliquer directement (il faudrait alors recalculer complètement le temps présumé). Toutefois, ce cas peut être géré en enregistrant une variable supplémentaire correspondant à la dernière propension non-nulle, et utiliser cette variable pour effectuer le redimensionnement (comme discuté dans la note 11 de (Gibson & Bruck, 2000)). Enfin, si la nouvelle propension est 0, le temps présumé est mis à l'infini.

Définition 6.4 (Instanciation de la machine abstraite avec la Méthode par Réaction Suivante)**.** L'activité de chaque réaction O est une paire $A = (a, t)$, où a dénote la propension de la réaction et t dénote le temps présumé de l'application de la réaction. La fonction $\mathsf{délai}(a)$ calcule la durée d'une réaction markovienne avec une propension a.

$$\mathsf{suite}(t, S, R) \triangleq (O, a, t') \quad \text{si } R(O) = (a, t') \wedge a > 0 \wedge t' = \min\{t \mid R(O) = (a, t)\} \tag{6.3}$$

$$\mathsf{init}(\overline{O}, (t, S, R)) \triangleq \{O_i \mapsto (t', a) \mid O_i \in \mathsf{fusion}(\overline{O}) \wedge a = \mathsf{propension}(O_i, S) \wedge t' = t + \mathsf{délai}(a)\} \tag{6.4}$$

$$\mathsf{m\text{-}à\text{-}j}(I, (t, S, R)) \triangleq \{O \mapsto (t', a') \mid R(O) = (t'', a) \wedge O = (\overline{J}, r, \overline{J'}) \wedge I \in \overline{J} \wedge a' = \mathsf{propension}(O, S) \wedge t' = t + (a/a')(t'' - t)\} \tag{6.5}$$

6.4 Instanciation de la Machine Abstraite Générique avec le π-Calcul Stochastique

Dans cette section, nous présentons une instanciation de la machine abstraite générique avec une variante du π-calcul stochastique en définissant les fonctions appropriées espèces et réactions. Cette instanciation inclut également une optimisation pour la simulation de complexes de processus.

La sémantique du calcul est directement utilisée pour dériver la fonction réactions. Une fonction processus est également définie pour chaque calcul afin de retraduire une espèce en un processus. Cette fonction sera utilisée pour prouver la correction de l'instanciation dans la section 6.5. En général, l'instanciation de la machine pour un calcul donné peut être effectuée de différentes manières, en permettant l'incorporation d'optimisations spécifiques au calcul choisi.

Des instanciations de la machine générique avec le calcul bioambiant et κ sont données dans l'appendice A.

6.4.1 Syntaxe et sémantique du calcul

La syntaxe de la variante du π-calcul stochastique utilisée dans ce chapitre est donnée dans la définition 6.5 et est basée sur (Phillips & Cardelli, 2007). Il est à noter que cette définition est différente de celle utilisée dans le chapitre 5 (définition 5.1 page 62). Un processus P peut être un choix parmi des actions C, une instance $X(\tilde{n})$ d'une définition X avec les paramètres $\tilde{n}$, une composition de processus parallèles $P \mid Q$, ou un processus $\nu x\, P$ avec un canal privé x. Un choix C consiste en une compétition entre zéro ou plus actions $\pi^i.P$, où π est l'action à effectuer avant d'exécuter le processus P, où i est un indice identifiant l'action. Nous utilisons 0 comme indice par défaut, et nous abrégeons π^0 en π. Une action π peut être un délai τ_r avec un taux r, un envoi $!x(\tilde{n})$ des valeurs sur un canal x, un envoi $!x(\nu\tilde{n}\,)$ de valeurs privées $\tilde{n}$ sur un canal x, ou la réception $?x(\tilde{m})$ de valeurs $\tilde{m}$ sur un canal x. Un environnement E consiste en un ensemble de définitions $X(\tilde{m}) \mapsto P$, où X est le nom de la définition, $\tilde{m}$ ses paramètres, et P le processus correspondant.

Les axiomes de congruence structurelle pour le π-calcul stochastique sont résumés de manière standard dans la définition 6.6, et les règles de réduction sont récapitulées dans la définition 6.7. La notation $P \xrightarrow{r,w} P'$ établit que le processus P peut devenir P' en exécutant une réaction w avec

Définition 6.5 (Syntaxe du π-calcul stochastique)**.**

$P ::=$	C	Choix	$\pi ::=$	τ_r	Délai
$\mid$	$X(\tilde{n})$	Instance	$\mid$	$!x(\tilde{n})$	Envoi
$\mid$	$P_1 \mid P_2$	Parallèle	$\mid$	$!x(\nu\tilde{m})$	Envoi borné
$\mid$	$\nu x\, P$	Restriction	$\mid$	$?x(\tilde{m})$	Réception
$C ::=$	$\pi_1^{i_1}.P_1 + .. + \pi_N^{i_N}.P_N$	Actions			
$E ::=$	$X_1(\tilde{m}_1) \mapsto P_1, .., X_N(\tilde{m}_N) \mapsto P_N$	Environnement			

Le choix vide dénote le processus nul $\mathbf{0}$. Pour chaque définition $X(\tilde{m}) \mapsto P$ dans l'environnement, nous supposons $\tilde{m} \subseteq \text{fn}(P)$, où $\text{fn}(P)$ dénote les noms libres de P. La restriction $\nu x\, P$ lie le nom x à P et à la fois $!x(\nu\tilde{m}).P$ et $?x(\tilde{m}).P$ lient les nom $\tilde{m}$ à P. Nous supposons également que tous les appels récursifs à une définition sont *gardés* à l'intérieur du préfixe π d'une action de telle sorte que pour une définition $X(\tilde{m}) \mapsto P$ donnée, tout appel récursif à X dans P ne peut s'effectuer qu'après une action π. Ceci prévient une expansion infinie des définitions des processus.

Définition 6.6 (Axiomes de congruence structurelle pour le π-calcul stochastique)**.** La congruence structurelle est réflexive, symétrique et transitive et s'applique dans tout contexte à l'intérieur d'un processus ou d'un choix. Les processus sont supposés égaux au renommage des noms liés et réordonnancement des termes dans un choix près. Nous supposons un environnement global E.

$$P \mid \mathbf{0} \equiv P \tag{6.6}$$
$$P_1 \mid P_2 \equiv P_2 \mid P_1 \tag{6.7}$$
$$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 \tag{6.8}$$
$$\nu x\, \mathbf{0} \equiv \mathbf{0} \tag{6.9}$$
$$\nu x\, \nu y\, P \equiv \nu y\, \nu x\, P \tag{6.10}$$
$$\nu x\, (P_1 \mid P_2) \equiv P_1 \mid \nu x\, P_2 \quad \text{si } x \notin \text{fn}(P_1) \tag{6.11}$$
$$X(\tilde{n}) \equiv P_{\{\tilde{n}/\tilde{m}\}} \quad \text{si } E(X(\tilde{m})) = P \tag{6.12}$$

Définition 6.7 (Réduction en π-calcul stochastique)**.**

$$\tau_r^i.P + C \xrightarrow{r,i} P \tag{6.13}$$
$$!x(\tilde{n})^{i_1}.P_1 + C_1 \mid ?x(\tilde{m})^{i_2}.P_2 + C_2 \xrightarrow{\text{taux}(x),(i_1,i_2)} P_1 \mid P_{2\{\tilde{n}/\tilde{m}\}} \tag{6.14}$$
$$!x(\nu\tilde{n})^{i_1}.P_1 + C_1 \mid ?x(\tilde{m})^{i_2}.P_2 + C_2 \xrightarrow{\text{taux}(x),(i_1,i_2)} \nu\tilde{n}(P_1 \mid P_{2\{\tilde{n}/\tilde{m}\}}) \tag{6.15}$$
$$P \xrightarrow{r,w} P' \Rightarrow \nu x\, P \xrightarrow{r,w} \nu x\, P' \tag{6.16}$$
$$P \xrightarrow{r,w} P' \Rightarrow P \mid Q \xrightarrow{r,w} P' \mid Q \tag{6.17}$$
$$Q \equiv P \xrightarrow{r,w} P' \equiv Q' \Rightarrow Q \xrightarrow{r,w} Q' \tag{6.18}$$

où un identifiant de réaction w est soit un simple indice i dénotant un délai, ou une paire d'indices (i_1, i_2) dénotant une interaction.

un taux r. L'identifiant de la réaction w peut être un indice i dénotant un délai τ_r^i, ou une paire d'indices (i_1, i_2) dénotant une interaction entre deux actions avec les indices i_1 et i_2, respectivement. Un processus peut évoluer de lui-même en exécutant un délai τ_r. Deux processus peuvent évoluer simultanément en communicant ou en se liant entre eux. Une communication entre deux processus est effectuée quand un processus envoie des valeurs $\tilde{n}$ sur un canal x, dénoté par $!x(\tilde{n})$, et un processus parallèle reçoit ces valeurs sur le même canal x, dénoté par $?x(\tilde{m})$. Une liaison entre deux processus est effectuée quand un processus envoie des valeurs privées $\tilde{n}$ sur un canal x, dénoté par $!x(\nu\tilde{n})$, qui sont alors partagées entre l'émetteur et le récepteur, représentant la formation d'un complexe entre ces deux processus.

Le comportement stochastique est introduit dans le calcul en associant à chaque délai τ_r un taux r et à chaque canal x un taux $\mathsf{taux}(x)$, où chaque taux caractérise une probabilité de distribution. Pour les taux distribués exponentiellement par la forme $\exp(\lambda)$, la probabilité qu'une réaction soit effective dans les t unités de temps est donnée par $F(t) = 1 - e^{-\lambda t}$. La durée moyenne d'une réaction est alors donnée par $1/\lambda$.

Nous dérivons une sémantique en CMTC pour le π-calcul stochastique directement de sa relation de réduction, en définissant au préalable une *forme indicée* des processus comme suit, basée sur (Phillips, 2009).

Définition 6.8 (Forme indicée d'un processus)**.** Un processus P est dans une *forme indicée* si il est de la forme $\nu x_1 \,..\nu x_M\,(C_1 \mid .. \mid C_N)$ où chaque action non-gardée π^i est associé à un unique indice i.

Nous pouvons montrer que tout processus est structurellement congruent à un processus en forme indicée, à un renommage des indices près (la preuve est directe). Nous notons que le seul rôle des indices est de permettre d'identifier de manière unique les actions individuelles, et que renommer ces indices n'a aucun effet sur les réductions qu'un processus peut exécuter (voir (Phillips, 2009) pour plus de détails). La sémantique en CMTC est donnée par la règle suivante, où le processus P est dans une forme indicée.

$$\frac{a = \left(\sum_{\{\lambda, w \mid P \xrightarrow{\exp(\lambda), w} P'\}} \lambda\right) > 0}{P \xrightarrow{a} P'} \tag{6.19}$$

La positivité stricte de a assure que nous pouvons dériver $P \xrightarrow{a} P'$ précisément quand il existe une réduction de P vers P' dans le π-calcul stochastique. Nous nous basons implicitement sur le fait que $\lambda > 0$ pour les taux exponentiels $\exp(\lambda)$. Nous dérivons une CMTC de telle sorte que les transitions partant d'un processus P sont données par l'ensemble $\{P \xrightarrow{a} P'\}$ pour chaque processus P' distinct, c.-à-d. qui n'est pas structurellement congruent.

6.4.2 Extraire les espèces et les réactions des processus

La première étape pour instancier la machine abstraite générique avec le π-calcul stochastique est la spécification d'une espèce. Nous supposons ici qu'une espèce est soit une instance $X(\tilde{n})$, soit un complexe d'instances $\nu\tilde{n}\,(X_1(\tilde{n}_1) \mid \ldots \mid X_M(\tilde{n}_M))$, où chaque instance correspond à un choix entre des actions. Notre approche est motivée par l'observation que le choix d'une action est l'unité basique des calculs, où deux choix parallèles interagissent en communicant au travers de canaux partagés. Une approche alternative pourrait supposer que les espèces correspondent directement

à un choix d'actions, au lieu d'utiliser une instance nommée $X(\tilde{n})$. Notre décision d'utiliser une instance nommée à l'avantage qu'une espèce peut être identifiée explicitement dans un modèle biologique par un nom informatif. Afin de formaliser la notion d'espèce en π-calcul stochastique, nous définissons la forme normale des processus (définition 6.9), et montrons que tout processus est structurellement congruent à une forme normale (proposition 6.1).

Proposition 6.1. *Tous les processus du π-calcul stochastique sont congruents à une forme normale donnée par la définition 6.9.*

Démonstration. Par induction sur la définition 6.10. En utilisant les congruences structurelles de la définition 6.6, nous augmentons l'environnement de telle sorte que tous les choix sont définis séparément (équation (6.12)), et nous remplaçons les instances qui ne sont pas un choix par leur définition correspondante (équation (6.12)). En utilisant la congruence structurelle dans le cadre de l'équation (6.11), nous modifions la restriction de telle sorte qu'un processus soit une composition parallèle d'espèces, où chaque espèce est soit une instance, soit un complexe. □

En utilisant notre forme normale pour les processus (que nous pouvons construire à l'aide de la définition 6.10), nous définissons les fonctions nécessaires à l'instanciation de la machine abstraite générique avec le π-calcul stochastique (définition 6.11). La fonction $\mathsf{espèces}(P)$ convertit un processus en un multi-ensemble d'espèces, la fonction $\mathsf{processus}(\overline{I})$ convertit un multi-ensemble d'espèces en un processus, et la fonction $\mathsf{réactions}(I, \widetilde{J})$ calcule le multi-ensemble de réactions de l'espèce I avec l'ensemble des espèces $\widetilde{J}$. Nous notons que la fonction $\mathsf{réactions}$ renvoie un multi-ensemble et non un ensemble car une même réaction peut potentiellement être générée de façon multiple, comme dans le processus $X \mapsto \tau_r.Y + \tau_r.Y$. La fonction renvoie le multi-ensemble des réactions unaires (délais) combiné avec le multi-ensemble des réactions binaires (communications et liaisons).

6.4.3 Exemple

Nous illustrons l'application de la machine abstraite générique au π-calcul stochastique avec l'exemple suivant de formation de complexe :

$$A = !x(\nu u).AB(u) \qquad AB(u) = !u.A$$
$$B = ?x(u).BA(u) \qquad BA(u) = ?u.B$$

Initialement, cent copies des processus A et B sont ajoutées au terme de la machine vide, écrit $(100 \cdot A \mid 100 \cdot B) \oplus (0, \emptyset, \emptyset)$, où la notation $100 \cdot X$ représente cent copies parallèles du processus X. Ceci donne le terme de la machine $(0, S, R)$, où S et R sont comme suit :

$$S = \{A \mapsto 100, B \mapsto 100\}$$
$$R = [(\{A, B\}, \mathsf{taux}(x), \{\nu u\,(AB(u) \mid BA(u))\}) \mapsto (10^4 \cdot \lambda(\mathsf{taux}(x)), t_1)]$$

La réaction impliquant les espèces A et B est exécutée au temps t_1, après qu'une copie des espèces A et B ait été supprimée et qu'une copie du complexe ait été ajoutée au terme de la machine :

$$\nu u\,(AB(u) \mid BA(u)) \oplus ((t_1, S, R) \ominus \{A, B\})$$

Définition 6.9 (Forme normale des processus en π-calcul stochastique)**.** Un processus P est dans une forme normale si il est constitué d'une composition parallèle d'espèces I, où une espèce peut être une instance $X(\tilde{n})$ ou un complexe d'instances $\nu\tilde{z}\,(X_1(\tilde{n}_1) \mid \ldots \mid X_M(\tilde{n}_M))$, et où chaque instance $X(\tilde{n})$ correspond à un choix d'actions. Nous supposons que $\tilde{z} \cap \tilde{n}_1 \cap \ldots \cap \tilde{n}_M \neq \emptyset$ et $\tilde{z} \subseteq \tilde{n}_1 \cup \ldots \cup \tilde{n}_M$ afin de minimiser la portée des noms restreints.

$$
\begin{array}{rcll}
P ::= & I_1 \mid .. \mid I_N & & \text{Espèces} \\
I ::= & X(\tilde{n}) & & \text{Instance} \\
\mid & \nu\tilde{z}\,(X_1(\tilde{n}_1) \mid .. \mid X_M(\tilde{n}_M)) & & \text{Complexe} \\
C ::= & \pi_1^{i_1}.P_1 + .. + \pi_N^{i_N}.P_N & & \text{Choix} \\
E ::= & X_1(\tilde{m}_1) \mapsto C_1, .., X_N(\tilde{m}_N) \mapsto C_N & & \text{Environnement}
\end{array}
$$

avec $N \geq 0$ et $M \geq 1$.

Définition 6.10 (Calcul de la forme normale des processus en π-calcul stochastique)**.** Nous écrivons $\prod_i P_i$ pour $P_1 \mid .. \mid P_N$, où $i \in \{1, .., N\}$. Nous écrivons $E(X(\tilde{n})) = C$ pour $E(X(\tilde{m})) = C'$ où $C = C'_{\{\tilde{n}/\tilde{m}\}}$. Nous notons que le cas $\mathsf{normale}(C)$ nécessite que l'environnement contienne une unique définition telle que $E(X(\tilde{n})) = C$.

$$
\begin{array}{rcl}
\mathsf{normale}(\mathbf{0}) & \triangleq & \mathbf{0} \\
\mathsf{normale}(X(\tilde{n})) & \triangleq & X(\tilde{n}) \quad \text{si } E(X(\tilde{n})) = C \\
\mathsf{normale}(X(\tilde{n})) & \triangleq & \mathsf{normale}(P) \quad \text{si } E(X(\tilde{n})) = P \neq C \\
\mathsf{normale}(P_1 \mid P_2) & \triangleq & \mathsf{normale}(P_1) \mid \mathsf{normale}(P_2) \\
\mathsf{normale}(C) & \triangleq & X(\tilde{n}) \quad \text{si } E(X(\tilde{n})) = C \\
\mathsf{normale}(\nu x\, P) & \triangleq & \mathsf{insérer}(x, \mathsf{normale}(P)) \\
\mathsf{insérer}(x, \prod_i I_i) & \triangleq & (\nu\tilde{z} \prod_k K_k) \mid \prod_j I_j \quad \text{si } I_k = \nu\tilde{z}_k\, K_k \\
 & & \wedge x \in \mathrm{fn}(I_k), x \notin \mathrm{fn}(I_j) \\
 & & \wedge \bigcap \tilde{z}_k = \emptyset, \tilde{z} = \{x\} \cup \bigcup \tilde{z}_k \\
 & & \wedge i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{K} \wedge \mathcal{J} \cap \mathcal{K} = \emptyset, \mathcal{I} = \mathcal{J} \cup \mathcal{K}
\end{array}
$$

Définition 6.11 (Machine abstraite générique instanciée avec le π-calcul stochastique)**.** En utilisant la définition des réductions du calcul pour dériver les réactions. Nous supposons un environnement global E contenant toutes les définitions des instances. Nous écrivons $i \in I$ si l'identifiant i est présent dans l'espèce I.

$$
\begin{array}{rcl}
\mathsf{espèces}(P) & \triangleq & [I_1, .., I_N] \quad \text{si } \mathsf{normale}(P) = (I_1 \mid .. \mid I_N) \\
\mathsf{processus}([I_1, .., I_N]) & \triangleq & (I_1 \mid .. \mid I_N) \\
\mathsf{réactions}(I, \widetilde{J}) & \triangleq & \mathsf{unaire}(I) \uplus \mathsf{binaire}(I, (\widetilde{J} \cup \{I\})) \\
\mathsf{unaire}(I) & \triangleq & [([I], r, \mathsf{espèces}(P)) \mid I \xrightarrow{r,i} P] \\
\mathsf{binaire}(I_1, \widetilde{J}) & \triangleq & [([I_1, I_2], r, \mathsf{espèces}(P)) \mid I_2 \in \widetilde{J} \wedge i_1 \in I_1 \wedge i_2 \in I_2 \\
 & & \wedge (I_1 \mid I_2) \xrightarrow{r,(i_1,i_2)} P]
\end{array}
$$

Ceci donne le terme de machine (t_1, S_1, R_1), où $\mathsf{actions}(\nu u(AB(u) \mid BA(u))) = \tau_u.(A \mid B)$ et S_1 et R_1 sont comme suit :

$$S_1 = \{A \mapsto 99, B \mapsto 99, \nu u\,(AB(u) \mid BA(u)) \mapsto 1\}$$
$$R_1 = [(\{A, B\}, \mathsf{taux}(x), \{\nu u\,(AB(u) \mid BA(u))\}) \mapsto (9801 \cdot \lambda(\mathsf{taux}(x)), t_3),$$
$$(\{\nu u\,(AB(u) \mid BA(u))\}, \mathsf{taux}(u), \{A, B\}) \mapsto (\lambda(\mathsf{taux}(u)), t_2)]$$

Nous notons que les algorithmes de simulation existants tels que (Phillips & Cardelli, 2007) gèrent N copies du complexe $\nu u(AB(u) \mid BA(u))$ en créant un nom unique pour chaque canal restreint u, comme suit :

$$\nu u_1 \ldots \nu u_N\,(AB(u_1) \mid BA(u_1) \mid \ldots \mid AB(u_N) \mid BA(u_N))$$

Au contraire, notre approche traite ces N instances comme N copies du même complexe $\nu u(AB(u) \mid BA(u))$, donnant moins d'espèces, moins de réactions et donc une simulation plus efficace.

6.5 Correction

Dans cette section nous prouvons la correction de la machine abstraite générique pour des calculs de processus avec une sémantique markovienne et une méthode de simulation exacte basée sur la loi d'action de masse. La preuve est donnée en termes d'équivalence entre les CMTC sous-jacentes. Il est suffisant de montrer que la CMTC générée par la sémantique du calcul est la même que celle engendrée par la machine. Les algorithmes de simulation exacte avec les cinétiques suivant la loi d'action de masse ayant déjà été montrés équivalents à l'équation stochastique maîtresse sous-jacente, il est suffisant de montrer la correspondance avec les CMTC engendrée par une méthode de simulation donnée, ici la Méthode Directe.

6.5.1 Théorèmes Génériques de Correction

Nous définissons une fonction $(\!|P|\!)_t$ qui encode un processus P du calcul en un terme correspondant de la machine abstraite à un temps de simulation donné t. Nous définissons également une fonction $[\![T]\!]$ qui décode un terme T de la machine abstraite en un processus du calcul correspondant. Ces fonctions d'encodage et de décodage entre le calcul C et la machine CM sont établies dans la définition 6.12. La définition du décodage requiert la définition additionnelle d'une fonction processus pour le calcul instancié afin de traduire un multi-ensemble d'espèces en un processus. Ceci peut être considéré comme l'inverse de la fonction espèces.

Définition 6.12 (Traduction entre un processus P du calcul et un terme de machine T)**.**

$$(\!|P|\!)_t \triangleq P \oplus (t, \emptyset, \emptyset)$$
$$[\![t, S, R]\!] \triangleq \mathsf{processus}(S)$$

Pour une association d'espèces $S = \{I_1 \mapsto i_1, .., I_N \mapsto i_N\}$ donnée, nous considérerons également S comme le multi-ensemble d'espèces $\{(I_1, i_1), .., (I_N, i_N)\}$.

La correction de l'encodage est établie en démontrant l'équivalence de réduction entre le calcul et la machine. Afin de préserver la correspondance, nous supposons une notion de congruence

structurelle pour les termes de la machine, où les termes sont structurellement congruents au renommage des définitions près, au nettoyage des définitions inutilisées près et à la congruence structurelle des processus près. Nous supposons également que la congruence structurelle sur les termes de la machine autorise la présence d'espèces additionnelles avec une population nulle, ainsi que des réactions additionnelles avec une propension nulle. De plus, nous supposons la présence d'un environnement de définition E. Comme discuté précédemment, il suffit de prouver la correction avec la Méthode Directe (définition 6.3) de simulation stochastique, la Méthode Directe et la Méthode par Réaction Suivante sont équivalentes et connues pour être correctes. La définition 6.13 établit des fonctions utilisées par les preuves à venir.

Définition 6.13 (Fonctions additionnelles utilisées par les preuves)**.**

$$\begin{aligned} \text{ens-réactions}(S, S') &\triangleq \{(\overline{l}, r, \overline{l'}) \mid (\overline{l}, r, \overline{l'}) \in \text{ens-réactions}(S) \wedge S' = (S \uplus \overline{l}) \uplus \overline{l'}\} \\ \text{ens-réactions}(S) &\triangleq \text{fusion}([O_i \mid O_i \in \text{réactions}(I, \widetilde{l'}) \wedge \widetilde{S} = \{I\} \cup \widetilde{l'}]) \end{aligned}$$

La fonction fusion qui combine les réactions identiques multiples est définie dans la définition 6.3.

Dans cette section, nous présentons une technique générique pour prouver la correction d'un calcul C équipé des fonctions espèces, réactions et processus induisant l'instanciation d'une machine abstraite CM. Pour pouvoir exploiter nos preuves génériques, les propositions suivantes doivent être démontrées au préalable :

Proposition 6.2 (Correction des espèces)**.** $\forall P, \overline{l}$ *:* $\text{processus}(\text{espèces}(P)) = P$ *et* $\text{espèces}(\text{processus}(\overline{l})) = \overline{l}$.

Proposition 6.3 (Correction des réactions)**.** $\forall S, S', a$ *:* $\text{processus}(S) \xrightarrow{a} \text{processus}(S')$ *si et seulement si* $a = \sum_{\{O \in \text{ens-réactions}(S,S')\}} \text{propension}(O, S)$ *et* $a > 0$.

En supposant que ces propositions sont correctes, nous présentons maintenant les preuves génériques de validité et de complétude pour l'encodage du calcul en se basant sur la sémantique en CMTC de la machine abstraite (équation (6.2)) et la sémantique en CMTC du calcul. Comme mentionné dans la définition 6.12, nous écrivons S pour la population des espèces mais également pour le multi-ensemble d'espèces correspondant. Le terme $T = (t, S, R)$ d'une machine abstraite est dit *bien formé*, noté $\text{bien-formé}(T)$ si $\text{dom}(R) = \text{ens-réactions}(S)$. La congruence structurelle décrite plus haut nous autorise à ignorer toute réaction O pour laquelle $\text{propension}(O, S) = 0$. Il est simple de montrer que $(\!|P|\!)_t$ est bien formé pour tout processus P et que cette propriété est conservée par les réductions de la machine abstraite. Les théorèmes génériques de correction sont comme suit.

Théorème 6.1 (Validité générique)**.** $\forall T, T', a$ *: si* $T \in CM$ *et* $\text{bien-formé}(T)$ *et* $T \xrightarrow{a} T'$ *alors* $[\![T]\!] \xrightarrow{a} [\![T']\!]$.

Démonstration. Admettons que $T \xrightarrow{a} T'$ avec $T = (t, S, R)$ et $T' = (t', S', R')$. Par définition de la réduction de la CMTC de la machine abstraite (équation (6.2)), nous avons $a = \sum_{\{b,O \mid T \xrightarrow{b,O} T'\}} b$ avec $a > 0$.

Si $T \xrightarrow{b,O} T'$ avec $O = (\overline{l}, r, \overline{l'})$, alors par définition de la réduction de la machine abstraite (équation (6.1)), nous avons $T' = \overline{l'} \oplus ((t', S, R) \ominus \overline{l})$ pour un certain t'. Par définition de l'addition ($\oplus$) et

de la suppression ($\ominus$) des espèces, nous avons $S' = (S \uplus \overline{I}) \uplus \overline{I'}$. Par définition de la fonction suite de la Méthode Directe (définition 6.3), nous avons également $O \in \mathsf{dom}(R)$ et $b = \mathsf{propension}(O, S) > 0$. T étant bien formé, par définition de bien-formé nous obtenons $\mathsf{dom}(R) = \mathsf{ens\text{-}réactions}(S)$ et ainsi $O \in \mathsf{ens\text{-}réactions}(S)$. Car $S' = (S \uplus \overline{I}) \uplus \overline{I'}$, par définition de $\mathsf{ens\text{-}réactions}(S, S')$, nous avons $O \in \mathsf{ens\text{-}réactions}(S, S')$. Ainsi, $a = \sum_{\{O \in \mathsf{ens\text{-}réactions}(S,S')\}} \mathsf{propension}(O, S)$ avec $a > 0$. Donc, par la proposition 6.3 nous obtenons $\mathsf{processus}(S) \xrightarrow{a} \mathsf{processus}(S')$.

Par définition de $[\![\cdot]\!]$ nous avons $[\![T]\!] = [\![(t, S, R)]\!] = \mathsf{processus}(S)$ et $[\![T']\!] = [\![(t', S', R')]\!] = \mathsf{processus}(S')$. Ainsi, $[\![T]\!] \xrightarrow{a} [\![T']\!]$ est vérifié. □

Théorème 6.2 (Complétude générique)**.** *$\forall P, P', a$: si $P \in C$ et $P \xrightarrow{a} P'$, alors $(\!|P|\!)_t \xrightarrow{a} (\!|P'|\!)_{t'}$ pour t donné et un certain t'.*

Démonstration. Admettons que $P \xrightarrow{a} P'$ avec $S = \mathsf{espèces}(P)$ et $S' = \mathsf{espèces}(P')$. Par la proposition 6.2, nous avons $\mathsf{processus}(S) \xrightarrow{a} \mathsf{processus}(S')$. Ainsi, par la proposition 6.3, nous avons $a = \sum_{\{O \in \mathsf{ens\text{-}réactions}(S,S')\}} \mathsf{propension}(O, S)$.

Soit $(\!|P|\!)_t = (t, S, R) = T$ et $(\!|P'|\!)_{t'} = (t', S', R') = T'$. Par définition de $(\!|\cdot|\!)_t$, de la fonction d'addition ($\oplus$) et de la fonction init de la Méthode Directe (définition 6.3), nous avons $\mathsf{dom}(R) = \mathsf{ens\text{-}réactions}(S)$ et $R(O) = \mathsf{propension}(O, S)$ pour chaque réaction O de $\mathsf{dom}(R)$. Similairement, nous avons $\mathsf{dom}(R') = \mathsf{ens\text{-}réactions}(S')$ et $R'(O') = \mathsf{propension}(O', S')$ pour chaque réaction O' de $\mathsf{dom}(R')$.

Si $O \in \mathsf{ens\text{-}réactions}(S, S')$ avec $O = (\overline{I}, r, \overline{I'})$, alors par définition de ens-réactions nous avons $S' = (S \uplus \overline{I}) \uplus \overline{I'}$. Ainsi, par définition de l'addition ($\oplus$) et suppression ($\ominus$) des espèces, nous obtenons $T' = \overline{I'} \oplus ((t', S, R) \ominus \overline{I})$. Donc, par définition de la réduction de la machine abstraite (équation (6.1)), nous avons $T \xrightarrow{b,O} T'$ avec $b = \mathsf{propension}(O, S)$. Ainsi, $a = \sum_{\{b,O \mid T \xrightarrow{b,O} T'\}} b$ avec $a > 0$. Donc, par la réduction de la CMTC de la machine abstraite (équation (6.2)), nous avons $T \xrightarrow{a} T'$. Donc $(\!|P|\!)_t \xrightarrow{a} (\!|P|\!)_{t'}$ est vérifié pour t donné et un certain t'. □

Nous prouvons maintenant la correction de l'instanciation avec le π-calcul stochastique en montrant les propositions 6.2 et 6.3.

6.5.2 π-Calcul Stochastique

Afin de prouver que l'encodage du π-calcul stochastique est correct, nous nous reposons sur la congruence structurelle de tout processus à sa forme normale, comme montré dans la proposition 6.1. Étant donnée la définition de $\mathsf{espèces}(P)$ dans la définition 6.11, il découle que si $P \equiv P'$ alors $\mathsf{espèces}(P) \equiv \mathsf{espèces}(P')$. Ceci montre que la conversion d'un processus et de sa forme normale s'effectue sans perte d'information pour l'espèce correspondante dans la machine abstraite générique.

Démonstration de la correction des espèces. Il est trivial de voir que la proposition 6.2 est vérifiée pour le π-calcul stochastique : la fonction espèces convertit simplement une composition parallèle de processus (en forme normale) en un multi-ensemble d'espèces correspondant, alors que la fonction processus convertit un multi-ensemble en une composition parallèle structurellement congruente au processus original. □

Démonstration de la correction des réactions. Nous savons que $\mathsf{processus}(S) \xrightarrow{a} \mathsf{processus}(S')$ est vérifié si et seulement si $a = \sum_{\{\lambda, w \mid \mathsf{processus}(S) \xrightarrow{\exp(\lambda), w} \mathsf{processus}(S')\}} \lambda$ avec $a > 0$.

Nous écrivons $\mathsf{indices}(P, P')$ pour l'ensemble $\{w \mid \exists r.\ P \xrightarrow{r,w} P'\}$. En π-calcul stochastique, les indices w peuvent être soit un simple indice i pour un délai unaire, ou une paire d'indices (i_1, i_2) pour une communication binaire ou une liaison. Étant donné un indice w, nous définissons les fonctions $\mathsf{source}(w)$, $\mathsf{cible}(w)$ et $\mathsf{taux}(w)$ comme suit, où nous écrivons $i \in I$ pour signifier que l'action d'indice i est présente dans le choix correspondant à l'espèce I.

$$\begin{aligned}
\mathsf{source}(w) &\triangleq \begin{cases} I & \text{si } w = i \wedge i \in I \\ I_1 \mid I_2 & \text{si } w = (i_1, i_2) \wedge i_1 \in I_1 \wedge i_2 \in I_2 \end{cases} \\
\mathsf{cible}(w) &\triangleq P' \quad \text{si } \mathsf{source}(w) \xrightarrow{\exp(\lambda), w} P' \\
\mathsf{taux}(w) &\triangleq \exp(\lambda) \quad \text{si } \mathsf{source}(w) \xrightarrow{\exp(\lambda), w} \mathsf{cible}(P')
\end{aligned}$$

Nous disons que $w \approx w'$ est vrai si et seulement si $\mathsf{source}(w) = \mathsf{source}(w')$, $\mathsf{cible}(w) = \mathsf{cible}(w')$ et $\mathsf{taux}(w) = \mathsf{taux}(w')$. Nous écrivons $\mathsf{cls\text{-}indices}(P, P')$ pour l'ensemble des classes de $\approx$-équivalences de $\mathsf{indices}(P, P')$. Nous écrivons $\mathsf{source}(\widetilde{w})$, $\mathsf{cible}(\widetilde{w})$ et $\mathsf{taux}(\widetilde{w})$ pour $\mathsf{source}(w)$, $\mathsf{cible}(w)$ et $\mathsf{taux}(w)$ pour un certain $w \in \widetilde{w}$.

Nous observons alors que $\mathsf{processus}(S) \xrightarrow{\exp(\lambda), w} \mathsf{processus}(S')$ est vérifié si et seulement si $\mathsf{source}(w) \xrightarrow{\mathsf{taux}(w), w} \mathsf{cible}(w)$, où $S' = (S \uplus \mathsf{espèces}(\mathsf{source}(w))) \uplus \mathsf{espèces}(\mathsf{cible}(w))$. Si nous posons $\mathsf{réaction}(\widetilde{w}) = (\mathsf{espèces}(\mathsf{source}(\widetilde{w})), \mathsf{taux}(\widetilde{w}), \mathsf{espèces}(\mathsf{cible}(\widetilde{w})))$, il est alors clair qu'à partir de ces définitions et de la définition 6.11, nous avons :

$$\{\mathsf{réaction}(\widetilde{w}) \mid \widetilde{w} \in \mathsf{cls\text{-}indices}(\mathsf{processus}(S), \mathsf{processus}(S'))\} = \mathsf{ens\text{-}réactions}(S, S')$$

c.-à-d. que chaque classe de $\approx$-équivalence d'indices de $\mathsf{indices}(\mathsf{processus}(S), \mathsf{processus}(S'))$ correspond à une réaction particulière dans la machine abstraite. En écrivant $p(S)$ pour abréger $\mathsf{processus}(S)$ et $\mathsf{card}(\widetilde{w})$ pour le nombre d'indices dans $\widetilde{w}$, nous obtenons alors :

$$\begin{aligned}
& \textstyle\sum_{\{\lambda, w \mid p(S) \xrightarrow{\exp(\lambda), w} p(S')\}} \\
= \ & \textstyle\sum_{w \in \mathsf{indices}(p(S), p(S'))} \mathsf{taux}(w) \\
= \ & \textstyle\sum_{\widetilde{w} \in \mathsf{cls\text{-}indices}(p(S), p(S'))} (\mathsf{taux}(\widetilde{w}) \times \mathsf{card}(\widetilde{w})) \\
= \ & \textstyle\sum_{\{\mathsf{réaction}(\widetilde{w}) \mid \widetilde{w} \in \mathsf{cls\text{-}indices}(p(S), p(S'))\}} \mathsf{propension}(\mathsf{réaction}(\widetilde{w}), S) \\
= \ & \textstyle\sum_{O \in \mathsf{ens\text{-}réactions}(S, S')} \mathsf{propension}(O, S).
\end{aligned}$$

Ainsi, nous obtenons $a = \sum_{O \in \mathsf{ens\text{-}réactions}(S, S')} \mathsf{propension}(O, S)$ avec $a > 0$. □

6.6 Le Cas Non-Markovien

Dans cette section, nous détaillons et prouvons génériquement l'instanciation de la machine abstraite générique pour une méthode de simulation non-markovienne. En particulier, nous montrons que les instanciations proposées du π-calcul et du calcul bioambiant stochastique (appendice A) restent correctes vis-à-vis de la simulation non-markovienne proposée, donnant ainsi la première simulation non-markovienne du calcul bioambiant (le π-calcul non-markovien ayant déjà été étudié dans (Priami, 1996)). L'instanciation de κ ne satisfait malheureusement pas les requis établis par la

preuve générique. Nous discuterons toutefois d'une piste permettant le support du κ non-markovien (instancié dans l'appendice A).

De manière générale, la gestion de distributions de probabilités autres que la distribution exponentielle dans les simulations stochastiques est très peu traitée dans la littérature. Dans le cadre des calculs de processus, (Priami, 1996) étudie le π-calcul stochastique muni de distributions générales de probabilités. Ce travail a également été repris dans (Prandi, Priami & Romanel, 2008) afin d'apporter une simulation non-markovienne du langage BlenX (Dematté et coll., 2008*a*).

La difficulté majeure d'une simulation non-markovienne est la perte de la propriété d'absence de mémoire de la distribution exponentielle : alors que la probabilité de tirer une action activée ne dépend que de l'état courant dans le cas markovien, dans le cas général, cette probabilité dépend du temps écoulé depuis l'activation de l'action. Par exemple, considérons un processus P où deux actions w_1 et w_2 sont possibles (ainsi $P \xrightarrow{w_1} P_1$ et $P \xrightarrow{w_2} P_2$) et où w_1 est encore possible dans P_2. Considérons l'exécution $P \xrightarrow{w_2} P_2 \xrightarrow{w_1} P'$: la distribution de la transition $P_2 \xrightarrow{w_1} P'$ est alors différente de celle de l'action w_1 car il faut prendre en compte le temps écoulé lors de la transition $P \xrightarrow{w_2} P_2$. Priami (1996) propose une définition de l'*exécution stochastique* en détaillant les calculs des distributions des transitions lors d'une exécution non-markovienne. La définition 6.14 résume cette dernière en adaptant les notations pour l'exécution des termes d'une machine abstraite. Cette définition repose notamment sur la connaissance de la réaction (notée R_j) responsable de l'activation de la réaction R_i associée à la transition dont la distribution est recherchée : R_i est possible dès que R_j s'est produite dans l'exécution donnée.

Définition 6.14 (Exécution stochastique)**.** Soit $\xi_0 = T_0 \xrightarrow{R_0} T_1 \xrightarrow{R_1} \dots \xrightarrow{R_n} T_{n+1}$ une exécution possible. L'*exécution stochastique* correspondante est donnée par

$$\xi_{n+1} = T_0 \xrightarrow{\tilde{F}_0, R_0} T_1 \xrightarrow{\tilde{F}_1, R_1} \dots \xrightarrow{\tilde{F}_n, R_n} T_{n+1}$$

où $\tilde{F}_i = \mathbb{P}\left[\delta_i \leq t + \sum_{h=j+1}^{i-1} \delta_h \;\middle|\; \delta_i > \sum_{h=j+1}^{i-1} T_h\right]$ est la distribution de la variable aléatoire δ_i décrivant le délai associé à la transition $T_i \xrightarrow{R_i} T_{i+1}$ et avec R_j la réaction responsable de l'activation de R_i.

Nous remarquons toutefois que cette façon de « redimensionner » la distribution des actions est difficilement implémentable dans notre machine abstraite : une telle méthode implique le recalcul de toutes les distributions des réactions non-markoviennes actives après chaque transition et peut donc se révéler très coûteux à implémenter.

Ainsi, nous avons formalisé une généralisation de la Méthode par Réaction Suivante (présentée en sous-section 6.3.2) pour la gestion des réactions non-markoviennes. Cette instanciation est détaillée dans la sous-section 6.6.1 et sa correction est établie dans la sous-section 6.6.2.

6.6.1 La Méthode par Réaction Suivante dans le Cas Non-Markovien

L'utilisation de réactions non-markoviennes dans MRS est équivalente à l'introduction d'une nouvelle espèce pour chaque molécule individuelle ayant un comportement non-markovien (Gibson & Bruck, 2000). Notre machine abstraite permettant la création dynamique de nouvelles espèces, elle peut être utilisée directement comme base pour la simulation des comportements non-markoviens.

Cette approche apporte ainsi un algorithme et une implémentation simple pour la simulation non-markovienne. Toutefois, nous notons que ce calcul peut devenir assez coûteux en présence d'un très grand nombre de molécules non-markoviennes. L'algorithme 6.1 résume les étapes nécessaires pour la simulation mixte de réactions markoviennes et non-markoviennes en suivant la méthode MRS. Cet algorithme formalise et étend l'algorithme de simulation non-markovienne discuté dans (Gibson & Bruck, 2000).

Algorithme 6.1 (Méthode par Réaction Suivante Généralisée)**.**

1. Initialisation :
   (a) Initialiser le nombre de molécules ; effectuer $temps \leftarrow 0$; générer un graphe de dépendances entre les molécules et les réactions.
   (b) Si $O_i \in$ {réactions markoviennes}, calculer la propension a_i de la réaction ; générer un temps présumé t_i en suivant la distribution exponentielle avec propension a_i.
   (c) Si $O_i \notin$ {réactions markoviennes}, fixer la propension a_i à 1 ; générer un temps présumé t_i en suivant la distribution de probabilité de la réaction.
2. Soit O_μ la réaction avec le temps présumé t_μ le plus faible.
3. Changer la population des molécules en accord avec l'exécution de la réaction O_μ ; effectuer $temps \leftarrow t_\mu$.
4. Initialiser les nouvelles réactions (étapes 1.(b) et 1.(c)).
5. Pour chaque réaction affectée O_α :
   (a) Mettre à jour la propension a'_α.
   (b) Si $\alpha \neq \mu$ et $O_\alpha \in$ {réactions markoviennes}, effectuer $t_\alpha \leftarrow (a_\alpha / a'_\alpha)(t_\alpha - temps) + temps$.
   (c) Si $\alpha \neq \mu$ et $O_\alpha \notin$ {réactions markoviennes}, si $a'_\alpha = 0$, supprimer la réaction O_α et son temps présumé t_α.
   (d) Si $\alpha = \mu$ et $O_\alpha \in$ {réactions markoviennes}, générer un nombre aléatoire ρ suivant la distribution exponentielle avec propension a'_α, et effectuer $t_\alpha \leftarrow \rho + temps$.
   (e) Si $\alpha = \mu$ et $O_\alpha \notin$ {réactions markoviennes}, supprimer la réaction O_α et son temps présumé t_α.
6. Aller à l'étape 2.

La machine abstraite générique est instanciée pour la MRS non-markovienne en associant un identifiant unique à chaque individu d'une espèce participant à une réaction non-markovienne (définition 6.15). La propension d'une réaction non-markovienne est fixée arbitrairement à 1 ou 0, selon la présence de la totalité des individus participant à la réaction. Le temps présumé d'une réaction non-markovienne est mis à jour en générant une nouvelle variable aléatoire, comme dans l'équation (6.4). Cette instanciation modifie notamment la fonction suite de l'instanciation markovienne de MRS (définition 6.4) afin de renommer les espèces produites lors d'une réaction (assurant ainsi leur unicité dans l'ensemble des espèces produites).

Il est important de remarquer le traitement particulier des réactants et des produits dans cette étape : seuls les réactants n'apparaissant pas dans les produits (et réciproquement) sont considérés. Ceci évite une suppression suivie d'un ré-ajout d'une espèce présente à la fois dans les réactants $\overline{l}$ et les produits $\overline{l'}$ d'une réaction : ces étapes sont redondantes et empêchent la validation par de nombreuses instances de la proposition 6.4 nécessaire pour la correction de la simulation obtenue, détaillée dans la sous-section suivante.

Définition 6.15 (Instanciation de la machine abstraite pour la Méthode Non-Markovienne par Réaction Suivante)**.** Les fonctions espèces et suite de la définition 6.4 sont remplacées par les fonctions espèces′ et suite′, respectivement, pour permettre de renommer les espèces. La fonction renommer renomme une espèce en lui ajoutant un identifiant unique en libellé. La fonction unique(ctr) incrémente un compteur global référencé par ctx et retourne la valeur incrémentée. La fonction NM(I) s'évalue à *vrai* si l'espèce I prend part à au moins une réaction non-markovienne, sinon elle s'évalue à *faux*.

$$
\begin{aligned}
\text{espèces}'(P) &\triangleq \text{renommer-tous}(\text{espèces}(P)) \\
\text{suite}'(t,S,R) &\triangleq (\overline{I} \uplus \overline{I'}, F, \text{renommer-tous}(\overline{I'} \uplus \overline{I})), a, t' \quad \text{si } (\overline{I}, r, \overline{I'}), a, t' = \text{suite}(t,S,R) \\
\text{renommer-tous}(\overline{I}) &\triangleq \{\text{renommer}(I_i) \mid I_i \in \overline{I}\} \\
\text{renommer}(I) &\triangleq \begin{cases} I^{\text{unique}(ctr)} & \text{si } NM(I) \\ I^0 & \text{sinon.} \end{cases} \\
\text{renommer}(I^{ctr}) &\triangleq \text{renommer}(I)
\end{aligned}
$$

6.6.2 Correction

Nous montrons ici une méthode de preuve générique concernant la simulation non-markovienne effectuée par notre machine abstraite instanciée avec la méthode de simulation MRS présentée dans la définition 6.15.

Le calcul instancié est bien entendu supposé posséder une sémantique non-markovienne permettant de générer l'exécution stochastique donnée par la définition 6.14. Ceci implique notamment la formalisation de l'activation d'une action par l'exécution d'une transition. Dans le cas du π-calcul stochastique ou du calcul bioambiant, cette notion peut être dérivée naturellement : une action est activée dès que tous ses processus impliqués sont présents (c.-à-d., dès que la transition générée par l'action devient possible) (Priami, 1996).

En suivant la démarche proposée en section 6.5, nous établissons la proposition 6.4 qui, si elle est vérifiée par le calcul instancié, implique la correction de la simulation non-markovienne (théorème 6.3) Nous supposons les propositions 6.2 et 6.3 page 99 correctes. Ces dernières démontrant notamment la correction qualitative de la machine abstraite (c.-à-d., sans tenir compte des probabilités des transitions), il est suffisant de prouver la correction de l'exécution stochastique obtenue.

Cette proposition assure qu'une réaction non-markovienne active dans deux termes T_1 et T_2 ne sera pas modifiée par la fonction m-à-j de la méthode MRS lors d'une transition de T_1 vers T_2. Nous rappelons que w dénote l'identifiant unique d'une transition dans le calcul.

Proposition 6.4 (Préservation des réactions)**.** $\forall T_1, T_2, T_1', T_2'$ *si* $T_1 \xrightarrow{a,(\overline{I},F,\overline{I'})} T_2$, $T_1 \xrightarrow{1,(\overline{J_1},F_1,\overline{J_1'})} T_1'$, $T_2 \xrightarrow{1,(\overline{J_2},F_2,\overline{J_2'})} T_2'$, *avec* NM($F_1$) *et* NM($F_2$), *et si* $[\![T_1]\!] \xrightarrow{w} [\![T_1']\!]$ *et* $[\![T_2]\!] \xrightarrow{w} [\![T_2']\!]$, *alors* $\overline{I} \cap \overline{J}_1 = \emptyset$.

Sous cette condition, le théorème 6.3 démontre que les probabilités des transitions suivent exactement celles d'une exécution stochastique correcte (définition 6.14).

Théorème 6.3 (Validité des probabilités)**.** *Étant donné un processus initial P_0 du calcul, la proposition 6.4 implique que toute exécution $(\!|P_0|\!)_0 \xrightarrow{R_0} T_1 \xrightarrow{R_1} \ldots \xrightarrow{R_n} T_{n+1} \ldots$ de la machine est une exécution stochastique correcte.*

Démonstration. D'après la définition 6.14, l'exécution est correcte si, pour tout $i \in [1; n]$, la transition $T_i \xrightarrow{R_i} T_{i+1}$ suit la distribution $\tilde{F}_i = \mathbb{P}\left[\delta_i \leq t + \sum_{h=j+1}^{i-1} \delta_h \;\middle|\; \delta_i > \sum_{h=j+1}^{i-1} T_h\right]$. D'après RMS (définitions 6.4 et 6.15), $\delta_i = t(R_{i+1}) - t(R_i)$ avec $t(R_i) = t_i^0 + \mathsf{délai}(R_i)$, où t_i^0 est le temps où R_i devient active et $\mathsf{délai}(R_i)$ une variable aléatoire suivant la distribution de la réaction R_i. En particulier, nous avons $t_i^0 = \sum_{h=0}^{j} \delta_j$, où R_j est la réaction activant R_i.

Nous remarquons que $t(R_i)$ est calculé à chaque mise à jour de la réaction R_i dans la machine (lors d'un changement de propension). Ainsi, $t(R_i)$ possède une valeur correcte si et seulement si son calcul est effectué juste après l'application de la réaction R_j, et plus jamais après. La proposition 6.4 assure une telle propriété : l'application d'une réaction n'influençant pas l'activation de R_i n'en supprime aucun réactant. □

Au regard des instanciations présentées dans la section 6.4, nous discutons brièvement de la correction de leur simulation non-markovienne.

Concernant le π-calcul stochastique, la preuve de la proposition 6.4 découle de la sémantique du π-calcul et de son utilisation directe lors de l'instanciation de la machine (définition 6.11) : si un processus $X(\tilde{n})$ évolue vers un autre processus (seul ou simultanément avec un processus parallèle), alors $X(\tilde{n})$ n'est plus présent dans le terme suivant la réduction. Ainsi, en utilisant les notations de la proposition 6.4, si une espèce non-markovienne $X(\tilde{n})^{id}$, participant à la réaction identifiée w dans le calcul, est telle que $X(\tilde{n})^{id} \in \overline{I} \cap \overline{J_1}$, alors la transition $[\![T_2]\!] \xrightarrow{w} [\![T_2']\!]$ est impossible. La simulation non-markovienne du π-calcul stochastique est donc correcte (théorème 6.3).

La correction de l'instanciation du calcul bioambiant est acquise de manière similaire. Nous remarquons que les espèces de localisation (a, b) ne sont pas concernées par la fonction renommer de la définition 6.15, ces dernières n'étant jamais considérées non-markoviennes (c.-à-d.$\mathsf{NM}((a, b))$ est toujours faux). La gestion de l'action `fusion`, discutée en fin de sous-section A.2, est acquise via une étape de normalisation du terme de la machine afin d'appliquer un renommage des localisation des espèces (spécifié par une espèce d'alias du type $a = b$). Dans le cas non-markovien, cette étape, non formalisée dans la définition présentée de notre machine abstraite, doit également être effectuée par la fonction suite$'$ de la définition 6.15 afin de préserver les réactions démarrées dans les bioambients fusionnés. Prenons l'exemple de la réaction suivante, sélectionnée par la fonction suite :

$$X^{a1} + Y^{b2} + (a, root) + (b, root) \longrightarrow X_2^{b3} + Y_2^{b4} + (a = b) + (b, root)$$

où $1, 2, 3, 4$ sont les identifiants rajoutés par la fonction renommer. La fonction suite$'$ applique alors la normalisation $a = b$ sur les espèces réactantes et produits, afin d'obtenir la réduction suivante :

$$X^{b1} + Y^{b2} \longrightarrow X_2^{b5} + Y_2^{b6} + (a = b)$$

Nous retrouvons alors les bonnes propriétés de l'instanciation du π-calcul stochastique montrant la validité de la simulation non-markovienne du calcul bioambiant.

Nous remarquons enfin que l'intanciation de κ proposée en sous-section A.2 ne satisfait pas la proposition 6.4 dans le cas général. Nous illustrons ceci par un simple exemple. Considérons les

règles κ suivantes, que nous supposons non-markoviennes :

$$A(x_u) \to A(x_p)$$
$$A(y), B(y) \to A(y^1), B(y^1)$$

L'instance de la machine abstraite appliquée sur la solution $A(x_u, y), B(y)$ produira deux espèces $A(x_u, y)^1$, $B(y)^2$ et deux réactions non-markoviennes, $A(x_u, y)^1 \to A(x_p, y)^3$ et $A(x_u, y)^1 + B(y)^2 \to A(x_u, y^{\nu 1})^4 + B(y^{\nu 1})^5$. Il apparaît qu'après application d'une de ces deux réactions, l'autre réaction sera desactivée et une nouvelle apparaîtra, afin de refléter le changement dans l'agent A. Cependant, la règle correspondante reste active : il n'y a pas de conflits entre les deux règles, ces dernières peuvent s'exécuter en parallèle sans se désactiver l'une-l'autre.

Une solution serait alors d'utiliser une méthode similaire à la gestion de l'action `fusion` dans l'intanciation du calcul bioambiant : lors de la modification d'un agent, une étape de normalisation appliquerait cette modification directement sur les termes de la machine. Toutefois, cette normalisation ne doit s'effectuer qu'en l'absence de conflits entre les modifications effectuées sur un agent et les réactions où cet agent est présent.

6.7 Modèles Multi-Calculs

Jusqu'ici, nous avons abordé la simulation de modèles dont le comportement complet peut être dérivé à partir d'un seul calcul de processus, tel que le calcul bioambiant. Cependant, la machine abstraite générique définie dans la section 6.2 est assez générale pour pouvoir supporter des modèles multi-calculs (hétérogènes) utilisant des composants écrits avec différents calculs de processus. Cette approche permet ainsi de choisir le langage le plus approprié pour la formalisation indépendante des différents composants d'un système.

6.7.1 Définir un Modèle Multi-Calcul

Étant donné un calcul C, nous écrivons $\mathbb{P}_C$ pour le type des processus de ce calcul, et $\mathbb{I}_C$ pour le type correspondant aux espèces. La définition des réactions en tant que triplet $(\overline{l}, r, \overline{l'})$ induit un type $\mathbb{R}_C$ pour les réactions du calcul C dépendant du type associé aux espèces $\mathbb{I}_C$. En dénotant par $\mathcal{P}_{\text{fin}}(X)$ l'ensemble des sous-ensembles finis de X, le type des fonctions espèces et réactions pour le calcul C peut se résumer comme ceci :

$$\begin{aligned} \text{espèces}_C &: \mathbb{P}_C \to \mathcal{P}_{\text{fin}}(\mathbb{I}_C) \\ \text{réactions}_C &: \mathbb{I}_C \to \mathcal{P}_{\text{fin}}(\mathbb{I}_C) \to \mathcal{P}_{\text{fin}}(\mathbb{R}_C) \end{aligned}$$

Supposons maintenant que nous souhaitons définir un modèle hétérogène en utilisant le sous-ensemble fini de calculs $\overline{C} \equiv \{C_1, \ldots, C_n\}$, chacun d'eux étant défini comme précédemment. Nous utilisons alors $\overline{C}$ comme un calcul de manière identique à C_i. En supposant que les types de processus et d'espèces des sous-calculs sont disjoints, nous définissons les processus et espèces de C en termes de ceux des sous-calculs, comme suit.

$$\begin{aligned} \mathbb{I}_{\overline{C}} &\triangleq \mathbb{I}_{C_1} \uplus \cdots \uplus \mathbb{I}_{C_n} \\ \mathbb{P}_{\overline{C}} &\triangleq \mathbb{P}_{C_1} \times \cdots \times \mathbb{P}_{C_n} \times \mathcal{P}_{\text{fin}}(\mathbb{R}_{\overline{C}}) \end{aligned}$$

Une espèce du calcul $\overline{C}$ étant l'espèce d'un et un seul sous-calcul, nous pouvons toujours déterminer le calcul correspondant à une espèce particulière. Les processus du calcul $\overline{C}$ correspondent aux processus de chaque sous-calcul C_i Les réactions utilisant, elles, les espèces du calcul $\overline{C}$, elles sont la seule charnière entre les calculs. Deux types de réactions sont alors définies : celles correspondant aux réactions internes des sous-calculs C_i et celles décrivant les interactions entre des espèces de sous-calculs différents. Nous appelons ces dernières les *réactions glus* : elles fournissent un lien entre les différents composants. Ces réactions doivent être spécifiées par l'utilisateur et devraient être dérivées naturellement de la structure du modèle hétérogène.

Nous définissons tout d'abord l'extraction des espèces d'un calcul donné à partir des espèces du multi-calcul : si $\widetilde{I} \in \mathbb{I}_{\overline{C}}$ alors $\pi_{C_i}(\widetilde{I}) = \{I \mid I \in \overline{I} \wedge I \in \mathbb{I}_{C_i}\}$. Ainsi, $\pi_{C_i}(\widetilde{I}) \in \mathbb{I}_{C_i}$ pour tout i, et $\{\pi_{C_i}(\widetilde{I}) \mid i \in \{1, \ldots, n\}\}$ est une partition de $\widetilde{I}$. Soit $P \in \mathbb{P}_{\overline{C}}$ un processus multi-calcul tel que $P \equiv (P_{C_1}, \ldots, P_{C_n}, G)$ où G est l'ensemble des réactions glus spécifiées par l'utilisateur. Les fonctions **espèces** et **réactions** pour le calcul $\overline{C}$ sont définies comme suit.

$$\begin{array}{rcl} \mathsf{espèces}_{\overline{C}}(P) & \triangleq & \mathsf{espèces}_{C_1}(P_1) \uplus \cdots \uplus \mathsf{espèces}_{C_n}(P_n) \\ \mathsf{réactions}_{\overline{C}}(I, \widetilde{I}) & \triangleq & \mathsf{réactions}_{C_i}(I, \pi_{C_i}(\widetilde{I})) \uplus \mathsf{glu}(I, \widetilde{I}, O) \quad \text{si } I \in \mathbb{I}_{C_i} \end{array}$$

où la fonction **glu** est définie par :

$$\mathsf{glu}(I, \widetilde{I}, G) \triangleq [O \mid O \in G \wedge I \in \mathsf{réactants}(O) \wedge \mathsf{réactants}(O) \subseteq \widetilde{I} \cup \{I\}]$$

La population initiale d'espèces pour le calcul $\overline{C}$ est simplement la réunion des population d'espèces initiales pour chaque sous-calcul. Pour calculer les réactions entre une espèce I et l'ensemble des espèces existantes $\widetilde{I}$, nous utilisons dans un premier temps la fonction **réactions** du sous-calcul approprié auquel nous ajoutons les réactions contenues dans G dont I est un réactant (ceci permet de s'assurer que les réactions glus seront présentes en un seul exemplaire dans l'ensemble total des réactions). Nous pouvons maintenant utiliser les techniques présentées précédemment pour dériver un simulateur pour des modèles multi-calculs utilisant des composants écrits avec différents calculs.

6.7.2 Exemple Simple

Nous considérons ici un exemple simple de modèle multi-calcul qui utilise des composants écrits en π-calcul stochastique et en κ. Nos processus en π-calcul stochastique sont définis comme suit.

$$A \triangleq !x.C \qquad B \triangleq ?x.\mathbf{0} \qquad C \triangleq ?z.\mathbf{0}$$

L'utilisation du canal z, utilisé seulement en réception, assure que le processus C ne prendra part à aucune réaction. Notre processus initial en π-calcul sera cent copies de A et cent copies de B. Notre règle κ est la suivante :

$$X(a), Y(a) \quad \rightarrow \quad X(a^1), Y(a^1)$$

Notre solution κ initiale consistera en cent copies de $Y(a)$. Enfin, nous définissons la réaction glu suivante ;

$$C \longrightarrow X(a)$$

Cette réaction glu lie les processus en π-calcul stochastique et les agents κ, créant un modèle multi-calcul. En partant avec cent copies de A, B et $Y(a)$, la première interaction sera entre A et B créant alors C. C ne participe à aucune réaction en π-calcul stochastique mais peut déclencher une réaction glu, qui produira l'agent $\kappa\, X(a)$. Ceci active alors la partie κ de notre modèle multi-calcul : les agents $X(a)$ et $Y(a)$ peuvent se lier sur le site a, pour devenir $X(a^1), Y(a^1)$ ne prenant part à aucune autre réaction. Ceci montre comment les réactions glus permettent la communication entre des composants séparés pour un modèle multi-calcul.

6.7.3 Modèles Récursifs

Nous notons que les sous-calculs mentionnés dans la section précédente peuvent également être déjà des calculs hétérogènes dont les espèces et réactions sont calculées récursivement avec la même procédure. Ainsi, nous pouvons définir des modèles hiérarchiques, où les sous-composants peuvent contenir des sous-composants écrits avec différents langages spécifiques, en utilisant notre machine abstraite générique comme couche commune pour l'implémentation. Ceci peut être une approche prometteuse pour la modélisation de très grands systèmes biologiques, ceux-ci étant souvent, par nature, hiérarchiques et composés de nombreuses unités fonctionnelles différentes, tout en conservant un mode fondamental d'interaction par réactions chimiques.

Cette capacité à définir des modèles complexes, hétérogènes et récursifs dans un même cadre que les modèles de calculs simples montre la grande expressivité de la machine abstraite générique présentée dans ce chapitre.

6.8 Discussion

Dans ce chapitre, nous avons présenté une machine abstraite générique pour la simulation des calculs de processus avec potentiellement un nombre non-borné d'espèces et de réactions. Nous avons instancié la machine abstraite avec deux méthodes de simulation markovienne et une non-markovienne ; et avec trois calculs de processus : le π-calcul stochastique, le calcul bioambiant et κ (appendice A). Nous avons démontré la correction des instanciations markoviennes de la machine pour ces trois calculs au moyen d'une méthode de preuve générique. Enfin, nous avons défini une méthode générale pour simuler simultanément des calculs de processus différents.

Pour chacun des trois calculs présentés dans ce chapitre et l'appendice A, la fonction spécifique réactions est dérivée depuis la sémantique sous-jacente du calcul. Cette approche repose sur l'existence d'une fonction processus qui transforme un multi-ensemble d'espèces en un processus, et sur la possibilité d'appliquer les réductions sur les processus extraits de leur contexte. Ceci suggère la possibilité de dériver automatiquement une instanciation pour beaucoup de calculs de processus, directement à partir de leur sémantique de réduction. La caractérisation des calculs de processus pour lesquels une telle procédure est possible serait intéressante à étudier.

En plus de définir les fonctions espèces et réactions, l'utilisateur doit également décider la constitution d'une *espèce* pour le calcul voulu. Par exemple, notre instanciation avec le π-calcul stochastique définit une espèce comme étant soit une instance de processus, soit un complexe de processus. Ceci permet d'optimiser le traitement des complexes au cours de la simulation, résultant en un gain d'efficacité. Des optimisations similaires pourraient être possibles pour l'instanciation avec le calcul bioambiant et κ et sont laissées comme de possibles travaux futurs.

L'utilisation de techniques de nettoyage de la mémoire (*garbage collection*), pour supprimer les espèces obsolètes et les réactions associées des termes de la machine abstraite, peut s'avérer nécessaire afin de supporter la simulation de systèmes générant un très grand nombre d'espèces rapidement consommées. Cette étape de nettoyage n'a aucun impact sur la correction de la simulation (seule les espèces avec une population nulle sont supprimées), et est ainsi périphérique aux définitions principales de la machine abstraite. Une façon générique d'incorporer cette étape pourrait utiliser une structure référençant les espèces avec une population nulle et définissant une fonction pour en extraire un ensemble d'espèces à supprimer, via l'utilisation de diverses heuristiques (temps depuis la mise à zéro de la population, etc.), qui peuvent dépendre du calcul instancié.

La suppression des espèces et réactions rendues obsolètes au cours d'une simulation pourra également être étudiée en détail, ceci pouvant avoir un impact significatif sur les performances d'une simulation à large échelle, et ces heuristiques de nettoyage dépendant vraisemblablement du calcul instancié.

Notre machine abstraite générique aspire à simuler un grand nombre de calculs de processus. Ceci inclut des calculs de processus capables d'effectuer des interactions n-aires, et des réactions avec des coefficients stœchiométriques arbitraires. Afin d'illustrer la flexibilité de notre approche, nous avons actuellement utilisé cette machine abstraite pour implémenter la simulation du calcul *DNA Strand Displacement* (DSD) pour la modélisation des circuits d'ADN (Phillips & Cardelli, 2009), du calcul *Genetic Engineering of Cells* (GEC) pour la modélisation de composants génétiques (Pedersen & Phillips, 2009), et du calcul *Stochastic Pi Machine* (SPiM) pour la modélisation générale des systèmes biologiques (Wang, Cardelli, Phillips, Piterman & Fisher, 2009). Les simulateurs de ces trois calculs sont disponibles en ligne sur `http://research.microsoft.com/dna`, `http://research.microsoft.com/gec` et `http://research.microsoft.com/spim`, respectivement. Le chapitre suivant montre également l'application de cette machine abstraite aux Frappes de Processus. Par expérience, l'approche présentée dans ce chapitre a permis d'accélérer significativement le développement de ces langages, en réduisant la difficulté d'implémentation des algorithmes de simulation stochastique spécifiques aux calculs, et permettant la réutilisation du code entre les différents projets. Des travaux en cours s'intéressent à l'implémentation d'autres calculs de processus tels que le *Brane calculus* (Cardelli, 2004) et un calcul basé sur les diagrammes d'états (Harel, 1987). Nous regardons également le développement de modèles hétérogènes complexes, tels que des modèles DSD d'ADN de molécules interagissant avec un modèle SPiM de la physiologie d'une cellule.

Chapitre **7**

Paramètres Temporels et Stochastiques des Frappes de Processus

Dans ce chapitre, nous appliquons les contributions apportées par les chapitres 5 et 6 pour introduire des paramètres temporels et stochastiques au sein des Frappes de Processus. Les actions des Frappes de Processus sont alors paramétrées par un taux d'utilisation et un facteur d'absorption de stochasticité, dérivables depuis un intervalle de temps ou d'une durée moyenne et d'une variance donnée.
Nous traitons ainsi la vérification formelle de propriétés quantitatives (via PRISM), la traduction en π-calcul stochastique markovien, et la simulation non-markovienne des Frappes de Processus dont les actions sont munies de taux d'utilisation et de facteurs d'absorption de stochasticité

7.1 Préliminaires

L'introduction de durées pour les transitions effectuées par l'exécution d'un modèle permet à la fois de restreindre les comportements induits par le modèle et de reproduire quantitativement le comportement du système modélisé. Ces durées sont généralement spécifiées dans deux cadres distincts : soit la durée suit une variable aléatoire (typiquement exponentielle) afin de calculer les probabilités d'observer certains comportements ; soit la durée est déterminée par un intervalle de temps afin de contraindre de manière déterministe les comportements obtenus.

Bien que l'utilisation de la distribution exponentielle pour gouverner les délais des transitions a un réel sens dans le cas de modélisations à l'échelle des molécules (une transition par réaction chimique) (Gillespie, 1977), dans le cadre des modélisations abstraites hybrides des RRB (sous-section 2.2.3 page 13), la notion de temporalité apportée par une variable alétoire exponentielle est bien moins pertinente, le changement de niveau qualitatif abstrayant généralement un grand nombre de réactions chimiques.

Dans le chapitre 5, nous avons proposé une méthode permettant de concilier à la fois la spécification de paramètres temporels et la spécification de paramètres stochastiques. Nous imposons un cadre global stochastique dans lequel la variance des durées, dotées d'un taux d'utilisation, peut être contrôlée par un facteur d'absorption de stochasticité. Il est alors possible d'obtenir un lien entre un intervalle de temps et le couple de paramètres stochastique du taux d'utilisation r et du facteur d'absorption sa. Les durées obtenues suivent la somme de sa variables aléatoires exponentielles

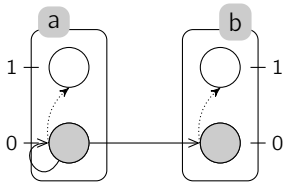


Figure 7.1 – Exemple de Frappes de Processus traduites en π-calcul stochastique et PRISM avec facteur d'absorption dans les sous-sections 7.2.1 et 7.2.2. L'état initial est défini par les processus grisés.

avec un taux $r.sa$, autrement appelé distribution d'Erlang ; en moyenne ces durées valent r^{-1} unités de temps avec une variance de $r^{-2}sa^{-1}$ (voir section 5.3 page 63).

Malgré la perte de l'hypothèse markovienne sur les modèles ainsi paramétrés, nous avons montré dans le chapitre 5 qu'il est possible de construire cette absorption de stochasticité dans le cadre markovien, notamment en π-calcul stochastique et en PRISM, permettant notamment la vérification formelle de propriétés quantitatives de ces modèles.

Dans le chapitre 6, la simulation des formalismes concurrents a été traitée en définissant une machine abstraite générique simplifiant la définition et la preuve des simulations de nombreux calculs de processus. Nous avons également montré l'instanciation de cette machine pour la simulation non-markovienne, ce qui a une application directe dans ce chapitre pour la simulation des modèles stochastiques distribués avec Erlang.

La spécialisation dans le cadre des Frappes de Processus de la construction du facteur d'absorption de stochasticité en π-calcul stochastique et en PRISM est traitée dans la section 7.2. La section 7.3 définit et démontre l'instanciation de la machine abstraite générique du chapitre 6 avec les Frappes de Processus. Enfin, nous discutons des apports de ce chapitre dans la section 7.4.

7.2 Traduction des Frappes de Processus avec Absorption de Stochasticité

Dans cette section, nous reprenons dans le cadre spécifique des Frappes de Processus les constructions présentées dans le chapitre 5 du facteur d'absorption de stochasticité en π-calcul stochastique et en PRISM.

En outre de montrer une construction directe depuis les Frappes de Processus munis de paramètres temporels et stochastiques dans des formalismes classiques, il apparaît que les constructions de l'absorption de stochasticité se retrouvent simplifiées comparé au cadre général du π-calcul.

Les sous-sections 7.2.1 et 7.2.2 détaillent respectivement la traduction en π-calcul stochastique markovien et en PRISM des Frappes de Processus dont la durée des actions suit une distribution de Erlang paramétrée par un taux d'utilisation et un facteur d'absorption de stochasticité. À titre d'exemple, nous exhibons la traduction des Frappes de Processus définies dans la figure 7.1 dans ces formalismes.

Définition 7.1 (**ΠC**$(\Sigma, L, \mathcal{H})$)**.** Pour toute sorte $a \in \Sigma$ et tout processus $a_i \in L_a$, nous définissons le processus π-Calcul A_i comme suit :

$$A_i(id, \tilde{c}, \tilde{k}) \triangleq \sum_{\substack{h = a_i \to b_j \Gamma^{\uparrow} b_k \in \mathcal{H} \\ a \neq b}} !h(id).A_i(id, \tilde{c}, \tilde{k})$$

$$+ \sum_{\substack{h = b_j \to a_i \Gamma^{\uparrow} a_k \in \mathcal{H} \\ a \neq b}} ?h(hid). \begin{pmatrix} [hid = k[h] \wedge (c[h] = sa_h - 1 \vee sa_h = 1)] \\ \quad A_k(\nu id, \tilde{c^{a_k}}, \tilde{k^{a_k}}) \\ | \, [hid = k[h] \wedge c[h] < sa_h - 1] \\ \quad A_i(id, \tilde{c}\{h \mapsto c[h] + 1\}, \tilde{k}) \\ | \, [hid \neq k[h]] \\ \quad A_i(id, \tilde{c}\{h \mapsto 1\}, \tilde{k}\{h \mapsto hid\}) \end{pmatrix}$$

$$+ \sum_{h = a_i \to a_i \Gamma^{\uparrow} a_k \in \mathcal{H}} \tau_h. \begin{pmatrix} [c[h] = sa_h - 1] & A_k(\nu id, \tilde{c^{a_k}}, \tilde{k^{a_k}}) \\ | \, [c[h] < sa_h - 1] & A_i(id, \tilde{c}\{h \mapsto c[h] + 1\}, \tilde{k}) \end{pmatrix}$$

avec $\tilde{c^{a_k}} = \tilde{k^{a_k}} = \{h \mapsto 0 \mid h \in \mathcal{H} \wedge \mathsf{cible}(h) = a_k\}$ les vecteurs initiaux des compteurs et identifiants des frappeurs, respectivement. À tout canal h ou action interne τ_h est affecté un taux d'utilisation $r_h.sa_h$.
Étant donné un état $s \in L$ des Frappes de Processus, nous définissons $[\![s]\!]$ le processus π-Calcul correspondant comme étant la composition parallèle des processus présents dans s :

$$[\![s]\!] = \prod_{a \in \Sigma} A_i(\nu id, \tilde{c^{a_i}}, \tilde{k^{a_i}}) \qquad \text{si } s[a] = a_i \ .$$

7.2.1 Des Frappes de Processus vers le π-Calcul Stochastique

Dans la sous-section 4.2.2 (chapitre 4), nous avons montré la traduction des Frappes de Processus en π-calcul (définition 4.3 page 47). En attachant aux actions h un taux d'utilisation r_h et un facteur d'absorption de stochasticité sa_h, nous obtenons alors, avec la même définition, un modèle des Frappes de Processus en π-calcul stochastique distribué par Erlang.

En se basant sur la traduction du π-calcul stochastique distribué par Erlang en π-calcul stochastique distribué exponentiellement présentée dans la sous-section 5.3.2 page 64, nous montrons dans la définition 7.1 une traduction directe des Frappes de Processus distribuées par Erlang en π-calcul stochastique markovien. Pour rappel, cette construction remplace une action suivant un taux r et un facteur d'absorption de stochasticité sa par sa actions suivant un taux $r.sa$. Contrairement à la traduction générique présentée en sous-section 5.3.2, cette traduction des Frappes de Processus ne requiert pas la capacité à avoir des listes extensibles dans les valeurs des paramètres, ni la capacité à avoir des compétitions dont la taille dépend de la valeur de certains paramètres du processus. En effet, contrairement au cas général en π-calcul, les actions sur les canaux résultant de la traduction des Frappes de Processus ne mettent en jeu que deux processus : un et un seul émetteur et un et un seul récepteur. Ainsi, il n'est plus nécessaire de partager des canaux privés afin d'identifier la communication à absorber ; seule l'identification de l'émetteur reste nécessaire.

Un canal h est défini avec un taux d'utilisation $r_h.sa_h$ pour chaque action $h \in \mathcal{H}$. À chaque processus a_i des Frappes de Processus $(\Sigma, L, \mathcal{H})$, un processus π-calcul A_i est défini, prenant en

paramètre un canal identifiant id, un compteur c_h et un canal k_h pour chaque action $h \in \mathcal{H}$ telle que a_i est la cible. Pour chaque frappe de a_i sur lui-même, une transition interne est effectuée, à l'issue de laquelle le processus est redémarré si le compteur associé n'a pas atteint le facteur d'absorption, ou alors le processus est remplacé par le processus correspondant au bond de l'action. Pour chaque frappe de a_i vers un autre processus, le processus π-calcul A_i écrit sur le canal associé et redémarre. Enfin, pour chaque frappe sur le processus a_i, le processus π-calcul A_i lit le canal identifiant hid du frappeur sur le canal h associé et effectue plusieurs tests : si $hid = k_h$, selon la valeur du compteur c_h, le processus redémarre en incrémentant c_h ou est remplacé par le processus π-calcul correspondant au bond ; si hid est différent de k_h, alors le frappeur a changé, et l'absorption reprend depuis le début, avec $k_h = hid$.

Cette construction étant un cas particulier de la construction présentée en sous-section 5.3.2, sa correction en découle directement.

La traduction des Frappes de Processus de la figure 7.1 en suivant cette définition résulte en le processus $A_0(\nu id, 0, 0) \mid B_0(\nu id, 0, 0)$ avec les définition des processus π-calcul suivants où l'on considère $h_1 = a_0 \rightarrow a_0 \,\Rsh\, a_1$ et $h_2 = a_0 \rightarrow b_0 \,\Rsh\, b_1$:

$$
\begin{aligned}
A_1(id) &\triangleq \mathbf{0} \qquad\qquad B_1(id) \triangleq \mathbf{0} \\
A_0(id, c_{h_1}, k_{h_1}) &\triangleq\ !h_2(id).A_0(id, c_{h_1}, k_{h_1}) \\
&\quad +\tau_{h_1}.\begin{pmatrix} [c_{h_1} = sa_{h_1} - 1] & A_1(\nu id) \\ \mid [c_{h_1} < sa_{h_1} - 1] & A_0(id, c_{h_1} + 1, k_{h_1}) \end{pmatrix} \\
B_0(id, c_{h_2}, k_{h_2})) &\triangleq\ ?h_2(hid).\begin{pmatrix} [hid = k_{h_2} \wedge c_{h_2} = sa_{h_2} - 1 \vee sa_{h_2} = 1]\ B_1(\nu id) \\ \mid [hid = k_{h_2} \wedge c_{h_2} < sa_{h_2} - 1]\ B_0(id, c_{h_2} + 1, k_{h_2}) \\ \mid [hid \neq k_{h_2}]\ B_0(id, 1, hid) \end{pmatrix}
\end{aligned}
$$

7.2.2 Des Frappes de Processus vers PRISM

La traduction des Frappes de Processus en PRISM permet la vérification automatique de propriétés quantitatives via le logiciel PRISM (Hinton et coll., 2006). La définition 7.2 montre une spécialisation de la construction de l'absorption de stochasticité en PRISM présentée dans la section 5.5 page 75 pour le cas particulier des Frappes de Processus.

La construction proposée est très proche de la section 5.5 : nous créons un module par sorte des Frappes de Processus ; chaque module possède une variable locale décrivant le processus actif courant. Pour chaque action dont la sorte est la cible, un compteur pour l'absorption de stochasticité est créé ; ce compteur est incrémenté à chaque fois que l'action est jouée, jusqu'à atteindre la valeur du facteur d'absorption de stochasticité : le processus courant est mis à jour et les compteurs remis à un. Dans le cas où la sorte a le rôle de frappeur et que son processus courant change, elle indique via une variable locale (notée d_h) que le compteur d'absorption de l'action h doit être remis à un.

La correction de cette traduction peut être déduite de la traduction des Frappes de Processus en Automates Finis Communicants (théorème 4.1 page 46) pour l'aspect qualitatif, l'aspect quantitatif ayant été traité dans la section 5.5.

Définition 7.2 (**PRISM**$(\Sigma, L, \mathcal{H})$)**.** Pour chaque sorte $a \in \Sigma$, nous définissons un module PRISM `proc_A` contenant les variables $varA$ et les actions $actA$ définies comme suit :

$$
\begin{aligned}
varA = &\{\mathtt{a} : [0..l_a] \mid L_a = [0; l_a]\} \\
&\quad \leftarrow \text{représente le processus de sorte } a \text{ actif ;} \\
&\cup \{\mathtt{c\_}h\text{: } [1..sa_h] \texttt{ init 1} \mid h \in \mathcal{H} \wedge \Sigma(\mathsf{cible}(h)) = a\} \\
&\quad \leftarrow \text{compteur d'absorption de stochasticité ;} \\
&\cup \{\mathtt{d\_}h\text{: } \texttt{bool init false} \mid h \in \mathcal{H} \wedge \mathsf{frappeur}(h) \neq \mathsf{cible}(h) \wedge \Sigma(\mathsf{frappeur}(h)) = a\} \\
&\quad \leftarrow \text{remise à zéro du compteur d'absorption de stochasticité.} \\
actA = &\{[]\ (\mathtt{a} = i)\ \&\ (\mathtt{c\_}h < sa_h) \rightarrow r_h * sa_h : (\mathtt{c\_}h' = \mathtt{c\_}h + 1); \\
&\quad []\ (\mathtt{a} = i)\ \&\ (\mathtt{c\_}h = sa_h) \rightarrow r_h * sa_h : (\mathtt{a}' = j)\ \&\ R_{a_i}\ \&\ S_{a_i}; \\
&\quad \mid h = a_i \rightarrow a_i \nearrow a_j \in \mathcal{H}\} \\
&\quad \leftarrow \text{auto-frappes d'un processus de sorte } a \text{ ;} \\
&\cup \{[h]\ (\mathtt{a} = i)\ \&\ \mathtt{d\_}h \rightarrow r_h * sa_h : (\mathtt{c\_}h' = 2); \\
&\quad [h]\ (\mathtt{a} = i)\ \&\ !\mathtt{d\_}h\ \&\ (\mathtt{c\_}h < sa_h) \rightarrow r_h * sa_h : (\mathtt{c\_}h' = \mathtt{c\_}h + 1); \\
&\quad [h]\ (\mathtt{a} = i)\ \&\ !\mathtt{d\_}h\ \&\ (\mathtt{c\_}h = sa_h) \rightarrow r_h * sa_h : (\mathtt{a}' = j)\ \&\ R_{a_i}\ \&\ S_{a_i}; \\
&\quad \mid h = b_k \rightarrow a_i \nearrow a_j \in \mathcal{H} \wedge b \neq a \wedge sa_h > 1\} \\
&\quad \leftarrow \text{frappes sur un processus de sorte } a \text{ avec absorption de stochasticité ;} \\
&\cup \{[h]\ (\mathtt{a} = i) \rightarrow r_h * sa_h : (\mathtt{a}' = j)\ \&\ R_{a_i}\ \&\ S_{a_i}; \mid h = b_k \rightarrow a_i \nearrow a_j \in \mathcal{H} \wedge b \neq a \\
&\qquad\qquad \wedge sa_h = 1\} \\
&\quad \leftarrow \text{frappes sur un processus de sorte } a \text{ sans absorption de stochasticité,} \\
&\cup \{[h]\ (\mathtt{a} = i) \rightarrow (\mathtt{d\_}h' = \mathtt{false}); \mid h \in \mathcal{H} \wedge \mathsf{frappeur}(h) = a_i \\
&\qquad\qquad \wedge \mathsf{frappeur}(h) \neq \mathsf{cible}(h)\} \\
&\quad \leftarrow \text{frappes par un processus de sorte } a
\end{aligned}
$$

où $R_{a_i} = \underset{h \in \mathcal{H} \mid \mathsf{cible}(h) = a_i}{\&} (\mathtt{c\_}h' = 1)$ et $S_{a_i} = \underset{h \in \mathcal{H} \mid \mathsf{frappeur}(h) = a_i \wedge \mathsf{frappeur}(h) \neq \mathsf{cible}(h)}{\&} (\mathtt{d\_}h' = \mathtt{c\_}h > 1)$.

Le listing 7.1 illustre la traduction des Frappes de Processus de la figure 7.1 en PRISM à l'aide de cette définition. Cet exemple est en fait une variante en Frappes de Processus de l'exemple présenté en sous-section 5.5.2 et traduit en PRISM avec absorption de stochasticité dans le listing 5.2 page 80. Ainsi, l'analyse formelle de l'impact des paramètres temporels sur le comportement obtenu (ici, la probabilité d'atteindre le processus a_0) s'applique directement au modèle PRISM du listing 7.1 et donc aux Frappes de Processus de la figure 7.1.

7.3 La Simulation des Frappes de Processus

Dans cette section, nous montrons et prouvons l'instanciation de la machine abstraite générique présentée dans le chapitre 6 pour la simulation des Frappes de Processus munies de paramètres stochastiques et temporels via l'absorption de stochasticité (c.-à-d., distribuées par Erlang).

Une approche pourrait se baser sur la traduction des Frappes de Processus distribuées par Erlang en π-calcul stochastique markovien présentée dans la section précédente (définition 7.1) et utiliser

```
ctmc
//h1 = a 0 -> a 0 1
//h2 = a 0 -> b 0 1
const double r_h1;
const int sa_h1;
const double r_h2;
const int sa_h2;

module proc_A
    a: [0..1] init 0;
    c_h1: [1..sa_h1] init 1;
    d_h2: bool init false;

    [] a=0 & c_h1<sa_h1 -> r_h1*sa_h1: (c_h1'=c_h1+1);
    [] a=0 & c_h1=sa_h1 -> r_h1*sa_h1: (a'=1)&(c_h1'=1)&(d_h2'=c_h2>1);
    [h2] a=0 -> (d_h2'=false);
endmodule

module proc_B
    b: [0..1] init 0;
    c_h2: [1..sa_h2] init 1;

    [h2] b=0 & d_h2 -> r_h2*sa_h2: (c_h2'=2);
    [h2] b=0 & !d_h2 &  c_h2<sa_h2 -> r_h2*sa_h2: (c_h2'=c_h2+1);
    [h2] b=0 & !d_h2 &  c_h2=sa_h2 -> r_h2*sa_h2: (b'=1) & (c_h2'=1);
endmodule
```

Listing 7.1 – Traduction des Frappes de Processus de la figure 7.1 en PRISM suivant la définition 7.2.

l'instanciation de la machine abstraite avec le π-calcul stochastique et une méthode de simulation markovienne. Cependant, une telle méthode ajoute un coût supplémentaire de traduction, et le modèle π-calcul obtenu peut engendrer un grand nombre de réactions au sein de la machine, sans toutefois présenter d'explosion combinatoire.

Nous exploitons ici la généricité de la machine abstraite afin de l'instancier directement avec les Frappes de Processus distribuées par Erlang. Nous nous basons sur la méthode de simulation non-markovienne instanciée dans la section 6.6 page 101 (la Méthode par Réaction Suivante) et définissons les fonctions nécessaires à l'instanciation des Frappes de Processus (définition 7.3). Nous définissons une espèce comme étant un processus, rendant les définitions des fonctions espèces et processus triviales (elle peuvent être assimilées à la fonction identité). Nous notons que la fonction processus retourne un état des Frappes de Processus. Enfin, une action $h = a_i \rightarrow b_j \Rsh b_k$ est transformée en une réaction $a_i + b_j \rightarrow a_i + b_k$, en lui attachant une distribution Erlang avec un taux r_h et un facteur d'absorption de stochasticité sa_h.

Définition 7.3 (Instanciation de la machine générique avec les Frappes de Processus)**.**

$$
\begin{aligned}
I ::= a_i \qquad & \text{Espèce processus} \\
\mathsf{espèces}(s) &\triangleq [a_i \mid a_i \in s] \\
\mathsf{processus}(\overline{I}) &\triangleq \{a \mapsto a_i \mid a_i \in \overline{I}\} \\
\mathsf{réactions}(a_i, \widetilde{I}) &\triangleq [([\mathsf{frappeur}(h), \mathsf{cible}(h)], \mathsf{Erlang}(r_h, sa_h), [\mathsf{frappeur}(h), \mathsf{bond}(h)]) \mid h \in \mathcal{H} \\
&\qquad \wedge (a_i = \mathsf{frappeur}(h) \wedge \mathsf{cible}(h) \in \widetilde{I} \vee a_i = \mathsf{cible}(h) \wedge \mathsf{frappeur}(h) \in \widetilde{I})]
\end{aligned}
$$

Nous remarquons que le procédé de renommage utilisé par l'instanciation de la méthode de simulation non-markovienne est ici inutile car, à tout instant, au plus un individu de chaque espèce est présent.

Nous montrons la correction de cette instanciation à l'aide de la généricité des théorèmes 6.1, 6.2 et 6.3. Il est ainsi suffisant de montrer les propositions suivantes :

- proposition 6.2 page 99 ($\forall P, \overline{I}$: $\mathsf{processus}(\mathsf{espèces}(P)) = P$ et $\mathsf{espèces}(\mathsf{processus}(\overline{I})) = \overline{I}$) : nos fonctions processus et espèces peuvent être considérés comme des fonctions identité rendant cette proposition triviale.
- proposition 6.3 page 99 ($\forall S, S', a$: $\mathsf{processus}(S) \xrightarrow{a} \mathsf{processus}(S')$ si et seulement si $a = \sum_{\{O \in \mathsf{ens\text{-}réactions}(S,S')\}} \mathsf{propension}(O, S)$ et $a > 0$) : il existe une bijection entre les actions jouables dans un état $\mathsf{processus}(S)$ et les réactions engendrées dans la machine. De plus, la population d'une espèce étant au plus un, nous avons forcément $a = 1$ si une transition est possible.
- proposition 6.4 page 104 ($\forall T_1, T_2, T_1', T_2'$ si $T_1 \xrightarrow{a,(\overline{I},F,\overline{I'})} T_2$, $T_1 \xrightarrow{1,(\overline{J_1},F_1,\overline{J_1'})} T_1'$, $T_2 \xrightarrow{1,(\overline{J_2},F_2,\overline{J_2'})} T_2'$, avec $\mathsf{NM}(F_1)$ et $\mathsf{NM}(F_2)$, et si $[\![T_1]\!] \xrightarrow{w} [\![T_1']\!]$ et $[\![T_2]\!] \xrightarrow{w} [\![T_2']\!]$, alors $\overline{I} \cap \overline{J}_1 = \emptyset$) : nous remarquons que lors de l'application d'une réaction, d'après la fonction suite' (définition 6.15), seule la cible de l'action des Frappes de Processus est enlevée et seul le bond est ajouté : en effet, le frappeur appartenant à la fois aux réactants et aux produits de la réaction, il est ignoré par le procédé de réduction de la machine. Ainsi, si $\overline{I} \cap \overline{J}_1 \neq \emptyset$, alors la transition $T_2 \xrightarrow{1,(\overline{J_2},F_2,\overline{J_2'})} T_2'$ est impossible.

7.4 Discussion

Dans ce chapitre, nous avons appliqué aux Frappes de Processus les résultats introduits dans les chapitres 5 et 6 concernant l'introduction du temps et de l'aléatoire dans les modèles des systèmes complexes.

Ainsi, la durée d'une action en Frappes de Processus est paramétrée par un taux d'utilisation r et un facteur d'absorption de stochasticité sa, et est une variable aléatoire suivant la distribution de la somme de sa variables aléatoires exponentielles avec un taux $r.sa$. La valeur moyenne de cette durée est de r^{-1} unités de temps avec une variance de $r^{-2}sa^{-1}$. Nous avons montré dans le chapitre 5 qu'il est possible de traduire en ces paramètres stochastiques un intervalle de temps au cours duquel l'action peut se produire, avec un degré de confiance fixé.

La vérification formelle des Frappes de Processus distribuées par Erlang, ainsi que leur traduction en π-calcul stochastique markovien, découle des résultats du chapitre 5, que nous avons ici simplifié pour le cas précis des Frappes de Processus. Enfin, nous avons exploité la généricité de la machine

abstraite présentée dans le chapitre 6 en démontrant son instanciation pour la simulation non-markovienne des Frappes de Processus.

Nous remarquons que l'introduction de paramètres temporels dans les Frappes de Processus peut permettre d'approximer les Frappes de Processus avec Priorités, dont l'intérêt pour l'encodage des formalismes classiques a été montré dans le chapitre 4. En effet, en attachant aux actions de forte priorité un taux d'utilisation élevé et aux actions de faibles priorité un taux d'utilisation faible, nous observons un comportement discret proche de celui observé avec l'utilisation de priorités.

La vérification formelle quantitative des modèles obtenus a été abordée dans le chapitre 5 et se heurte à une explosion combinatoire lorsque les facteurs d'absorption de stochasticité sont élevés. Dans la partie suivante, nous présentons des méthodes de vérification formelle à grande échelle par analyse statique des Frappes de Processus pour des propriétés discrètes. L'extension de ces analyses aux propriétés quantitatives (comme la probabilité d'atteindre un processus dans un intervalle de temps donné) est discutée dans le chapitre 12 traitant des perspectives de cette thèse.

Troisième partie

Analyse Statique des Frappes de Processus

En exploitant la simplicité des Frappes de Processus, nous développons dans cette partie deux analyses statiques permettant respectivement la caractérisation complète des points fixes des dynamiques et l'approximation supérieure et inférieure de propriétés d'atteignabilité.
Le chapitre 8 traite dans un premier temps de l'extraction de points fixes topologiques des RRB : ces points fixes sont communs à tous les Réseaux Booléens partageant le même graphe des interactions. L'extraction des points fixes des Frappes de Processus est ensuite effectuée via une analyse topologique d'une représentation complémentaire dite de Graphe Sans-Frappe.
Le chapitre 9 établit une interprétation abstraite des scénarios des Frappes de Processus permettant un calcul efficace d'approximations supérieures et inférieures de propriétés d'atteignabilité successive de processus. Les processus requis pour la satisfaction d'une telle propriété peuvent également être calculés. Notre méthode possède une complexité théorique faible (polynomiale selon le nombre de processus, exponentielle selon le nombre de processus au sein d'une seule sorte), au prix d'une potentielle absence de conclusion, garantissant la possibilité d'analyser de très grands modèles.

Chapitre **8**

Analyse Statique des Points Fixes

Dans le cadre des modélisations des RRB, les points fixes d'un modèle correspondent aux états stables (stationnaires) du système et sont d'un grand intérêt pour caractériser ses comportements. Nous commençons ce chapitre par la caractérisation des *points fixes topologiques* dans le cadre des modélisations des RRB en Réseaux Booléens : un tel point est fixe pour tous les Réseaux Booléens possédant le même graphe des interactions. Nous obtenons que si un graphe des interactions G ne possède aucun cycle (non-orienté) négatif, et que tous ses composants possèdent au moins un prédécesseur dont au plus un non-signé, alors G engendre exactement deux points fixes topologiques (leur caractérisation est alors immédiate) ; dans le cas contraire, G ne possède aucun point fixe topologique.
Nous traitons enfin l'extraction complète des points fixes de Frappes de Processus données via une analyse topologique du modèle. L'énumération des points fixes est alors réduite à l'énumération des n-cliques dans un graphe n-parti. Appliquée à la recherche des points fixes dans la dynamique généralisée d'un RRB, nous obtenons des résultats similaires aux conditions sur les points fixes topologiques des Réseaux Booléens.

8.1 Préliminaires

L'étude des points fixes (et plus généralement des bassins d'attractions) des dynamiques apporte une compréhension importante des différents comportements d'un RRB (Wuensche, 1998).

La recherche des points fixes étant généralement un problème NP-complet (Barrett, Hunt III, Marathe, Ravi, Rosenkrantz, Stearns & Tosic, 2001), de nombreuses méthodes d'analyses statiques ont été développées afin d'établir des bornes sur le nombre de points fixes possibles dans une dynamique discrète à partir du graphe des interactions (Richard, 2009) ; ou encore leur caractérisation exhaustive à partir de la spécification complète des paramètres discrets de René Thomas (Naldi et coll., 2007) (voir également le chapitre 2).

Dans le but de compléter les analyses portant sur le lien entre la structure du graphe des interactions d'un Réseau Booléen et ses points fixes, nous introduisons la notion de *point fixe topologique* : étant donné un Réseau Booléen F, nous notons $G(F)$ son graphe des interactions (défini formellement dans la section suivante ; voir également la section 2.2 page 10). Le point x est un point fixe topologique si et seulement si $F'(x) = x$ pour tout Réseau Booléen F' tel que $G(F) = G(F')$. Nous obtenons alors une condition nécessaire et suffisante pour l'existence d'un

point fixe topologique d'un graphe des interactions G : si G ne possède aucun cycle (non-orienté) négatif, et si tous ses composants possèdent au moins un prédécesseur dont au plus un non-signé, alors G possède exactement deux points fixes topologiques x et x' ; sinon, G ne possède aucun point fixe topologique. De plus nous obtenons une caractérisation immédiate de ces points fixes topologiques, et notamment la propriété que x est l'inverse de x' (si une coordonnée de x vaut 0 la correspondante vaut 1 dans x', et vice-versa).

Enfin, nous abordons la caractérisation des points fixes dans les Frappes de Processus. Il apparaît que les points fixes des Frappes de Processus composées de n sortes sont exactement les n-cliques dans une représentation complémentaire des Frappes de Processus appelée Graphe Sans-Frappe (deux processus sont mis en relation si et seulement si aucun d'eux ne frappe l'autre). L'extraction des points fixes des Frappes de Processus revient alors à énumérer les n-cliques d'un graphe n-parti, pouvant être implémenté de manière efficace.

La caractérisation des points fixes des Frappes de Processus permet l'étude des points fixes sur des RRB dont les paramètres discrets n'ont été spécifiés que partiellement : ces points fixes sont alors communs à toutes les dynamiques issues d'un raffinement de cette spécification partielle.

Appliquée à la caractérisation des points fixes dans la dynamique généralisée des Réseaux Discrets, nous obtenons un résultat similaire à celui obtenu pour les points fixes topologiques, en se restreignant toutefois aux graphes sans arc non-signé. La preuve obtenue par les Frappes de Processus est directe et intuitive, soulignant un apport des Frappes de Processus comme outil d'analyse des systèmes dynamiques. En fait, c'est cette dernière caractérisation qui a inspiré le résultat sur les points fixes topologiques obtenu dans le cadre des Réseaux Booléens.

Nous terminons par la caractérisation des jardins d'Éden des Frappes de Processus : un état est un jardin d'Éden si il n'est accessible par aucun autre état. Nous définissons alors les Frappes de Processus Inverses dans lesquelles un jardin d'Éden correspond à un et un seul point fixe.

Ce chapitre est structuré comme suit. La section 8.2 présente la caractérisation des points fixes topologiques d'un graphe des interactions des Réseaux Booléens (travail effectué en collaboration avec Adrien Richard, I3S & CNRS, Sophia Antipolis, France). La section 8.3 établit une analyse statique des Frappes de Processus en général permettant la caractérisation complète de ses points fixes et jardins d'Éden. Enfin, la section 8.4 conclut et résume les contributions de ce chapitre.

Les résultats présentés dans la section 8.2 sont publiés dans (Paulevé & Richard, 2010) ; les résultats présentés dans la sous-section 8.3.1 sont publiés dans (Paulevé, Magnin & Roux, 2011*b*).

8.2 Points Fixes Topologiques dans les Réseaux Booléens ‡

Nous nous intéressons aux relations entre les états stables et la topologie des Réseaux Booléens. D'une part, la dynamique d'un réseau booléen entre n composants est habituellement décrite pour les itérations successives d'une application F de $\{0,1\}^n$ vers lui-même. Les états stables du réseau correspondent alors aux points fixes de F. D'autre part, la topologie du réseau est généralement décrite par un graphe des interactions G qui peut être déduit de F. Les nœuds correspondent aux composants du réseau, et les arcs, orientés et signés, décrivent les relations de causalité en terme d'activation et d'inhibition entre les composants.

Les réseaux booléens ont été appliqués à de nombreux domaines, et particulièrement pour la

‡. Travail effectué en collaboration avec Adrien Richard (I3S & CNRS, Nice, France).

modélisation des réseaux biologiques (voir par exemple le travail de Kauffman (Kauffman, 1969; Kauffman, 1993) et Thomas (Thomas, 1973; Thomas & d'Ari, 1990)). Les relations entre G et les points fixes de F sont d'un intérêt particulier dans ce contexte : les points fixes ont souvent une signification biologique (p. ex. des motifs stables d'expression génique correspondant à des fonctions cellulaires particulières) (Kauffman, 1993; Thomas & d'Ari, 1990; Mendoza, Thieffry & Alvarez-Buylla, 1999), et les premières informations fiables obtenues quand les biologistes étudient un réseau biologique sont souvent représentées en terme de graphe des interactions (de Jong, 2002).

Dans cette section, nous étudions les points fixes de F qui dépendent seulement du graphe des interactions G : nous parlons alors de *points fixes topologiques de* F. Les points fixes topologiques de F peuvent être vus comme des points fixes « robustes » dans le sens où ils restent fixes même après une perturbation de F qui n'affecte pas le graphe des interactions G du réseau. Notre résultat principal est la caractérisation du nombre de points fixes topologiques de F en fonction de la structure de G. Ceci utilise et généralise un théorème de Aracena, Demongeot et Goles (Aracena, 2008; Aracena et coll., 2004).

8.2.1 Définitions

Nous présentons ici les définitions utilisées dans la suite de cette section. La définition 8.1 présente les graphes des interactions de manière similaire au chapitre 3, mais où nous autorisons la présence d'arcs non-signés entre les composants ; les notions de chaînes et de composantes connexes sont également formalisées dans cette définition. La définition 8.3 établit l'inférence du graphe des interactions correspondant à une application booléenne donnée. La définition 8.4 impose une condition sur les graphes des interactions admis dans la suite de cette section, à savoir l'existence d'une application booléenne lui correspondant. La définition 8.5 formalise la notion de point fixe topologique, sur laquelle les résultats de cette section s'appuient : un point fixe d'une application booléenne est dit topologique si il est point fixe pour toute application booléenne possédant le même graphe des interactions ; dans un tel cas, nous parlons également d'un point fixe topologique d'un graphe des interactions donné. Enfin, la définition 8.6 présente les notations booléennes utilisées dans la suite de cette section.

Définition 8.1 (Graphe des interactions)**.** Un *graphe des interactions* $\mathcal{G}$ est un graphe orienté sur $\{1, \ldots, n\}$ où chaque arc ji (de j vers i) est soit *positif*, *négatif*, ou *non-signé*. L'ensemble des arcs positifs, négatifs et non-signés de $\mathcal{G}$ est dénoté par $\mathcal{G}^+$, $\mathcal{G}^-$, et $\mathcal{G}^0$, respectivement. L'ensemble des *prédécesseurs positifs (resp. négatifs, non-signés)* d'un sommet i est $\mathcal{G}_i^+ = \{j \,|\, ji \in \mathcal{G}^+\}$ (resp. $\mathcal{G}_i^- = \{j \,|\, ji \in \mathcal{G}^-\}$, $\mathcal{G}_i^0 = \{j \,|\, ji \in \mathcal{G}^0\}$). L'ensemble des *prédécesseurs signés* de i est $\mathcal{G}_i^+ \cup \mathcal{G}_i^-$. Nous dénotons par $\tilde{\mathcal{G}}$ le graphe des interactions que nous obtenons en supprimant les arcs non-signés de $\mathcal{G}$, c.-à-d. $\tilde{\mathcal{G}}^+ = \mathcal{G}^+$, $\tilde{\mathcal{G}}^- = \mathcal{G}^-$ et $\tilde{\mathcal{G}}^0 = \emptyset$.

Définition 8.2 (Propriétés d'un graphe des interactions $\mathcal{G}$)**.** Une *chaîne* (chemin non-orienté) de $\mathcal{G}$ est une séquence de $p \geq 1$ sommets $i_0 i_1 \ldots i_p$ telle que $i_k i_{k+1}$ ou $i_{k+1} i_k$ est un arc de $\mathcal{G}$, $0 \leq k < p$. Une telle chaîne *relie* i_0 et i_p, et est un *cycle* si $i_0 = i_p$. Une chaîne sans arc non-signé est *signé* ; elle est dite *positif* si il contient un nombre pair d'arcs négatifs, elle est dite *négatif* dans le cas contraire. $\mathcal{G}$ est *connexe* si il existe une chaîne reliant chaque paire de sommets disctincts. Une *composante connexe* de $\mathcal{G}$ est un sous-ensemble maximal C de sommets avec la propriété que $\mathcal{G}$ possède une chaîne reliant chaque paire de sommets distincts pris dans C.

Définition 8.3 (Graphe $G(F)$ des interactions de F)**.** Considérons une application booléenne

$$F = (f_1, \ldots, f_n) : \{0,1\}^n \to \{0,1\}^n, \qquad x = (x_1, \ldots, x_n) \mapsto F(x) = (f_1(x), \ldots, f_n(x)).$$

La *dérivée discrète* de f_i selon la variable x_j est l'application $f_{ij} : \{0,1\}^n \to \{-1, 0, 1\}$ définie pour tout $i, j \in \{1, \ldots, n\}$ par :

$$f_{ij}(x) = f_i(x_1, \ldots, x_{j-1}, 1, x_{j+1}, \ldots, x_n) - f_i(x_1, \ldots, x_{j-1}, 0, x_{j+1}, \ldots, x_n) \ .$$

Le *graphe des interactions de F* est le graphe des interactions $G(F)$ défini par : pour $i, j = 1, \ldots, n$, il existe un arc ji si $f_{ij} \neq 0$, et cet arc est positif si $f_{ij} \geq 0$, négatif si $f_{ij} \leq 0$, et non-signé sinon (c.-à-d. si f_{ij} est des fois positif, des fois négatif). Nous notons que $f_{ij} \neq 0$ si et seulement si la valeur de f_i dépend de la valeur de x_j.

Définition 8.4 (Graphe des interactions admissible)**.** Un graphe des interactions $\mathcal{G}$ est *admissible* si il existe une application $F : \{0,1\}^n \to \{0,1\}^n$ telle que $G(F) = \mathcal{G}$.

Définition 8.5 (Point fixe topologique)**.** Étant donnée une application booléenne $F : \{0,1\}^n \to \{0,1\}^n$, un point $x \in \{0,1\}^n$ est un *point fixe topologique* de F si $F'(x) = x$ pour toute application $F' : \{0,1\}^n \to \{0,1\}^n$ telle que $G(F') = G(F)$.
Étant donné un graphe des interactions admissible $\mathcal{G}$, $x \in \{0,1\}^n$ est un *point fixe topologique de $\mathcal{G}$* si $F(x) = x$ pour toute application $F : \{0,1\}^n \to \{0,1\}^n$ telle que $G(F) = \mathcal{G}$. Ainsi, x est un point fixe topologique de F si et seulement si x est un point fixe topologique de $G(F)$.

Définition 8.6 (Opérations booléennes)**.** Nous posons $\overline{0} = \sigma^-(0) = \sigma^+(1) = 1$; $\overline{1} = \sigma^-(1) = \sigma^+(0) = 0$; et $\overline{x} = (\overline{x_1}, \overline{x_2}, \ldots, \overline{x_n})$. Une somme de variables booléennes est toujours une somme booléenne ($1 + 1 = 1$), et la somme modulo 2 est dénotée par $\oplus$ ($1 \oplus 1 = 0$). Le produit de variables booléennes est le produit usuel ($0.1 = 0$, $1.1 = 1$, etc.). Par convention, le produit vide est 1 et la somme vide est 0.

8.2.2 Résultats

Soit F une application de $\{0,1\}^n$ vers lui-même telle que $G(F)$ ne possède aucun arc non-signé. Nous disons que $G(F)$ a la propriété $\mathcal{P}$ si $G(F)$ est connexe, si chaque sommet de $G(F)$ a un prédécesseur, et si $G(F)$ ne possède aucun cycle (non-orienté) négatif. Aracena, Demongeot et Goles (Aracena et coll., 2004; Aracena, 2008) ont prouvé un théorème qui peut s'exprimer dans nos notations de la façon suivante :

Si $G(F)$ a la propriété $\mathcal{P}$, alors il existe $x \in \{0,1\}^n$ tel que x et $\overline{x}$ sont des points fixes de F.

Une conséquence simple non mentionnée dans la construction de leur preuve (que nous utilisons et étendons ici) est que x et $\overline{x}$ sont en fait des points fixes topologiques de F, et qu'aucun autre point fixe topologique n'existe. Ainsi, étant donné un graphe des interactions $\mathcal{G}$ sans arc non-signé (un tel graphe est toujours admissible, voir la remarque 8.1 ci-dessous), nous obtenons :

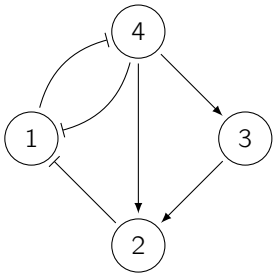


Figure 8.1 – Exemple de graphe des interactions possédant deux points fixes topologiques : $x = \langle 0, 1, 1, 1\rangle$ et $\overline{x} = \langle 1, 0, 0, 0\rangle$.

Si $\mathcal{G}$ a la propriété $\mathcal{P}$, alors $\mathcal{G}$ a exactement deux points fixes topologiques, et ils sont de la forme $x, \overline{x}$.

Dans la suite, nous montrons que la réciproque de cette version plus forte du théorème de Aracena et coll. est vraie :

Si $\mathcal{G}$ possède exactement deux points fixes topologiques, alors ils sont de la forme $x, \overline{x}$, et $\mathcal{G}$ a la propriété $\mathcal{P}$.

Ces deux faits sont contenus dans le théorème suivant qui donne une caractérisation du nombre de points fixes topologiques pour n'importe quel graphe des interactions admissible :

Théorème 8.1. *Soit $\mathcal{G}$ un graphe des interactions admissible. Soit p le nombre de composantes connexes de $\tilde{\mathcal{G}}$.*

1. *Si chaque sommet de $\mathcal{G}$ possède un prédécesseur et au plus un prédécesseur non-signé, et si $\mathcal{G}$ ne possède aucun cycle (non-orienté) négatif, alors $\mathcal{G}$ possède exactement 2^p points fixes topologiques. Dans le cas contraire, $\mathcal{G}$ ne contient aucun point fixe topologique.*
2. *Si x est un point fixe topologique de $\mathcal{G}$, alors $\overline{x}$ est un point fixe topologique de $\mathcal{G}$.*

Nous soulignons le fait que si $\mathcal{G}$ possède m arcs, le nombre de composantes connexes de $\tilde{\mathcal{G}}$ peut être calculé en $\mathcal{O}(n + m)$, et la présence d'un cycle (non-orienté) négatif dans $\mathcal{G}$ peut être vérifiée avec la même complexité. Ainsi, d'après le théorème 8.1, le nombre de points fixes topologiques de $\mathcal{G}$ peut être calculé en $\mathcal{O}(n + m)$. La preuve montre également que ces points fixes topologiques peuvent être complètement caractérisés en $\mathcal{O}(n + m)$ étapes.

La figure 8.1 illustre l'application du théorème 8.1 sur un graphe des interactions possédant deux points fixes topologiques.

8.2.3 Preuve du Théorème 8.1

Nous commençons avec un lemme basique sur les arcs non-signés :

Lemme 8.1. *Soit $\mathcal{G}$ un graphe des interactions. Si $\mathcal{G}$ est admissible, alors chaque sommet de $\mathcal{G}$ possédant un unique prédécesseur non-signé possède au moins deux prédécesseurs signés.*

Démonstration. Supposons que $\mathcal{G}$ soit admissible, et que $\mathcal{G}$ possède un sommet i avec un prédécesseur

unique non-signé k. Alors, pour chaque $F : \{0,1\}^n \to \{0,1\}^n$ telle que $G(F) = \mathcal{G}$, il existe $x, y \in \{0,1\}^n$ tel que :

$$\begin{array}{rclcrcl} 0 & = & f_i(x_1,\ldots,x_{k-1},0,x_{k+1},\ldots,x_n) & < & f_i(x_1,\ldots,x_{k-1},1,x_{k+1},\ldots,x_n) & = & 1\ , \\ 1 & = & f_i(y_1,\ldots,y_{k-1},0,y_{k+1},\ldots,y_n) & > & f_i(y_1,\ldots,y_{k-1},1,y_{k+1},\ldots,y_n) & = & 0\ . \end{array}$$

Ainsi, la valeur de f_i dépend de la valeur d'au moins une variable x_j, $j \neq k$, et il est facile de voir que si f_i dépend seulement de x_k et x_j, alors j est un prédécesseur non-signé de i, ce qui contredit notre hypothèse. □

Remarque 8.1. Cette condition nécessaire pour l'admissibilité est également suffisante. En effet, si chaque sommet de $\mathcal{G}$ avec un unique prédécesseur non-signé possède au moins deux prédécesseurs signés, alors il est facile de voir que $\mathcal{G}$ est le graphe des interactions de l'application F définie par :

1. pour chaque i sans prédécesseur non-signé, $f_i(x) = \sum_{j\in\mathcal{G}_i^+} x_j + \sum_{j\in\mathcal{G}_i^-} \overline{x_j}$;
2. pour chaque i avec un unique prédécesseur non-signé k, $f_i(x) = x_k\, \sigma^{s_1}(x_{l_1}) + \overline{x_k}\, \sigma^{s_2}(x_{l_2}) + \sum_{j\in\mathcal{G}_i^+\setminus\{l_1,l_2\}} + \sum_{j\in\mathcal{G}_i^-\setminus\{l_1,l_2\}} \overline{x_j}$, où $l_1 \in \mathcal{G}_i^{s_1}$ et $l_2 \in \mathcal{G}_i^{s_2}$ sont deux prédécesseurs signés de i ;
3. pour chaque i avec $p \geq 2$ prédécesseurs non-signés $k_1,\ldots,k_p$, $f_i(x) = \sum_{1\leq q<p}(x_{k_q} \oplus x_{k_{q+1}}) + \sum_{j\in\mathcal{G}_i^+} x_j + \sum_{j\in\mathcal{G}_i^-} \overline{x_j}$.

Nous posons désormais le lemme principal :

Lemme 8.2. *Soit $\mathcal{G}$ un graphe des interactions admissible. Un point $\alpha \in \{0,1\}^n$ est un point fixe topologique de $\mathcal{G}$ si et seulement si (1) chaque sommet de $\mathcal{G}$ possède un prédécesseur et au plus un prédécesseur non-signé ; et (2) $\alpha_j = \alpha_i$ pour tout $ji \in \mathcal{G}^+$, et $\alpha_j \neq \alpha_i$ pour tout $ji \in \mathcal{G}^-$.*

Démonstration. (*Condition suffisante*) Soit $F : \{0,1\}^n \to \{0,1\}^n$ telle que $G(F) = \mathcal{G}$, on montre que α est un point fixe de F. Supposons, par contradiction, qu'il existe un sommet i tel que $f_i(\alpha) \neq \alpha_i$. Si i ne possède aucun prédécesseur non-signé, nous posons $X = \{x \mid f_i(x) = \alpha_i\}$; i possédant un prédécesseur, f_i n'est pas constante, donc X n'est pas vide. Si i possède un unique prédécesseur non-signé k, nous posons $X = \{x \mid f_i(x) = \alpha_i, x_k = \alpha_k\}$; k étant un prédécesseur non-signé de i, pour tout $a, b \in \{0,1\}$, il existe x tel que $f_i(x) = a$ et $x_k = b$, donc X n'est pas vide. Soit x un point de X minimisant la distance de Hamming $d(x,\alpha)$ valant le nombre de $j \in \{1,\ldots,n\}$ tel que $x_j \neq \alpha_j$. Comme $f_i(x) \neq f_i(\alpha)$, il existe j tel que $x_j \neq \alpha_j$, et, par construction, j est un prédécesseur signé de i. Considérons le point y tel que $y_j = \overline{x_j} = \alpha_j$ et $y_k = x_k$ pour chaque sommet $k \neq j$. Nous avons $d(y,\alpha) = d(x,\alpha) - 1$. Donc $y \notin X$ et nous déduisons que $f_i(y) \neq f_i(x) = \alpha_i$. Donc $f_{ij}(x) > 0$ si $\alpha_i = x_j$, et $f_{ij}(x) < 0$ si $\alpha_i \neq x_j$. Comme j est un prédécesseur signé de i, et comme $x_j \neq \alpha_j$, nous déduisons que soit $ji \in \mathcal{G}^+$ et $\alpha_i \neq \alpha_j$, soit $ji \in \mathcal{G}^-$ et $\alpha_i = \alpha_j$, ce qui est une contradiction. Ainsi $f_i(\alpha) = \alpha_i$ pour tout sommet i.

(*Condition nécessaire*) Supposons que α est un point fixe topologique de $\mathcal{G}$. Nous montrons que les conditions (1) et (2) sont vérifiées pour tout sommet i. Admettons que $h_i : \{0,1\}^n \to \{0,1\}$ est admissible si il existe $F : \{0,1\}^n \to \{0,1\}^n$ tel que $f_i = h_i$ et $G(F) = \mathcal{G}$. Donc si h_i est admissible, $h_i(\alpha) = \alpha_i$.
Supposons que i possède $p \geq 2$ prédécesseurs non-signés $k_1,\ldots,k_p$. Considérons les quatre applications de $\{0,1\}^n$ vers $\{0,1\}$:

$$\begin{aligned} h_i^1(x) &= \textstyle\prod_{1\leq q<p}(x_{k_q} \oplus x_{k_{q+1}}) \cdot \prod_{j\in\mathcal{G}_i^+} x_j \cdot \prod_{j\in\mathcal{G}_i^-} \overline{x_j}\ , \\ h_i^2(x) &= \textstyle\prod_{1\leq q<p}\overline{(x_{k_q} \oplus x_{k_{q+1}})} \cdot \prod_{j\in\mathcal{G}_i^+} x_j \cdot \prod_{j\in\mathcal{G}_i^-} \overline{x_j}\ , \\ h_i^3(x) &= \textstyle\sum_{1\leq q<p}(x_{k_q} \oplus x_{k_{q+1}}) + \sum_{j\in\mathcal{G}_i^+} x_j + \sum_{j\in\mathcal{G}_i^-} \overline{x_j}\ , \\ h_i^4(x) &= \textstyle\sum_{1\leq q<p}\overline{(x_{k_q} \oplus x_{k_{q+1}})} + \sum_{j\in\mathcal{G}_i^+} x_j + \sum_{j\in\mathcal{G}_i^-} \overline{x_j}\ . \end{aligned}$$

Comme h_i^r est admissible pour $r = 1, 2, 3, 4$, nous obtenons $h_i^r(\alpha) = \alpha_i$ pour $r = 1, 2, 3, 4$. Mais si $h_i^1(\alpha) = 1$ alors $h_i^2(\alpha) = 0$, et si $h_i^3(\alpha) = 0$ alors $h_i^4(\alpha) = 1$. Nous déduisons que $h_i^1(\alpha) \neq h_i^2(\alpha)$ ou $h_i^3(\alpha) \neq h_i^4(\alpha)$, ce qui est une contradiction. Donc i possède au plus un prédécesseur non-signé. Nous avons donc les des cas suivants :

1. *le sommet i ne possède aucun prédécesseur non-signé.* Considérons les deux applications suivantes de $\{0,1\}^n$ vers $\{0,1\}$:

$$h_i^1(x) = \prod_{j \in \mathcal{G}_i^+} x_j \cdot \prod_{j \in \mathcal{G}_i^-} \overline{x_j}, \qquad h_i^2(x) = \sum_{j \in \mathcal{G}_i^+} x_j + \sum_{j \in \mathcal{G}_i^-} \overline{x_j}.$$

   Comme h_i^1 et h_i^2 sont admissibles, nous obtenons $h_i^1(\alpha) = h_i^2(\alpha) = \alpha_i$. Si i n'a pas de prédécesseur, alors $h_i^1(\alpha) = 1$ et $h_i^2(\alpha) = 0$, ce qui est une contradiction. Donc i possède un prédécesseur et satisfait la condition (1). Supposons que $ji \in \mathcal{G}^+$. Si $\alpha_i = 1$ alors $h_i^1(\alpha) = 1$ donc $\alpha_j = 1$, et si $\alpha_i = 0$ alors $h_i^2(\alpha) = 0$ donc $\alpha_j = 0$. Dans les deux cas, $\alpha_j = \alpha_i$. Nous prouvons similairement que $\alpha_j \neq \alpha_i$ pour chaque $ji \in \mathcal{G}^-$; donc i satisfait la condition (2).

2. *le sommet i possède un unique prédécesseur non-signé k.* Il est suffisant de montrer que la condition (2) est vérifiée. Par le lemme 8.1, i possède au moins deux prédécesseurs signés, disons $l_1 \in \mathcal{G}_i^{s_1}$ et $l_2 \in \mathcal{G}_i^{s_2}$. Considérons alors les quatre applications suivantes :

$$\begin{aligned}
h_i^1(x) &= x_k\, \sigma^{s_1}(x_{l_1}) + \overline{x_k}\, \sigma^{s_2}(x_{l_2}) + \textstyle\sum_{j \in \mathcal{G}_i^+ \setminus \{l_1, l_2\}} x_j + \sum_{j \in \mathcal{G}_i^- \setminus \{l_1, l_2\}} \overline{x_j} \ , \\
h_i^2(x) &= x_k\, \sigma^{s_2}(x_{l_2}) + \overline{x_k}\, \sigma^{s_1}(x_{l_1}) + \textstyle\sum_{j \in \mathcal{G}_i^+ \setminus \{l_1, l_2\}} x_j + \sum_{j \in \mathcal{G}_i^- \setminus \{l_1, l_2\}} \overline{x_j} \ , \\
h_i^3(x) &= (x_k + \sigma^{s_1}(x_{l_1}))(\overline{x_k} + \sigma^{s_2}(x_{l_2})) \textstyle\prod_{j \in \mathcal{G}_i^+ \setminus \{l_1, l_2\}} x_j \prod_{j \in \mathcal{G}_i^- \setminus \{l_1, l_2\}} \overline{x_j} \ , \\
h_i^4(x) &= (x_k + \sigma^{s_2}(x_{l_2}))(\overline{x_k} + \sigma^{s_1}(x_{l_1})) \textstyle\prod_{j \in \mathcal{G}_i^+ \setminus \{l_1, l_2\}} x_j \prod_{j \in \mathcal{G}_i^- \setminus \{l_1, l_2\}} \overline{x_j} \ .
\end{aligned}$$

   Il est facile de voir que h_i^r est admissible pour $r = 1, 2, 3, 4$. Donc $h_i^r(\alpha) = \alpha_i$ pour $r = 1, 2, 3, 4$. Pour tout $j \in (\mathcal{G}_i^+ \cup \mathcal{G}_i^-) \setminus \{l_1, l_2\}$, nous prouvons, comme dans le premier cas, que $\alpha_j = \alpha_i$ si $ji \in \mathcal{G}^+$ et $\alpha_j \neq \alpha_i$ si $ji \in \mathcal{G}^-$. Alors, si $\alpha_i = 0$ nous avons $h_i^1(\alpha) = h_i^2(\alpha) = 0$, et nous déduisons que $\sigma^{s_1}(\alpha_{l_1}) = \sigma^{s_2}(\alpha_{l_2}) = 0$. Si $\alpha_i = 1$ alors $h_i^3(\alpha) = h_i^4(\alpha) = 1$, et nous déduisons que $\sigma^{s_1}(\alpha_{l_1}) = \sigma^{s_2}(\alpha_{l_2}) = 1$. Donc $\sigma^{s_1}(\alpha_{l_1}) = \sigma^{s_2}(\alpha_{l_2}) = \alpha_i$ dans les deux cas, ainsi la condition (2) est vérifiée pour tout prédécesseur signé de i.

□

Remarque 8.2. La condition (2) est équivalente à la condition « toute chaîne de $\tilde{\mathcal{G}}$ reliant j et i est positive si $\alpha_j = \alpha_i$, et négative si $\alpha_j \neq \alpha_i$ ». En conséquence, si $\mathcal{G}$ possède un point fixe topologique, alors $\tilde{\mathcal{G}}$ et $\mathcal{G}$ n'ont aucun cycle (non-orienté) négatif.

Remarque 8.3. Nous déduisons du lemme 8.2 que si chaque sommet de $\mathcal{G}$ possède au moins un prédécesseur non-signé, alors α est un point fixe topologique de $\mathcal{G}$ si et seulement si α est un point fixe topologique de $\tilde{\mathcal{G}}$.

La seconde partie du théorème 8.1 est une conséquence immédiate du lemme 8.2. Afin de prouver la première partie du théorème, nous utilisons un dernier lemme.

Lemme 8.3. *Soit $\mathcal{G}$ un graphe des interactions admissible. Si $\tilde{\mathcal{G}}$ est connexe, si chaque sommet de $\mathcal{G}$ possède un prédécesseur et au plus un prédécesseur non-signé, et si $\mathcal{G}$ ne possède aucun cycle (non-orienté) négatif, alors $\mathcal{G}$ possède exactement deux points fixes topologiques.*

Démonstration. Pour chaque sommet $i \neq 1$, soit P_{1i} une chaîne de $\tilde{\mathcal{G}}$ reliant 1 et i ($\tilde{\mathcal{G}}$ est connexe). Soit $\alpha \in \{0,1\}^n$ défini par : $\alpha_1 = 0$, $\alpha_i = 0$ si P_{1i} est positif, et $\alpha_i = 1$ sinon ($2 \leq i \leq n$). Si $ji \in \mathcal{G}^+$ et $\alpha_j \neq \alpha_i$, alors, par définition, P_{1j} et P_{1i} ont des signes opposés. Donc ces chaînes,

avec l'arc positif ji, forment un cycle (non-orienté) négatif, ce qui est une contradiction. Nous montrons similairement que si $ji \in \mathcal{G}^-$ alors $\alpha_j \neq \alpha_i$. Ainsi, par le lemme 8.2, α et $\overline{\alpha}$ sont des points fixes topologiques de $\mathcal{G}$. Considérons un point $\beta \neq \alpha$ et $\beta \neq \overline{\alpha}$. Il existe alors i, j tel que $\beta_i = \alpha_i$ et $\beta_j \neq \alpha_j$. Soit P une chaîne de $\tilde{\mathcal{G}}$ reliant j et i. D'après la remarque 8.2, P est positif si et seulement si $\alpha_j = \alpha_i$. Donc P est positif si et seulement si $\beta_j \neq \beta_i$, et donc β n'est pas un point fixe topologique de $\mathcal{G}$. □

Supposons que $\tilde{\mathcal{G}}$ possède p composantes connexes, et supposons que $\mathcal{G}$ vérifie la propriété $\mathcal{P}'$ suivante : chaque sommet de $\mathcal{G}$ possède un prédécesseur et au plus un prédécesseur non-signé, et $\mathcal{G}$ ne possède aucun cycle (non-orienté) négatif. Alors, chaque composante connexe de $\tilde{\mathcal{G}}$ induit un graphe des interactions qui satisfait les conditions du lemme précédent, et qui a donc exactement deux points fixes topologiques. Il est maintenant clair que $\tilde{\mathcal{G}}$ possède exactement 2^p points fixes topologiques, et nous déduisons de la remarque 8.3 que $\mathcal{G}$ possède également 2^p points fixes topologiques. Si $\mathcal{G}$ ne satisfait pas la propriété $\mathcal{P}'$, d'après le lemme 8.2 et la remarque 8.2, $\mathcal{G}$ ne possède aucun point fixe topologique. Ceci termine la preuve du théorème 8.1.

8.3 Points Fixes et Jardins d'Éden des Frappes de Processus

Nous traitons maintenant de la découverte des points fixes et des jardins d'Éden dans le cadre des Frappes de Processus. Un point fixe de Frappes de Processus données est un état dans lequel aucune action n'est jouable (définition 8.7) ; un jardin d'Éden est un état inaccessible depuis tout autre état des Frappes de Processus (définition 8.8).

Définition 8.7 (Point Fixe des Frappes de Processus)**.** Soient $(\Sigma, L, \mathcal{H})$ des Frappes de Processus. Un état $s \in L$ est un *point fixe* si aucune action de $\mathcal{H}$ n'est jouable ($\forall h \in \mathcal{H}, \mathsf{frappeur}(h) \notin s \vee \mathsf{cible}(h) \notin s$).

Définition 8.8 (Jardin d'Éden des Frappes de Processus)**.** Soient $(\Sigma, L, \mathcal{H})$ des Frappes de Processus. Un état $s \in L$ est un *jardin d'Éden* si et seulement si il n'existe aucun état $s' \in L$ tel que $\exists h \in \mathcal{H}$ avec $s' \cdot h = s$; autrement dit, aucun état ne peut mener à s.

Dans la sous-section 8.3.1, nous montrons que l'ensemble des points fixes de Frappes de Processus données peut être complètement caractérisé via une analyse structurelle des actions. En utilisant cette propriété, nous apportons dans la sous-section 8.3.2 une preuve alternative des résultats sur la caractérisation des points fixes topologiques des Réseaux Booléens présentés dans la section 8.2. Enfin, nous montrons dans la sous-section 8.3.3 que les jardins d'Éden sont caractérisables de manière similaire aux points fixes des Frappes de Processus dont les frappes ont été inversées.

8.3.1 Caractérisation des Points Fixes

Étant données des Frappes de Processus, nous introduisons une représentation complémentaire à celle de l'hyper-graphe des frappes que nous appelons *Graphe Sans-Frappe* (définition 8.9 et figure 8.2). Le Graphe Sans-Frappe de Frappes de Processus $(\Sigma, L, \mathcal{H})$ met en relation deux processus

de sortes différentes si et seulement si aucun d'entre eux ne se frappe. Les sommets d'un Graphe Sans-Frappe peuvent être séparés en $n \leq |\Sigma|$ partitions, où aucun sommet d'une partition n'a de relation avec un autre sommet de la même partition. Un tel graphe est appelé n-parti (définition 8.10, propriété 8.1).

Définition 8.9 (Graphe Sans-Frappe)**.** Le *Graphe Sans-Frappe* des Frappes de Processus $(\Sigma, L, \mathcal{H})$ est un graphe (V, E) non-orienté où les sommets V et les arcs E sont définis de la manière suivante :

$$V = \{a_i \mid a \in \Sigma \wedge a_i \in L_a \wedge \nexists h \in \mathcal{H}, \mathsf{frappeur}(h) = a_i \wedge \mathsf{cible}(h) = a_i\} \ ,$$
$$E = \{\{a_i, b_j\} \subseteq V \mid \nexists h \in \mathcal{H}, \{\mathsf{frappeur}(h), \mathsf{cible}(h)\} = \{a_i, b_j\}\} \ .$$

Définition 8.10 (Graphe n-parti)**.** Un graphe (V, E) est *n-parti* si et seulement si V peut s'écrire sous la forme $V = V^1 \cup \cdots \cup V^n$, où les V^k sont tous disjoints ($\forall i, j \in \{1, \ldots, n\}, i \neq j, V^i \cap V^j = \emptyset$) et avec $\forall \{a_i, b_j\} \in E, \nexists k \in \{1, \ldots, n\}, \{a_i, b_j\} \subseteq V^k$.

Propriété 8.1. *Le Graphe Sans-Frappe (V, E) des Frappes de Processus $(\Sigma, L, \mathcal{H})$ est n-parti, $n \leq |\Sigma|$.*

Démonstration. Par construction de (V, E) (définition 8.9), il existe un partitionnement de V où chaque partition est un sous-ensemble des processus d'une et une seule sorte ; et à chaque sorte correspond au plus une partition. □

Étant données des Frappes de Processus $(\Sigma, L, \mathcal{H})$, nous démontrons que l'ensemble des $|\Sigma|$-cliques de leur Graphe Sans-Frappe est exactement l'ensemble de tous les points fixes des Frappes de Processus (théorème 8.2). Une n-clique d'un graphe est un sous-ensemble de n sommets tous en relation deux à deux entre eux (définition 8.11).

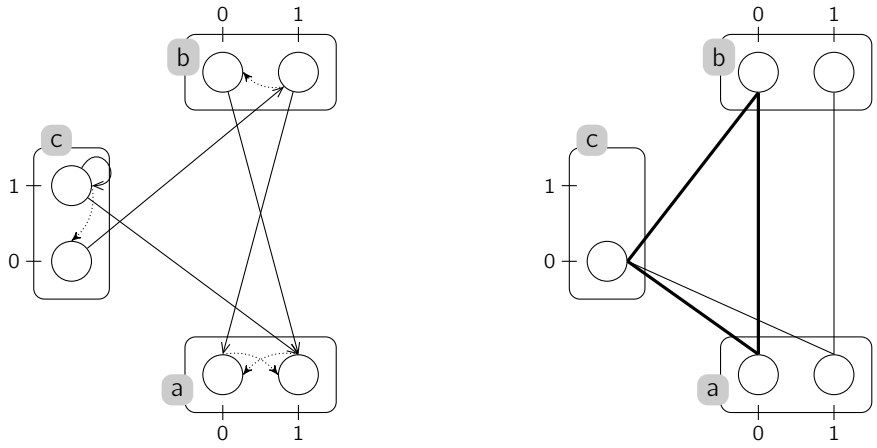
Définition 8.11 (n-clique)**.** Étant donné un graphe (V, E), $C \subseteq V$ est une *$|C|$-clique* du graphe si et seulement si $\forall \{a_i, b_j\} \subseteq C, \{a_i, b_j\} \in E$.

Théorème 8.2. *Soient $(\Sigma, L, \mathcal{H})$ des Frappes de Processus. Un état $s \in L$ est un point fixe des Frappes de Processus si et seulement si s correspond à une $|\Sigma|$-clique du Graphe Sans-Frappe associé.*

Démonstration. Par la définition 8.7, s est un point fixe si et seulement si aucune paire de processus présents dans s ne se frappe, ce qui est équivalent à la présence d'une $|\Sigma|$-clique dans le Graphe Sans-Frappe entre tous les processus de s. □

La figure 8.2 illustre le théorème 8.2 avec des Frappes de Processus possédant un seul point fixe. Le Graphe Sans-Frappe associé ne contient pas le processus c_1, car il exerce une frappe sur lui-même.

Nous exhibons maintenant un algorithme (algorithme 8.1) de recherche des n-cliques dans le Graphe Sans-Frappe (V, E) des Frappes de Processus $(\Sigma, L, \mathcal{H})$ avec $n = |\Sigma|$. Cet algorithme, d'une



(a) Hyper-graphe des frappes (b) Graphe Sans-Frappe

Figure 8.2 – Frappes de Processus représentées par l'hyper-graphe des frappes (a) et le Graphe Sans-Frappe (b). Le Graphe Sans-Frappe contient une et une seule 3-clique entre a_0, b_0 et c_0 (lignes épaisses) : l'état $\langle a_0, b_0, c_0 \rangle$ est le seul point fixe des Frappes de Processus.

complexité exponentielle selon le nombre de sortes, prend avantage de la propriété 8.2 pour élaguer au maximum l'énumération des n-cliques potentielles.

Propriété 8.2. *Une n-clique d'un graphe n-parti possède un et un seul sommet dans chaque partition du graphe.*

Algorithme 8.1 (Énumération des n-cliques d'un graphe n-parti)**.** Soit (V, E) un graphe n-parti tel que $V = \bigcup_{a \in \Sigma} V^a,\ V^a \subseteq L_a$, où $|\Sigma| = n$.

1. Si il existe une partition vide, alors, aucune n-clique n'est présente (propriété 8.2). Nous supposons par la suite que $V^a \neq \emptyset,\ \forall a \in \Sigma$.
2. Pour chaque partition V^a et chaque sommet $a_i \in V^a$, nous définissons, avec toute autre partition V^b, l'ensemble des sommets $E^b_{a_i}$ présents dans la partition V^b en relation avec a_i : $E^b_{a_i} = \{b_j \in V^b \mid \{a_i, b_j\} \in E\},\ \forall b \in \Sigma, b \neq a$. Enfin, nous posons $E^a_{a_i} = \{a_i\}$.
3. Nous répétons autant que possible le processus d'élagage suivant : pour chaque processus a_i, si il existe $b \in \Sigma$ tel que $E^b_{a_i} = \emptyset$, le processus a_i est supprimé des candidats, car il ne peut appartenir à une n-clique (propriété 8.2).
4. Enfin, nous énumérons les n-cliques potentielles avec la méthode récursive suivante, appelée initialement avec $C = \emptyset$:
   (a) si C contient un processus de chaque sorte, alors C est une n-clique.
   (b) Sinon, choisir une nouvelle partition V^a (C ne contient pas de processus de sorte a) ;
   (c) pour chaque processus $a_i \in V^a$: vérifier que a_i est en relation avec tous les processus de C, si oui, recommencer la procédure avec $C = C \cup \{a_i\}$.

   Afin d'optimiser cette énumération, le choix de la partition peut se faire selon la cardinalité de ses relations : choisir la partition avec le plus faible nombre de relations réduira le processus itératif.

Sur l'exemple de la figure 8.2, c_1 est immédiatement supprimé des candidats ($E^b_{c_1} = \emptyset$) ; les partitions associées à b et c sont d'abord choisies (car impliquées dans seulement 3 relations), et

un seul état est entièrement testé, $\langle a_0, b_0, c_0 \rangle$, se révélant être une 3-clique et donc le seul point fixe des Frappes de Processus.

8.3.2 Application aux Points Fixes Topologiques des Graphes des Interactions

Dans la section 3.4 page 28 (chapitre 3) nous avons défini en Frappes de Processus la dynamique généralisée d'un graphe des interactions : la dynamique obtenue contient la dynamique de tous les réseaux discrets issus d'un paramétrage de ce graphe des interactions.

Ainsi, étant données les Frappes de Processus issues de la dynamique généralisée d'un graphe des interactions, les points fixes de la dynamique obtenus sont également des points fixes de tous les réseaux discrets correspondant à ce graphe des interactions, et donc, d'après la section 8.2 sont des points fixes topologiques.

Dans cette sous-section nous montrons une preuve alternative de la caractérisation des points fixes topologiques présentée en section 8.2 en raisonnant uniquement dans le cadre des Frappes de Processus. La preuve ainsi obtenue est directe et intuitive, démontrant l'intérêt des Frappes de Processus en tant qu'outil de réflexion. Le raisonnement par abstraction via la dynamique généralisée restreint toutefois le résultat aux graphes des interactions sans arcs non-signés, c.-à-d. un composant ne peut pas à la fois être un régulateur positif et négatif d'un même composant. En revanche, obtenue dans un premier temps sur les réseaux booléens, la preuve se généralise simplement aux réseaux discrets en général, sous les contraintes discutées en section 3.4, à savoir l'extrémité des paramètres discrets.

Théorème 8.3. *Soit $\mathcal{G} = (\Gamma, E_+, E_-)$ un graphe des interactions connexe (définition 3.3 page 27). Si tous ses sommets ont un prédécesseur, $E_+ \cap E_- = \emptyset$ et $\mathcal{G}$ ne possède aucun cycle (non-orienté) négatif, alors la dynamique généralisée en Frappes de Processus de ce graphe contient exactement* deux points fixes s^1, s^2 *avec* $\forall a \in \Gamma, s^1[a] \in \{a_0, a_{l_a}\}$ *et* $s^2[a] = \begin{cases} a_0 & \text{si } s^1[a] = a_{l_a} \\ a_{l_a} & \text{sinon,} \end{cases}$ *où l_a est le niveau maximum du composant a.*
Si $\mathcal{G}$ ne satisfait pas ces conditions, alors les Frappes de Processus ne contiennent aucun point fixe.

Démonstration. Nous remarquons au préalable que si s est un point fixe des Frappes de Processus, alors il existe un cycle entre tous les sous-ensembles possibles de processus présents dans s dans le Graphe Sans-Frappe correspondant (formant alors une $|\Gamma|$-clique).

Si il existe un composant a sans prédécesseur dans $\mathcal{G}$, alors tous les processus de a s'auto-frappent dans la dynamique généralisée, les supprimant ainsi du Graphe Sans-Frappe correspondant, et rendant impossible la présence d'une $|\Gamma|$-clique.

Si il existe $a \to b \in E_+ \cap E_-$, alors il n'existe aucune relation entre un processus de sorte a et un processus de sorte b dans le Graphe Sans-Frappe, et donc aucune $|\Gamma|$-clique n'est présente.

Plaçons-nous tout d'abord dans le cas booléen ($\forall a \in \Gamma, l_a = 1$). D'après la construction de la dynamique généralisée, la figure 8.3 illustre les différentes relations élémentaires possibles entre deux composants. Il apparaît alors qu'une $|\Gamma|$-clique est possible dans le Graphe Sans-Frappe si et seulement si tout cycle (non-orienté) dans $\mathcal{G}$ est positif.

Comme l'illustre la figure 8.4 le cas où les composants ont un nombre fini et dénombrable de niveaux est similaire au cas booléen : une clique n'est possible qu'entre les niveaux extrêmes des composants, et impose l'absence de cycle (non-orienté) négatif dans $\mathcal{G}$.

Enfin, d'après les relations possibles dans le Graphe Sans-Frappe, nous obtenons que si s^1 est un point fixe alors $\forall a \in \Gamma, s^1[a] \in \{a_0, a_{l_a}\}$ et il existe un et un seul autre point fixe s^2, avec $\forall a \in \Gamma, s^2[a] = \begin{cases} a_0 & \text{si } s^1[a] = a_{l_a} \\ a_{l_a} & \text{sinon,} \end{cases}$ d'après la symétrie des relations dans le Graphe Sans-Frappe. □

8.3.3 Jardins d'Éden

En généralisant la définition 8.8 d'un jardin d'Éden dans un graphe de transitions dont les sommets sont les états et les arcs (orientés) les transitions possibles, il est facile de remarquer qu'un jardin d'Éden est un point fixe du graphe où le sens des transitions a été inversé.

La définition 8.12 introduit les Frappes de Processus Inverses où la cible et le bond de chaque action sont inversés. Le théorème 8.4 assure alors que les Frappes de Processus ainsi obtenues décrivent exactement les transitions inverses des Frappes de Processus de départ. Ainsi, nous pouvons établir le corollaire 8.1, permettant la caractérisation de tous les jardins d'Éden de Frappes de Processus données à l'aide des résultats de la sous-section 8.3.1 : un état est un jardin d'Éden pour des Frappes de Processus données si et seulement si il est un point fixe des Frappes de Processus Inverses

Définition 8.12 (Frappes des Processus Inverses)**.** Étant données des Frappes de Processus $(\Sigma, L, \mathcal{H})$, les *Frappes de Processus Inverses* sont définies par $(\Sigma, L, \mathcal{H}^{-1})$, où
– $a_j \to a_j \Rsh a_i \in \mathcal{H}^{-1}$ si et seulement si $a_i \to a_i \Rsh a_j \in \mathcal{H}$;
– $a_i \to b_k \Rsh b_j \in \mathcal{H}^{-1}$ si et seulement si $a_i \to b_j \Rsh b_k \in \mathcal{H}$, avec $a_i \neq b_j$.

Théorème 8.4. *Soient $(\Sigma, L, \mathcal{H})$ des Frappes de Processus, $(\Sigma, L, \mathcal{H}^{-1})$ les Frappes de Processus Inverse et $s, s' \in L$ deux états. $\exists h \in \mathcal{H}, s \cdot h = s'$ si et seulement si $\exists h' \in \mathcal{H}^{-1}, s' \cdot h' = s$.*

Démonstration. Soit $h = a_i \to s[b] \Rsh s'[b] \in \mathcal{H}$ tel que $s \cdot h = s'$ avec $\{a, b\} \subset \Sigma$. Si $a_i = s[b]$, alors $h' = s'[b] \to s'[b] \Rsh s[b] \in \mathcal{H}^{-1}$, donc $s' \cdot h' = s$; si $a_i \neq s[b]$, alors $h' = a_i \to s'[b] \Rsh s[b] \in \mathcal{H}^{-1}$, donc $s' \cdot h' = s$. Enfin, d'après la définition 8.12, $(\mathcal{H}^{-1})^{-1} = \mathcal{H}$, donc la réciproque est vraie. □

Corollaire 8.1. *Soient $(\Sigma, L, \mathcal{H})$ des Frappes de Processus et $s \in L$ un état. s est un point fixe de $(\Sigma, L, \mathcal{H})$ si et seulement si s est un jardin d'Éden des Frappes de Processus Inverses $(\Sigma, L, \mathcal{H}^{-1})$.*

8.4 Discussion

Dans ce chapitre, nous avons présenté des analyses statiques originales permettant la caractérisations de certains points fixes dans le cadre des Réseaux Booléens et la caractérisation complète des points fixes des dynamiques des Frappes de Processus.

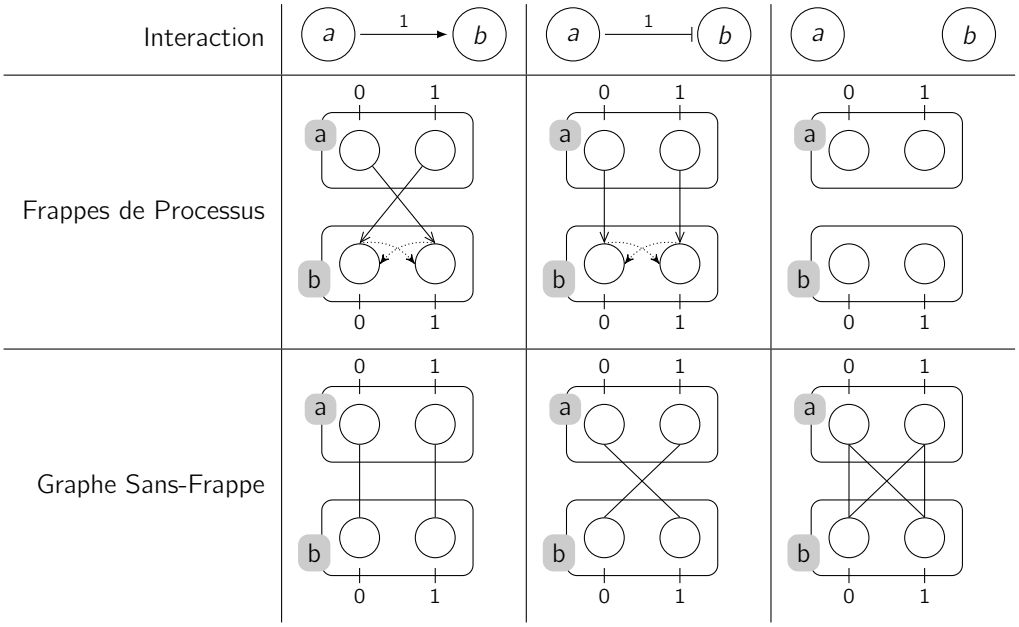


Figure 8.3 – Briques de base pour la construction de la dynamique généralisée en Frappes de Processus d'un graphe des interactions où tous les composants sont dotés de deux niveaux (notés 0 et 1) ; avec le Graphe Sans-Frappe correspondant.

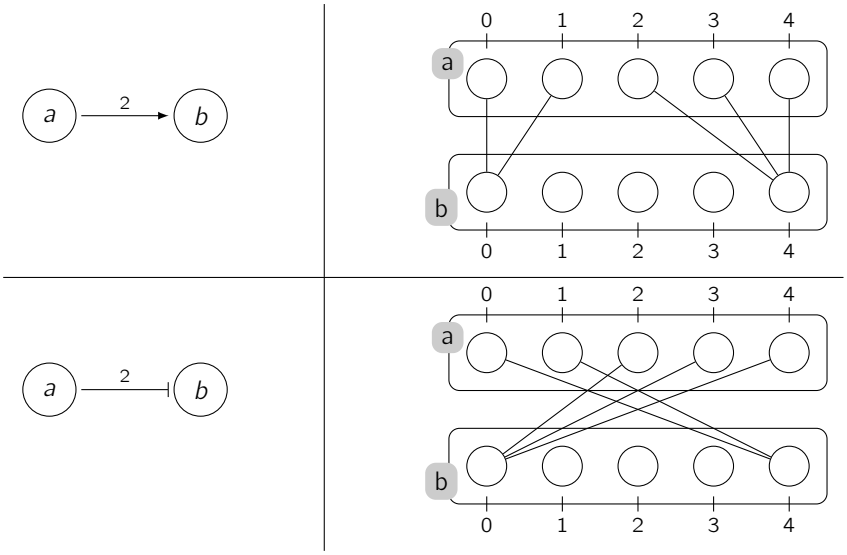


Figure 8.4 – (droite) Graphe Sans-Frappe obtenu lors de la construction de dynamique généralisée d'une interaction élémentaire entre les composants *a* et *b* (gauche) possédant chacun 5 niveaux.

Concernant les Réseaux Booléens, nous avons démontré une relation directe entre la topologie du graphe des interactions et la caractérisation des points fixes dits topologiques : ces points sont fixes pour toute application booléenne possédant ce même graphe des interactions. De ce fait, de tels points fixes sont calculables avec une complexité linéaire selon la taille du graphe.

Concernant les Frappes de Processus, la caractérisation complète de tous les points fixes (et des jardins d'Éden) est équivalente à la recherche des n-cliques (n est le nombre de sortes) dans le Graphe Sans-Frappe, une représentation complémentaire des Frappes de Processus où sont mis en relations les processus n'agissant pas entre-eux. Un algorithme permettant une recherche efficace de ces n-cliques a été présenté ; bien que d'une complexité maximale exponentielle, les méthodes d'élagage employées offrent de bons résultats en pratique (voir chapitre 11). Des travaux futurs pourraient toutefois prendre avantage d'une analyse topologique plus poussée des Frappes de Processus afin d'optimiser la recherche de ces n-cliques.

Cette approche purement structurelle apporte une compréhension efficace de la preuve d'existence ou d'absence de points fixes. Appliquée à la caractérisation des points fixes dans la dynamique généralisée en Frappes de Processus des réseaux de régulation biologique, nous obtenons directement des résultats en rapport avec les points fixes topologiques dans le cadre des Réseaux Booléens. Ceci montre l'utilité des Frappes de Processus en tant qu'outil d'aide à la réflexion, permettant la formulation simple de raisonnements intuitifs. À propos, historiquement, la caractérisation des points fixes topologiques dans les Réseaux Booléens présentée en section 8.2 a été inspirée par les résultats obtenus sur les points fixes de la dynamique généralisée présentés en sous-section 8.3.2.

Enfin, la caractérisation des attracteurs de la dynamique requiert une analyse des dynamiques possibles bien plus poussée que pour les points fixes et constitue une perspective intéressante dans le cadre du développement de techniques d'analyse statique des dynamiques des Frappes de Processus.

Chapitre **9**

Interprétation Abstraite des Scénarios dans les Frappes de Processus

L'analyse des dynamiques des Réseaux de Régulation Biologique (RRB) requiert des méthodes innovantes pour contenir l'explosion de l'espace des états à explorer. Dans ce chapitre, nous présentons une approche originale pour la décision de propriétés d'atteignabilité en exploitant la simplicité des Frappes de Processus introduites dans le chapitre 3. En définissant des abstractions complémentaires d'une succession d'actions (scénario), nous construisons plusieurs approximations supérieures (sup.) et inférieures (inf.) de propriétés d'atteignabilité successive des processus au sein des Frappes de Processus.
L'extraction des processus nécessaires à la satisfaction d'une propriété d'atteignabilité donnée dérive de nos méthodes. Une telle information peut ainsi conduire au contrôle du système modélisé en prévenant l'atteinte de tels processus clés.
Les analyses obtenues possèdent une complexité théorique limitée (polynomiale selon le nombre total de processus et exponentielle selon le nombre de processus au sein d'une seule sorte) promettant une applicabilité efficace pour la vérification de très grands modèles en Frappes de Processus ; ce que nous montrons dans le chapitre 11.

9.1 Préliminaires

L'analyse statique par interprétation abstraite (Cousot & Cousot, 1977) aspire à fournir des analyses efficaces d'un modèle sans l'exécuter. De manière générale, ceci est acquis via l'abstraction d'un modèle et de sa sémantique afin d'obtenir un système abstrait dont le comportement est plus simple à interpréter (comprendre, analyser) que le système concret. Étant donnée une propriété sur la dynamique du modèle dont la validité doit être vérifiée, les méthodes par interprétation abstraite produisent alors des approximations de cette validité (et peuvent donc se révéler non-concluantes) : une approximation supérieure (sup.) permet de garantir sa non-validité, alors qu'une approximation inférieure (inf.) garantit sa validité.

Des travaux récents ont notamment construit des analyses génériques par interprétation abstraite des systèmes mobiles (Feret, 2005), des systèmes (séquentiels) hybrides (Bouissou, 2008), ou encore du langage κ (Danos et coll., 2008) dans le cadre des modélisations à l'échelle moléculaire des systèmes biologiques.

Dans ce chapitre, nous développons une analyse statique par interprétation abstraite dédiée aux Frappes de Processus avec comme objectif l'obtention d'algorithmes très efficaces exploitant la simplicité de ce formalisme.

En particulier, même si une exécution abstraite du modèle est produite par certaines approximations, et sert en général à expliquer notre méthode, nos techniques se veulent également proches d'une analyse topologique des modèles. En effet, la plupart des approximations sup. et inf. développées résultent de la recherche de propriétés particulières dans des structures abstraites extraites des Frappes de Processus. Cette structure étant de taille limitée (polynomiale selon le nombre total de processus et exponentielle selon le nombre de processus au sein d'une seule sorte) et sa construction d'une complexité raisonnable (exponentielle selon le nombre de processus au sein d'une seule sorte), les analyses produites aspirent à supporter des Frappes de Processus possédant un très grand nombre de sortes, du moment que le nombre de processus par sorte est limité.

Nous nous intéressons ici à la décision de propriétés d'atteignabilité successive de processus. Notre approche se base sur deux abstractions complémentaires des scénarios (successions d'application d'actions) des Frappes de Processus : en *séquences d'objectifs* et en *séquences de bonds*. Une séquence d'objectifs peut représenter la spécification d'une propriété d'atteignabilité. Alors, à l'aide des séquences de bonds (calculées directement depuis les Frappes de Processus), nous définissons des *opérateurs de raffinement* permettant de préciser (de rendre plus concrète) une séquence d'objectifs en la complétant par des atteignabilités nécessaires. Par le développement de raisonnements récursifs et itératifs, nous définissons alors plusieurs approximations sup. et inf. de la concrétisabilité d'une propriété d'atteignabilité donnée, c.-à-d. de l'existence d'un scénario pouvant être abstrait par cette propriété.

Le chapitre 11 illustre l'application de notre méthode pour l'analyse de RRB contenant plus de cent composants. Contrairement aux méthodes dynamiques existantes, pour lesquelles une telle analyse est souvent impossible, notre implémentation répond quasiment instantanément aux décisions d'atteignabilité. Ceci encourage l'application de nos méthodes sur des RRB encore plus grands, mais également à la poursuite du développement de nouvelles analyses statiques permettant la dérivation d'autres propriétés dynamiques des Frappes de Processus.

Ce chapitre est structuré comme suit. La section 9.2 résume les résultats obtenus en se focalisant sur leur mise en œuvre. La section 9.3 présente nos abstractions complémentaires des scénarios de Frappes de Processus et définit les opérateurs de raffinement des abstractions. Ces opérateurs de raffinement sont alors utilisés pour construire des approximations sup. et inf. de l'atteignabilité de processus dans la section 9.4, détaillant également leur implémentation et complexité. Enfin, la section 9.5 résume et discute les contributions de ce chapitre.

Une version préliminaire des résultats présentés dans ce chapitre est publiée dans (Paulevé, Magnin & Roux, 2011*a*).

9.2 Résumé des Résultats Obtenus

Cette section présente, de manière informelle, les différents résultats acquis dans ce chapitre pour l'analyse statique de propriétés d'atteignabilité des Frappes de Processus. Dans un souci de simplicité, nous nous focaliserons donc sur la mise en pratique des analyses proposées, et non sur les techniques des preuves, détaillées dans les sections suivantes.

Approche Générale. Soit $\mathcal{PH} = (\Sigma, L, \mathcal{H})$ des Frappes de Processus pour lesquelles nous voulons vérifier la propriété d'atteignabilité $\mathcal{R}$ suivante : depuis un état $s \in L$ donné, il est possible d'atteindre le processus a_i, puis (après un certain nombre d'actions) le processus b_j, etc. Les travaux présentés dans ce chapitre établissent des propriétés $\mathcal{P}$ et $\mathcal{Q}$ permettant respectivement l'approximation supérieure (sup.) et inférieure (inf.) de cette propriété $\mathcal{R}$:

Si $\mathcal{PH}$ n'a pas la propriété $\mathcal{P}$, alors l'atteignabilité $\mathcal{R}$ est impossible (approximation sup. de $\mathcal{R}$).

Si $\mathcal{PH}$ a la propriété $\mathcal{Q}$, alors l'atteignabilité $\mathcal{R}$ est possible (approximation inf. de $\mathcal{R}$).

Les propriétés $\mathcal{P}$ et $\mathcal{Q}$ se vérifient efficacement de manière statique sur $\mathcal{PH}$, permettant ainsi une approximation sup. et inf. rapide de $\mathcal{R}$. Dans les faits, $\mathcal{P}$ et $\mathcal{Q}$ sont des disjonctions entre plusieurs propriétés ($\mathcal{P} = \mathcal{P}_1 \vee \mathcal{P}_2 \vee \dots$) : leur vérification peut alors être simplifiée par la vérification d'une des sous-propriétés. Nous esquissons ces propriétés dans la suite de cette section.

Dans le cadre de l'étude d'un Réseau Booléen (ou Discret) F (voir chapitre 2 page 9), l'encodage de F en Frappes de Processus (noté $\mathbf{PH}(F)$) résultant en une approximation sup. des comportements de f (chapitre 4), les approximations sup. apportées dans ce chapitre ont une application directe pour l'analyse de F :

Si $\mathbf{PH}(F)$ n'a pas la propriété $\mathcal{P}$, alors l'atteignabilité $\mathcal{R}$ est impossible pour F.

Structures Abstraites des Frappes de Processus. Le calcul des propriétés $\mathcal{P}$ et $\mathcal{Q}$ se base sur différentes structures abstrayant les Frappes de Processus : ces abstractions oublient certaines informations afin de rendre la décision des propriétés plus simple. La construction et la taille de ces structures se veut d'une complexité limitée.

En pratique ces structures sont des graphes mettant en relation trois types de nœuds : les *processus*, les *objectifs* (par exemple $a_0 \Gamma^* a_i$) spécifiant l'atteinte d'un processus depuis un processus initial, et les *solutions* décrivant un ensemble de processus nécessaires à la résolution d'un objectif. Ainsi, ces structures contiennent des informations sur les processus nécessaires à l'atteinte de certains processus voulus, permettant alors, à travers différentes constructions, de mettre en avant des contradictions (propriétés $\mathcal{P}_1, \mathcal{P}_2, \dots$) ou des conditions suffisantes (propriétés $\mathcal{Q}_1, \mathcal{Q}_2, \dots$) pour la satisfaction de $\mathcal{R}$.

Deux structures abstraites $\mathcal{A}$ et $\lceil\mathcal{B}\rceil$ sont principalement utilisées pour calculer respectivement $\mathcal{P}$ et $\mathcal{Q}$. La construction de $\mathcal{A}$ s'effectue récursivement : partant des processus dont l'atteignabilité est demandée par $\mathcal{R}$, les processus sont liés à l'objectif les atteignant depuis l'état initial (imposé également par $\mathcal{R}$) ; ensuite, toutes les solutions d'un objectif sont calculées : une solution contient l'ensemble minimal des processus permettant, localement, de résoudre l'objectif. La construction de $\lceil\mathcal{B}\rceil$ utilise $\mathcal{A}$ comme base : si un objectif a plusieurs solutions, une solution peut être choisie arbitrairement (les autres sont supprimées) ; si un processus a_i est référencé dans $\lceil\mathcal{B}\rceil$, tous les processus de sorte a sont liés à un objectif partant de a_i.

La figure 9.2(haut) et (bas) illustre respectivement les structures $\mathcal{A}$ et $\lceil\mathcal{B}\rceil$ calculées à partir des Frappes de Processus données dans la figure 9.1 depuis l'état initial $\langle a_1, b_0, c_0, d_1\rangle$ pour $\mathcal{A}$ et $\langle a_1, b_1, c_1, d_0\rangle$ pour $\lceil\mathcal{B}\rceil$ afin d'étudier l'atteignabilité du processus d_2. Les nœuds représentés par un cercle sont les solutions, les objectifs annotés par le symbole $\perp$ sont les objectifs sans solution.

Propriétés Principales. Nous énonçons ici les propriétés d'approximation sup. $\mathcal{P} = \mathcal{P}_1 \vee \mathcal{P}_2 \vee \mathcal{P}_3$ et d'approximation inf. $\mathcal{Q} = \mathcal{Q}_1 \vee \mathcal{Q}_2$ pour la décision de l'atteignabilité $\mathcal{R}$ sur des Frappes de

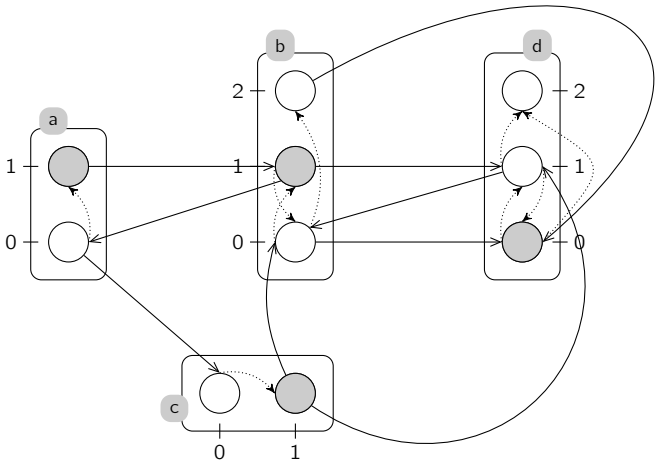
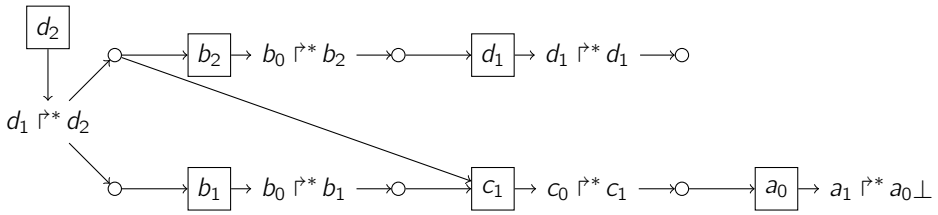


Figure 9.1 – Exemple de Frappes de Processus. L'état actuel contient les processus grisés : $\langle a_1, b_1, c_1, d_0 \rangle$.



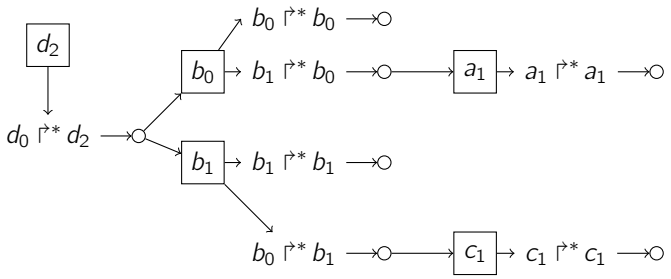


Figure 9.2 – (haut) Structure abstraite $\mathcal{A}$ calculée sur les Frappes de Processus de la figure 9.1 pour l'atteignabilité de d_2 depuis l'état $\langle a_1, b_0, c_0, d_1 \rangle$. (bas) Structure abstraite $\lceil \mathcal{B} \rceil$ pour l'atteignabilité de d_2 depuis l'état $\langle a_1, b_1, c_1, d_0 \rangle$.

Processus données. Partant de la structure abstraite $\mathcal{A}$, nous obtenons une première approximation sup. $\mathcal{P}_1$ de $\mathcal{R}$ (cas restreint du théorème 9.1 page 148 par souci de simplicité) :

$\mathcal{P}_1$: la structure abstraite $\mathcal{A}'$ contient tous les processus dont l'atteignabilité est demandée par $\mathcal{R}$.

La structure $\mathcal{A}'$ est obtenue à partir de la structure $\mathcal{A}$ en supprimant récursivement les objectifs sans solution, les processus sans objectif, et les solutions dont un processus a été supprimé par ce procédé. La figure 9.2(haut) donne un exemple de structure $\mathcal{A}$ pour laquelle $\mathcal{P}_1$ n'est pas vérifée.

En se basant sur la structure abstraite $\lceil\mathcal{B}\rceil$, l'approximation inf. $\mathcal{Q}_1$ est également démontrée (théorème 9.4 page 157) :

$\mathcal{Q}_1$: la structure abstraite $\lceil\mathcal{B}\rceil$ est sans cycle et toutes ses feuilles sont des solutions.

La figure 9.2(bas) donne un exemple de structure $\lceil\mathcal{B}\rceil$ pour laquelle la propriété $\mathcal{Q}_1$ est vérifiée.

La vérification des propriétés $\mathcal{P}_1$ et $\mathcal{Q}_1$ est dite désordonnée : l'ordre des atteignabilités dans $\mathcal{R}$ n'est pas pris en compte. Les propriétés $\mathcal{P}_2$ (théorème 9.2 page 152) et $\mathcal{Q}_2$ (corollaire 9.3 page 157) prennent avantage de cette séquentialité via un raisonnement itératif :

$\mathcal{P}_2$: $\mathcal{P}_1$ est vraie avec $\mathcal{R}_1$ et $\mathcal{P}_2$ est vraie avec $\mathcal{R}_{2..}$.

$\mathcal{Q}_2$: $\mathcal{Q}_1$ est vraie avec $\mathcal{R}_1$ et $\mathcal{Q}_2$ est vraie avec $\mathcal{R}_{2..}$.

$\mathcal{R}_1$ représente $\mathcal{R}$ tronqué à sa première atteignabilité et $\mathcal{R}_{2..}$ la propriété $\mathcal{R}$ à partir de sa seconde atteignabilité et dont l'état initial a été modifié (nous ne détaillons pas cette étape ici).

Enfin, dans le cas de l'approximation sup. $\mathcal{P}$, une dernière propriété $\mathcal{P}_3$ est démontrée en exploitant des contraintes d'ordre entre les occurences des processus, extraites statiquement depuis les Frappes de Processus : nous écrivons $q \lhd p$ si le processus q ne peut pas apparaître après p (théorème 9.3 page 154) :

$\mathcal{P}_3$: pour tout $p \in \mathsf{minProc}(\mathcal{R}_{n..})$ et $q \in \mathsf{minProc}(\mathcal{R}_{m..})$, $\neg(q \lhd p)$ pour tout $n < m$ possible,

où minProc calcule les processus nécessaires à la satisfaction de $\mathcal{R}_{n..}$ (resp. $\mathcal{R}_{m..}$).

Le calcul de la propriété $\mathcal{P}$ permet également d'extraire un ensemble de processus clés pour sa satisfaction, résultant en le raisonnement suivant, où z_l est un processus clé et $\mathcal{PH} \setminus z_l$ représente les Frappes de Processus $\mathcal{PH}$ où le processus z_l est supprimé :

Si $\mathcal{PH}$ a la propriété $\mathcal{P}$, alors $\mathcal{PH} \setminus z_l$ n'a pas la propriété $\mathcal{P}$.

Ainsi, si $\mathcal{PH}$ satisfait $\mathcal{Q}$ (et donc $\mathcal{R}$), la suppression du processus clé z_l assure que $\mathcal{R}$ n'est plus satisfait. Dans le cadre de l'étude des RRB, la suppression d'un processus peut être vue comme une opération dite de *knockdown* sur le composant associé : certains niveaux du composants vont être bloqués (par un médicament par exemple).

Dans les sections suivantes, les successions d'atteignabilités sont spécifiées (de manière équivalente) par des *séquences d'objectifs* ; et la notion d'état est généralisée à la notion de contexte.

Aide pour la lecture des sections suivantes

La suite de chapitre détaille et prouve les résultats présentés dans cette section en utilisant une méthode de preuve qui se veut générique et réutilisable pour le développement de travaux futurs,

discutés notamment dans la section 9.5 et le chapitre 12.

La section 9.3 définit notamment deux abstractions complémentaires des scénarios : les *séquences d'objectifs*, permettant la spécification d'une succession d'atteignabilités de processus, et les *séquences de bonds*, spécifiant des atteignabilités supplémentaires nécessaires pour l'atteinte d'un processus donné. Des opérateurs dits de *raffinement* sont alors développés : partant d'une séquence d'objectifs (donc d'une abstraction pouvant représenter plusieurs scénarios), et en exploitant la complémentarité avec les séquences de bonds, ces opérateurs calculent des séquences d'objectifs plus *précises*, c.-à-d. représentant avec plus de détails *le même ensemble de scénarios*, dits concrets.

La section 9.4 présente alors différentes conditions nécessaires (les approximations sup., dénotées dans cette section par $\mathcal{P}_1$, $\mathcal{P}_2$ et $\mathcal{P}_3$) ou suffisantes (les approximations inf., dénotées dans cette section par $\mathcal{Q}_1$ et $\mathcal{Q}_2$) pour l'*existence de scénarios concrets* représentés par une séquence d'objectifs donnée. La séquence d'objectifs représentant la propriété d'atteignabilité successive à vérifier, l'existence de tels scénarios est équivalente à la validité de la propriété.

Afin de faciliter la compréhension des techniques développées dans cette dernière section, l'application des différentes approximations est détaillée sur des exemples à chaque fin de sous-section.

9.3 Interprétation Abstraite des Scénarios

Après avoir introduit des définitions préliminaires (sous-section 9.3.1), cette section établit deux abstractions complémentaires des scénarios : par les *séquences d'objectifs* (sous-section 9.3.2) et par les *séquences de bonds* (sous-section 9.3.3). La première abstraction décrit une succession de changements de processus par sortes (appelés objectifs), alors que la dernière détaille les actions jouées lors de la résolution de ces objectifs. Alors que les séquences d'objectifs peuvent être vues comme des représentations partielles des scénarios, les séquences de bonds mettent en avant les actions nécessaires à la résolution d'un objectif.

En utilisant la complémentarité de ces deux abstractions, un ensemble d'opérateurs de raffinement des séquences d'objectifs est dérivé dans la sous-section 9.3.4. Ces raffinements ont pour but de préciser automatiquement une abstraction donnée afin de rendre la décision de sa concrétisabilité plus simple à calculer. La figure 9.3 résume les dérivations possibles entre des Frappes de Processus et les différentes abstractions des scénarios. Nous pouvons ici voir une abstraction comme la spécification d'une propriété, et la décision de concrétisabilité comme la satisfaction de la spécification sur le modèle donné.

9.3.1 Préliminaires

Cette sous-section introduit les notions d'*objectif* et de *contexte* utilisées par les abstractions développées dans ce chapitre.

L'atteignabilité du processus a_j depuis un processus a_i est appelée un objectif qui est dénoté par $a_i \Rsh^* a_j$ (définition 9.1).

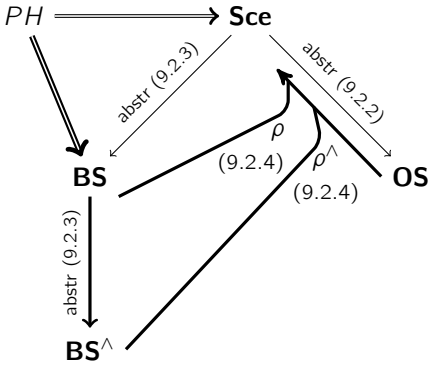


Figure 9.3 – Relations de dérivation entre les Frappes de Processus (PH), les scénarios (**Sce**), les séquences d'objectifs (**OS**, sous-section 9.3.2), les séquences de bonds (**BS** et $\mathbf{BS}^\wedge$, sous-section 9.3.3) et les opérateurs de raffinement (p. ex. ρ et $\rho^\wedge$, sous-section 9.3.4). Les relations en lignes épaisses sont utilisées dans la section 9.4 pour la décision de concrétisabilité d'une séquence d'objectifs.

Définition 9.1 (Objectif (**Obj**))**.** L'atteignabilité d'un processus a_j depuis a_i est appelé un *objectif*, noté $a_i \Rsh^* a_j$. L'ensemble des objectifs est dénoté par $\mathbf{Obj} = \{a_i \Rsh^* a_j \mid a \in \Sigma \wedge (a_i, a_j) \in L_a^2\}$. Étant donné un objectif $P \in \mathbf{Obj}$, où $P = a_i \Rsh^* a_j$, $\Sigma(P) = a$, $\mathsf{cible}(P) = a_i$, $\mathsf{bond}(P) = a_j$. Un objectif P est *trivial* si $\mathsf{cible}(P) = \mathsf{bond}(P)$.

Nous étendons la notion d'état par la notion de contexte (définition 9.2). Un contexte référence l'ensemble des processus pour chaque sorte qui peuvent être considérés comme l'état courant.

Définition 9.2 (Contexte ς (**Ctx**))**.** Un *contexte* ς associe à chaque sorte dans Σ un sous-ensemble non-vide de ses processus : $\forall a \in \Sigma, \varsigma[a] \subseteq L_a \wedge \varsigma[a] \neq \emptyset$. **Ctx** réfère à l'ensemble des contextes.

Étant donné un contexte ς, nous notons $a_i \in \varsigma$ si et seulement si $a_i \in \varsigma[a]$; $ps \in \wp(\mathbf{Proc})$, $ps \subseteq \varsigma \Leftrightarrow \forall a_i \in ps, a_i \in \varsigma$. Le recouvrement d'un contexte ς par un ensemble de processus ps est noté $\varsigma \Cap ps$ (définition 9.3). Par exemple, $\langle a_1, a_2, b_1, c_1\rangle \Cap \{a_3, b_2, b_3\} = \langle a_3, b_2, b_3, c_1\rangle$.

Définition 9.3 ($\Cap : \mathbf{Ctx} \times \wp(\mathbf{Proc}) \mapsto \mathbf{Ctx}$)**.** Étant donnés un contexte $\varsigma \in \mathbf{Ctx}$ et un ensemble de processus $ps \in \wp(\mathbf{Proc})$, le recouvrement de ς par ps est noté $\varsigma \Cap ps$ et est défini par

$$\forall a \in \Sigma, (\varsigma \Cap ps)[a] = \begin{cases} \{p \in ps \mid \Sigma(p) = a\} & \text{si } \exists p \in ps, \Sigma(p) = a, \\ \varsigma[a] & \text{sinon.} \end{cases}$$

De manière similaire à la définition 3.2 page 26, un scénario $\delta \in \mathbf{Sce}$ est *jouable* dans le contexte ς si et seulement si $\mathsf{support}(\delta) \subseteq \varsigma$. Le jeu de δ dans ς est dénoté par $\varsigma \cdot \delta$ où $\varsigma \cdot \delta = \varsigma \Cap \mathsf{fin}(\delta)$.

9.3.2 Abstraction des Scénarios en Séquences d'Objectifs

Au cours de l'exécution d'un scénario, nous pouvons observer des processus de différentes sortes bondir les un après les autres, en suivant le jeu des actions. Une abstraction d'une telle exécution est une succession d'objectifs : le processus a_j est atteint (après un certain nombre d'actions) depuis a_i, puis le processus b_j est atteint depuis b_i, etc. Ceci forme une séquence d'objectifs (définition 9.4). La jointure d'un objectif à une séquence d'objectifs est spécifiée dans la définition 9.5.

Définition 9.4 (Séquence d'objectifs (**OS**))**.** Une *séquence d'objectifs* est une séquence $\omega = P_1 :: \ldots :: P_{|\omega|}$, où $\forall n \in \mathbb{I}^\omega, \omega_n \in \mathbf{Obj}$ et $a_i = \mathsf{cible}(\omega_n) \Rightarrow \mathsf{der}_a(\omega_{1..n-1}) \in \{\varnothing, a_i\}$. L'ensemble des séquences d'objectifs est référé par **OS**. Les définitions de der_a (équation (3.2) page 25), preum_a (équation (3.1)), $\mathsf{support}$ (équation (3.3)) et fin (équation (3.4)) sont dérivées simplement pour les séquences d'objectifs en omettant le cas des frappeurs.

Définition 9.5 ($\oplus : \mathbf{OS} \times \mathbf{Obj} \mapsto \mathbf{OS}$)**.** La jointure entre une séquence d'objectifs $\omega \in \mathbf{OS}$ et un objectif $a_i \Rsh^* a_j \in \mathbf{Obj}$ est définie par :

$$\omega \oplus a_i \Rsh^* a_j = \begin{cases} \omega :: a_i \Rsh^* a_j & \text{si } a \notin \Sigma(\omega), \\ \omega :: \mathsf{der}_a(\omega) \Rsh^* a_j & \text{sinon.} \end{cases}$$

Un scénario peut être abstrait par plusieurs séquences d'objectifs, décrivant de manière plus ou moins éparse les changements de processus. Par exemple, le scénario $a_0 \to \mathbf{c}_0 \Rsh \mathbf{c}_1 :: b_1 \to \mathbf{a}_0 \Rsh \mathbf{a}_1 :: a_1 \to \mathbf{b}_1 \Rsh \mathbf{b}_0 :: b_0 \to \mathbf{d}_0 \Rsh \mathbf{d}_1 :: d_1 \to \mathbf{b}_0 \Rsh \mathbf{b}_2$ peut être abstrait par $c_0 \Rsh^* c_1 :: \mathbf{a}_0 \Rsh^* \mathbf{a}_1 :: \mathbf{b}_1 \Rsh^* b_0 :: d_0 \Rsh^* d_1 :: b_0 \Rsh^* \mathbf{b}_2$; ou, de manière plus éparse, par $a_0 \Rsh^* a_1 :: \mathbf{b}_1 \Rsh^* \mathbf{b}_2$; ou encore par $b_1 \Rsh^* b_2$; etc. (les processus en gras sont ceux gardés dans l'abstraction suivante).

L'ensemble des scénarios concrétisant (c.-à-d. pouvant être abstrait par) une séquence d'objectifs ω dans un contexte ς est donné par $\gamma_\varsigma(\omega)$ (définition 9.6). Nous définissons également la concrétisation d'un ensemble de séquences d'objectifs comme l'union de leurs concrétisations (définition 9.7).

Définition 9.6 ($\gamma_\varsigma : \mathbf{OS} \mapsto \wp(\mathbf{Sce})$)**.** Étant donné $\omega \in \mathbf{OS}$, $\gamma_\varsigma(\omega)$ est l'ensemble des scénarios concrétisant ω dans le contexte ς :

$$\begin{aligned} \gamma_\varsigma(\omega) = \{\delta \in \mathbf{Sce} \mid & (\omega^\triangle = \varepsilon \wedge \delta = \varepsilon) \vee (\omega^\triangle \neq \varepsilon \wedge \mathsf{support}(\delta) \subseteq \varsigma \\ & \wedge \exists \phi : \mathbb{I}^\omega \mapsto \mathbb{I}^\delta, (\forall n, m \in \mathbb{I}^\omega, n < m \Leftrightarrow \phi(n) \leq \phi(m)) \\ & \wedge \forall n \in \mathbb{I}^\omega, \mathsf{bond}(\omega_n) \in \varsigma \cdot \delta_{1..\phi(n)})\} \ , \end{aligned}$$

où $\omega^\triangle$ réfère à la séquence d'objectifs ω où les objectifs triviaux ont été supprimés.

Définition 9.7 ($\gamma_\varsigma : \wp(\mathbf{OS}) \mapsto \wp(\mathbf{Sce})$)**.** $\gamma_\varsigma(\Omega) = \{\delta \in \gamma_\varsigma(\omega) \mid \omega \in \Omega\}$.

Nous remarquons que la concrétisation d'une séquence d'objectifs ω ne dépend pas de son support (noté $\mathsf{support}(\omega)$), ce dernier étant imposé par le contexte de la concrétisation. En ce sens,

nous utilisons $\star \Rsh^* a_i$ pour dénoter un objectif où la cible peut être n'importe quel processus de sorte a présent dans le contexte :

$$\gamma_\varsigma(\star \Rsh^* a_i) = \gamma_\varsigma(a_j \Rsh^* a_i), \forall a_j \in \varsigma[a] \ . \tag{9.1}$$

Nous référons enfin par α_ς l'opération d'abstraction inverse à la concrétisation d'un ensemble de scénarios (définition 9.8). Ceci produit directement la propriété 9.1 garantissant la cohérence de notre approche par concrétisation (Cousot & Cousot, 1992) : étant donné un ensemble Δ de scénarios abstaits par α_ς en un ensemble de séquences d'objectifs Ω, la concrétisation de cet ensemble Ω contient bien les scénarios concrets originaux Δ.

Définition 9.8 ($\alpha_\varsigma : \wp(\mathbf{Sce}) \mapsto \wp(\mathbf{OS})$)**.**

$$\alpha_\varsigma(\Delta) = \{\omega \in \mathbf{OS} \mid \exists \delta \in \Delta, \delta \in \gamma_\varsigma(\omega)\} \ .$$

Propriété 9.1. $\forall \Delta \in \wp(\mathbf{Sce}), \Delta \subseteq \gamma_\varsigma(\alpha_\varsigma(\Delta)) \ .$

Démonstration. $\forall \delta \in \Delta$, il existe $\omega \in \mathbf{OS}$ tel que $\delta \in \gamma_\varsigma(\omega)$ (définition 9.6). Ainsi, $\omega \in \alpha_\varsigma(\Delta)$ (définition 9.8), donc $\delta \in \gamma_\varsigma(\alpha_\varsigma(\Delta))$. □

9.3.3 Abstraction des Scénarios en Séquences de Bonds

Les séquences de bonds résultent d'un raisonnement local à une seule sorte a. Bondir de a_i vers a_j (c.-à-d., résoudre l'objectif $a_i \Rsh^* a_j$) peut demander le jeu de plusieurs actions sur le processus de sorte a, formant une *séquence de bonds* (définition 9.9). Nous remarquons que les séquences de bonds ne sont généralement pas des scénarios : par exemple $b_i \to a_i \Rsh a_j :: b_j \to a_j \Rsh a_k$ est une séquence de bonds mais n'est pas un scénario si $b_i \neq b_j$.

Pour une séquence de bonds ζ, toutes les cibles et bonds des actions partagent la même sorte $\Sigma(\zeta)$. Dans le cadre de ce chapitre, nous ne considérons pas les séquences contenant des cycles entre les cibles et bonds des actions. Ainsi, la longueur maximum d'une séquence de bonds pour une sorte a est le nombre de processus de la sorte a.

Définition 9.9 (Séquence de bonds (**BS**))**.** Une *séquence de bonds* ζ est une séquence d'actions telle que $\forall n \in \mathbb{I}^{\zeta}, n < |\zeta|, \mathsf{bond}(\zeta_n) = \mathsf{cible}(\zeta_{n+1})$. **BS** dénote l'ensemble des séquences de bonds. Nous référons à l'ensemble des séquences de bonds *résolvant* l'objectif P par $\mathbf{BS}(P)$:

$$\mathbf{BS}(a_i \Rsh^* a_j) = \{\zeta \in \mathbf{BS} \mid \mathsf{cible}(\zeta_1) = a_i \wedge \mathsf{bond}(\zeta_{|\zeta|}) = a_j\} \ .$$

Évidemment, $\mathbf{BS}(a_i \Rsh^* a_i) = \{\varepsilon\}$; et $\mathbf{BS}(a_i \Rsh^* a_j) = \emptyset$ si il n'existe aucune possibilité d'atteindre a_j depuis a_i.

L'ensemble complet des séquences de bonds peut être calculé directement à partir de l'ensemble des actions $\mathcal{H}$ des Frappes de Processus sans aucune énumération des scénarios. Étant donné un objectif $a_i \Rsh^* a_j$, le calcul des séquences de bonds $\mathbf{BS}(a_i \Rsh^* a_j)$ (définition 9.9) s'effectue par un parcours en « profondeur d'abord » des actions agissant sur la sorte a afin de former une séquence de bonds sans cycle. Un tel calcul est exponentiel suivant le nombre d'actions vers la sorte a, et peut être considéré efficace quand ce nombre est petit devant le nombre total des actions.

Nous considérons également une représentation plus éparse d'une séquence de bonds ζ résolvant un objectif P en ne considérant que l'ensemble des frappeurs de ces actions dont la sorte est différente de celle de P. Nous dénotons par $\mathbf{BS}^{\wedge}(P)$ l'ensemble de ces séquences de bonds abstraites (définition 9.10).

Définition 9.10 ($\mathbf{BS}^{\wedge} : \mathbf{Obj} \mapsto \wp(\mathbf{Proc})$)**.**

$$\mathbf{BS}^{\wedge}(P) = \{\zeta^{\wedge} \mid \zeta \in \mathbf{BS}(P), \nexists \zeta' \in \mathbf{BS}(P), \zeta'^{\wedge} \subsetneq \zeta^{\wedge}\} \ ,$$

où $\zeta^{\wedge} = \{\mathsf{frappeur}(\zeta_n) \mid n \in \mathbb{I}^{\zeta} \wedge \Sigma(\mathsf{frappeur}(\zeta_n)) \neq \Sigma(P)\}$.

Nous remarquons que $\mathbf{BS}^{\wedge}(P)$ peut être calculé directement à partir des Frappes de Processus de la même manière que $\mathbf{BS}(P)$; mais de façon plus efficace car seuls les ensembles minimaux de frappeurs sont conservés, élaguant des explorations redondantes.

Les relations d'abstraction et de concrétisation entre les scénarios et les séquences de bonds (abstraites) peuvent être dérivées facilement et ne sont pas détaillées ici.

En étudiant l'exemple de Frappes de Processus de la figure 9.1 page 138, $\zeta = a_1 \rightarrow b_1 \Rsh b_0 :: d_1 \rightarrow b_0 \Rsh b_2$ est la seule séquence de bonds résolvant l'objectif $b_1 \Rsh^* b_2$ (c.-à-d. $\mathbf{BS}(b_1 \Rsh^* b_2) = \{\zeta\}$, et $\mathbf{BS}^{\wedge}(b_1 \Rsh^* b_2) = \{\{a_1, d_1\}\}$).

9.3.4 Raffinement des Séquences d'Objectifs

Un opérateur de raffinement prend avantage de plusieurs abstractions différentes d'un même objet pour en construire une nouvelle abstraction plus précise que les originales. Intuitivement, un tel raffinement s'obtient en combinant les informations conservées dans les différentes abstractions d'entrées. Une définition générique d'opérateurs de raffinements dans le cadre de l'interprétation abstraite peut être trouvée dans (Feret, 2005).

Nous définissons dans cette sous-section plusieurs opérateurs afin de raffiner une séquence d'objectifs à l'aide des abstractions des scénarios en séquence de bonds. La séquence d'objectifs produite est alors complétée par des objectifs nécessaires à sa concrétisation, inférés depuis les séquences de bonds.

Avant d'introduire les opérateurs de raffinement des séquences d'objectifs, nous définissons la relation $\preccurlyeq_{\mathrm{OS}}$ entre deux séquences d'objectifs (définition 9.11). Informellement, si $\omega \preccurlyeq_{\mathrm{OS}} \omega'$, nous disons que ω est une abstraction plus précise que ω', et donc $\gamma_{\varsigma}(\omega) \subseteq \gamma_{\varsigma}(\omega')$ (propriété 9.2). Avec une telle définition, une séquence d'objectifs jointe à un objectif est une abstraction plus précise que l'objectif seul (propriété 9.3).

Définition 9.11 ($\preccurlyeq_{\mathrm{OS}} \subset \mathbf{OS} \times \mathbf{OS}$)**.** $\omega \preccurlyeq_{\mathrm{OS}} \omega'$ si et seulement si les propriétés suivantes sont satisfaites :
- $|\omega| \geq |\omega'|$;
- il existe une application $\phi : \mathbb{I}^{\omega'} \mapsto \mathbb{I}^{\omega}$ telle que $\forall n \in \mathbb{I}^{\omega'}, \mathsf{bond}(\omega'_n) = \mathsf{bond}(\omega_{\phi(n)})$ et $\forall n, m \in \mathbb{I}^{\omega'}, n < m \Leftrightarrow \phi(n) < \phi(m)$.

Propriété 9.2. $\omega \preccurlyeq_{\mathrm{OS}} \omega' \Longrightarrow \gamma_{\varsigma}(\omega) \subseteq \gamma_{\varsigma}(\omega')$.

Propriété 9.3. *Étant donnés* $\omega \in \mathbf{OS}$ *et* $P \in \mathbf{Obj}$, $\omega \oplus P \preccurlyeq_{\mathsf{OS}} P$.

Enfin, étant donné un objectif P, $\mathbf{BS}_\varsigma(P)$ (définition 9.12) et $\mathbf{BS}^\wedge_\varsigma(P)$ (définition 9.13) étendent respectivement $\mathbf{BS}(P)$ et $\mathbf{BS}^\wedge(P)$ au contexte ς.

Définition 9.12 ($\mathbf{BS}_\varsigma : \mathbf{Obj} \mapsto \wp(\mathbf{BS})$)**.**

$$\mathbf{BS}_\varsigma(\star \Rsh^* a_j) = \bigcup_{a_i \in \varsigma[a]} \mathbf{BS}(a_i \Rsh^* a_j) \ .$$

Définition 9.13 ($\mathbf{BS}^\wedge_\varsigma : \mathbf{Obj} \mapsto \wp(\wp(\mathbf{Proc}))$)**.**

$$\mathbf{BS}^\wedge_\varsigma(\star \Rsh^* a_j) = \bigcup_{a_i \in \varsigma[a]} \mathbf{BS}^\wedge(a_i \Rsh^* a_j) \ .$$

Raffinement d'un objectif avec BS (ρ).

Nous construisons la fonction β telle que, étant donné un objectif P, une séquence de bonds $\zeta \in \mathbf{BS}(P)$ est abstraite par $\beta(\zeta)$ en la séquence d'objectifs décrivant l'atteinte successive de ces frappeurs (définition 9.14). Par définition de $\mathbf{BS}_\varsigma$, si un scénario concrétise P dans le contexte ς, il concrétise nécessairement une séquence de bonds $\zeta \in \mathbf{BS}_\varsigma(P)$, et donc $\beta(\zeta)$. L'opérateur de raffinement $\rho(P, \mathbf{BS}_\varsigma(P))$ étend l'objectif P vers l'ensemble des séquences d'objectifs où P est préfixé par chaque séquence d'objectifs $\beta(\zeta)$, où $\zeta \in \mathbf{BS}_\varsigma(P)$ (définition 9.15). Enfin, le lemme 9.1 établit la correction de ce raffinement, assurant la préservation de l'ensemble des concrétisations.

Définition 9.14 ($\beta : \mathbf{BS} \mapsto \wp(\mathbf{OS})$)**.**

$$\beta(\zeta) = \{\omega \in \mathbf{OS} \mid |\omega| = |\zeta| \wedge \forall n \in \mathbb{I}^\zeta, \mathsf{bond}(\omega_n) = \mathsf{frappeur}(\zeta_n)\} \ .$$

Définition 9.15 ($\rho : \mathbf{Obj} \times \wp(\mathbf{BS}) \mapsto \wp(\mathbf{OS})$)**.**

$$\rho(P, zs) = \{\omega \oplus P \mid \omega \in \beta(\zeta), \zeta \in zs\} \ .$$

Lemme 9.1. $\gamma_\varsigma(P) = \gamma_\varsigma(\rho(P, \mathbf{BS}_\varsigma(P)))$.

Démonstration. ($\supseteq$) $\forall \omega \in \rho(P, \mathbf{BS}_\varsigma(P)), \omega \preccurlyeq_{\mathsf{OS}} P$, ainsi $\gamma_\varsigma(P) \supseteq \gamma_\varsigma(\rho(P, \mathbf{BS}_\varsigma(P)))$; ($\subseteq$) Par définition de $\mathbf{BS}_\varsigma(P)$, $\forall \delta \in \gamma_\varsigma(P), \exists \omega \in \rho(P, \mathbf{BS}_\varsigma(P)), \delta \in \gamma_\varsigma(\omega)$, donc $\gamma_\varsigma(P) \subseteq \gamma_\varsigma(\rho(P, \mathbf{BS}_\varsigma(P)))$. □

Raffinement d'un objectif avec BS$^\wedge$ ($\rho^\wedge$).

Le raffinement d'un objectif P avec $\mathbf{BS}^\wedge$ est similaire au raffinement précédent. Un ensemble de frappeurs $ps \in \mathbf{BS}^\wedge(P)$ est abstrait par $\beta^\wedge(ps)$ en l'ensemble des séquences d'objectifs décrivant n'importe quel ordre d'atteinte de ces frappeurs. La relation entre les séquences d'objectifs dans $\beta(\zeta)$ et dans $\beta^\wedge(ps)$ est soulignée par la propriété 9.4. Le raffinement $\rho^\wedge(P, \mathbf{BS}^\wedge_\varsigma(P))$ est présenté dans la définition 9.17 et la préservation des concrétisations est établie par le lemme 9.2.

Définition 9.16 ($\beta^\wedge : \wp(\mathbf{Proc}) \mapsto \wp(\mathbf{OS})$)**.**

$$\beta^\wedge(ps) = \{\omega \in \mathbf{OS} \mid |\omega| = |ps| \wedge \forall p \in ps, \exists n \in \mathbb{I}^\omega, \mathsf{bond}(\omega_n) = p\} \ .$$

Propriété 9.4. $\forall \zeta \in \mathbf{BS}(P), \forall \omega \in \beta(\zeta), \exists \omega' \in \beta^\wedge(\zeta^\wedge), \omega \preccurlyeq_{\mathsf{OS}} \omega'$.

Définition 9.17 ($\rho^\wedge : \mathbf{Obj} \times \wp(\wp(\mathbf{Proc})) \mapsto \wp(\mathbf{OS})$)**.**

$$\rho^\wedge(P, pss) = \{\omega \oplus P \mid \omega \in \beta^\wedge(ps), ps \in pss\} \ .$$

Lemme 9.2. $\gamma_\varsigma(P) = \gamma_\varsigma(\rho^\wedge(P, \mathbf{BS}^\wedge_\varsigma(P)))$.

Démonstration. ($\supseteq$) $\forall \omega \in \rho^\wedge(P, \mathbf{BS}^\wedge_\varsigma(P)), \omega \preccurlyeq_{\mathsf{OS}} P$; ($\subseteq$) par la propriété 9.4, $\forall \omega \in \rho(P, \mathbf{BS}_\varsigma(P))$, $\exists \omega' \in \rho^\wedge(P, \mathbf{BS}^\wedge_\varsigma(P)), \omega \preccurlyeq_{\mathsf{OS}} \omega'$, ainsi $\gamma_\varsigma(\rho(P, \mathbf{BS}_\varsigma(P)) \subseteq \gamma_\varsigma(\rho^\wedge(P, \mathbf{BS}^\wedge_\varsigma(P))$. □

Raffinement d'une séquence d'objectifs ($\widetilde{\rho}$).

Finalement, afin de généraliser les raffinements définis sur les objectifs pour les séquences d'objectifs, nous montrons un opérateur de raffinement de séquence d'objectifs s'appuyant indifféremment sur un des raffinements définis ci-dessus. Nous choisissons par exemple l'opérateur ρ et définissons le raffinement $\widetilde{\rho}(\omega, \mathbf{BS})$ (définition 9.19). Informellement, ce raffinement sélectionne un objectif ω_n dans la séquence d'objectifs ω, raffine cet objectif avec ρ, et retourne tous les entrelacements de la séquence raffinée obtenus avec la séquence d'objectifs $\omega_{1..n-1}$ précédent ω_n (fonction entrelace, définition 9.18) ; l'ensemble des concrétisations est alors préservé (lemme 9.3 ci-dessous).

Définition 9.18 (entrelace : $\mathbf{OS} \times \mathbf{OS} \mapsto \wp(\mathbf{OS})$)**.**

$$\begin{aligned}\mathsf{entrelace}(\omega^1, \omega^2) = \{\omega \in \mathbf{OS} \mid |\omega| = |\omega^1| + |\omega^2| \wedge \exists \phi^1 : \mathbb{I}^{\omega^1} \mapsto \mathbb{I}^\omega, \phi^2 : \mathbb{I}^{\omega^2} \mapsto \mathbb{I}^\omega,\\ (\forall n, m \in \mathbb{I}^{\omega^1}, n < m \Leftrightarrow \phi^1(n) < \phi^1(m))\\ \wedge(\forall n, m \in \mathbb{I}^{\omega^2}, n < m \Leftrightarrow \phi^2(n) < \phi^2(m))\\ \wedge(\nexists n^1 \in \mathbb{I}^{\omega^1}, n^2 \in \mathbb{I}^{\omega^2}, \phi^1(n^1) = \phi^2(n^2))\} \ .\end{aligned}$$

Définition 9.19 ($\widetilde{\rho} : \mathbf{OS} \times \wp(\mathbf{BS}) \mapsto \wp(\mathbf{OS})$)**.**

$$\begin{aligned}\widetilde{\rho}(\omega, \mathbf{BS}) = \{\varpi \oplus \omega_{n..|\omega|} \mid n \in \mathbb{I}^\omega, \omega' \oplus \omega_n \in \rho(\omega_n, \mathbf{BS}_\varsigma(\omega_n)),\\ \varpi \in \mathsf{entrelace}(\omega_{1..n-1}, \omega')\} \ .\end{aligned}$$

Lemme 9.3. $\gamma_\varsigma(\omega) = \gamma_\varsigma(\widetilde{\rho}(\omega, \mathbf{BS}))$.

Démonstration. ($\supseteq$) $\forall \omega' \in \widetilde{\rho}(\omega, \mathbf{BS}), \omega' \preccurlyeq_{\mathsf{OS}} \omega$;
($\subseteq$) par le lemme 9.1 et la définition 9.18, $\delta \in \gamma_\varsigma(\omega) \Rightarrow \exists \omega' \in \widetilde{\rho}(\omega, \mathbf{BS}), \delta \in \gamma_\varsigma(\omega')$. □

9.4 Approximations Sup. et Inf. de l'Atteignabilité

Nous définissons le problème de l'*Atteignabilité de Processus* comme la décision de concrétisabilité d'une séquence d'objectifs $\omega \in \mathbf{OS}$ pour des Frappes de Processus données dans un contexte ς ; c.-à-d. la non-vacuité de l'ensemble $\gamma_\varsigma(\omega)$. Le problème de l'atteignabilité de processus peut également être formulé dans une sous-classe de la logique temporelle CTL (Clarke & Emerson, 1981), restreinte à la forme suivante :

$$\Phi ::= a_i \mid a_i \wedge \varphi \qquad \varphi ::= \mathtt{EF}\ \Phi \ ,$$

où $a_i \in \mathbf{Proc}$ est vrai si l'état courant contient le processus a_i et $\mathtt{EF}$ est le prédicat classique « existe fatalement ». Étant donnée une séquence d'objectifs ω, nous pouvons l'encoder en CTL en utilisant la définition récursive $[|\cdot|]$ suivante :

$$[|a_j \Rsh^* a_i :: \varepsilon|] = \mathtt{EF}\ a_i \qquad [|a_j \Rsh^* a_i :: \omega|] = \mathtt{EF}\ (a_i \wedge [|\omega|]) \text{ si } \omega \neq \varepsilon \ .$$

En utilisant les opérateurs de raffinement définis dans la section précédente, nous établissons plusieurs conditions nécessaires ou suffisantes pour la concrétisabilité d'une séquence d'objectifs dans un contexte donné ς. Ces séquences d'objectifs peuvent être soit spécifiées par un utilisateur (afin de vérifier des propriétés temporelles du modèle), soit extraites depuis l'ensemble **BS** (en utilisant β, définition 9.14) afin de raffiner cet ensemble de séquences de bonds. En effet, si une séquence de bonds s'avère impossible à concrétiser dans ς, elle peut être ignorée pour toutes les analyses s'effectuant dans le cadre de ς.

Les approximations que nous construisons ont pour but d'être très rapides à calculer, surpassant ainsi le problème de l'explosion combinatoire de l'espace des états à explorer, inhérente à ce type d'analyse dynamique. Bien que non-concluantes dans certains cas, nous verrons dans le chapitre sur les applications biologiques (chapitre 11 page 169) que ces analyses se révèlent très efficaces lors de l'étude des dynamiques des grands réseaux de régulations biologiques, et promettent de supporter des analyses à très grande échelle.

Le reste de cette section est structuré comme suit. La sous-section 9.4.1 présente une première approximation sup. basée sur une analyse désordonnée des objectifs requis pour concrétiser une séquence d'objectifs donnée. La sous-section 9.4.2 raffine cette première approximation en exploitant la séquentialité des objectifs à concrétiser ; ensuite, la sous-section 9.4.3 prend avantage des contraintes d'ordre entre les occurrences des processus pour compléter ces approximations sup. Enfin, la sous-section 9.4.4 établit une approximation inf. de la décision d'atteignabilité de processus.

9.4.1 Approximation Sup. Désordonnée

Étant donnés un contexte ς et une séquence d'objectifs ω, nous obtenons que ω est concrétisable seulement si chaque objectif $\omega_n, n \in \mathbb{I}^\omega$ est concrétisable indépendamment dans le même contexte (proposition 9.1). En ce sens, nous pouvons approximer la concrétisabilité d'une séquence d'objectifs en appliquant récursivement la proposition 9.1 afin d'extraire les objectifs de la séquence donnée, et en utilisant l'opérateur de raffinement $\rho^\wedge$ afin d'étendre l'objectif en plusieurs séquences d'objectifs.

Proposition 9.1. $\gamma_\varsigma(\omega) \neq \emptyset \Longrightarrow \forall n \in \mathbb{I}^\omega, \gamma_\varsigma(\omega_n) \neq \emptyset$.

Démonstration. Par la définition 9.11 et la propriété 9.2, $\omega \preccurlyeq_{\mathrm{OS}} \omega_i$. □

Étant donné un objectif P, $\mathsf{minCont}_\varsigma(P)$ (définition 9.20) est l'ensemble des objectifs re-ciblés Q, avec $\mathsf{cible}(P) \neq \mathsf{cible}(Q)$ et $\mathsf{bond}(P) = \mathsf{bond}(Q)$, qui sont toujours dérivés lors d'une application récursive de $\rho^\wedge(P, \mathbf{BS}^\wedge(P))$ et de la proposition 9.1 (lemme 9.4). La procédure récursive vérifiant les conditions nécessaires pour la concrétisabilité d'un objectif est alors résumée par la proposition 9.2. Si P est concrétisable, alors il existe une exécution de cette procédure sans cycle, et où tous les objectifs testés possèdent au moins une solution (théorème 9.1).

Définition 9.20 ($\mathsf{minCont}_\varsigma : \mathbf{Obj} \mapsto \wp(\mathbf{Obj})$)**.**

$$\mathsf{minCont}_\varsigma(\star \Rsh^* a_j) = \{a_k \Rsh^* a_j \mid a_k \neq a_j \wedge \forall a_i \in \varsigma[a], a_k \in \mathsf{minCont}_\varsigma^{Obj}(a, a_i \Rsh^* a_j)\}$$
$$\mathsf{minCont}_\varsigma^{Obj} : \Sigma \times \mathbf{Obj} \mapsto \wp(\mathbf{Proc})$$
$$\mathsf{minCont}_\varsigma^{Obj}(a, P) = \emptyset \qquad \text{si } \mathbf{BS}^\wedge(P) = \emptyset, \text{ sinon,}$$
$$\mathsf{minCont}_\varsigma^{Obj}(a, P) = \{p \in \mathbf{Proc} \mid \forall ps \in \mathbf{BS}^\wedge(P), \exists q \in ps, p \in \mathsf{minCont}_\varsigma^{Proc}(a, q)\}$$
$$\mathsf{minCont}_\varsigma^{Proc} : \Sigma \times \mathbf{Proc} \mapsto \wp(\mathbf{Proc})$$
$$\mathsf{minCont}_\varsigma^{Proc}(a, b_i) = \begin{cases} \{b_i\} & \text{si } a = b, \\ \{p \in \mathbf{Proc} \mid \forall b_j \in \varsigma[b], & \\ \quad p \in \mathsf{minCont}_\varsigma^{Obj}(a, b_j \Rsh^* b_i)\} & \text{sinon.} \end{cases}$$

Lemme 9.4. $a_k \Rsh^* a_j \in \mathsf{minCont}_\varsigma(\star \Rsh^* a_j) \Longrightarrow \gamma_\varsigma(\star \Rsh^* a_j) = \gamma_\varsigma(\star \Rsh^* a_k :: a_k \Rsh^* a_j)$.

Démonstration. Par induction sur $\mathsf{minCont}_\varsigma$, a_k apparaît dans tous les raffinements récursifs de $\star \Rsh^* a_j$ par $\rho^\wedge$. □

Proposition 9.2. $\gamma_\varsigma(P) \neq \emptyset \Longrightarrow \exists ps \in \mathbf{BS}_\varsigma^\wedge(P), \forall p \in ps, \gamma_\varsigma(\star \Rsh^* p) \neq \emptyset$, *et* $\forall Q \in \mathsf{minCont}_\varsigma(P)$, $\gamma_\varsigma(Q) \neq \emptyset$.

Démonstration. Par le lemme 9.2 page 146, le lemme 9.4 et la proposition 9.1. □

Théorème 9.1 (Approximation sup. désordonnée)**.** $\gamma_\varsigma(\omega) \neq \emptyset \Longrightarrow \forall n \in \mathbb{I}^\omega$, *il existe une application récursive finie de la proposition 9.2 pour* $\gamma_\varsigma(\omega_n) \neq \emptyset$ *telle que pour tous les objectifs testés* P, $\mathbf{BS}_\varsigma^\wedge(P) \neq \emptyset$.

Démonstration. Par induction, si $\gamma_\varsigma(P) \neq \emptyset$ requiert $\gamma_\varsigma(P') \neq \emptyset, P' \neq P$, et si $\gamma_\varsigma(P') \neq \emptyset$ requiert à son tour $\gamma_\varsigma(P) \neq \emptyset$, alors $\gamma_\varsigma(P) = \emptyset$; ainsi, par la proposition 9.1, il existe une application récursive finie de la proposition 9.2 pour $\gamma_\varsigma(\omega_n) \neq \emptyset, \forall n \in \mathbb{I}^\omega$. Enfin, la proposition 9.2 implique la non-vacuité de $\mathbf{BS}_\varsigma^\wedge(P)$ pour chaque objectif P testé. □

Implémentation.

Étant donnés un contexte ς et une séquence d'objectifs ω, nous définissons une structure abstraite $\mathcal{A}_\varsigma^\omega$ (définition 9.21) qui mime les relations entre les objectifs lors de l'exécution de la proposition 9.2 (lemme 9.5). $\mathcal{A}_\varsigma^\omega$ rassemble trois relations $\mathsf{Req}_\varsigma^\omega$, $\mathsf{Sol}_\varsigma^\omega$ et $\mathsf{Cont}_\varsigma^\omega$), respectivement les requis, les solutions et les continuités minimales (ou re-ciblages minimaux).

Définition 9.21 ($\mathcal{A}^{\omega}_{\varsigma}$)**.** Étant donnés un contexte ς et une séquence d'objectifs ω, nous définissons la structure abstraite $\mathcal{A}^{\omega}_{\varsigma} = (\mathsf{Req}^{\omega}_{\varsigma}, \mathsf{Sol}^{\omega}_{\varsigma}, \mathsf{Cont}^{\omega}_{\varsigma})$, où $\mathsf{Req}^{\omega}_{\varsigma}$, $\mathsf{Sol}^{\omega}_{\varsigma}$ et $\mathsf{Cont}^{\omega}_{\varsigma}$ sont définis par :

$$\begin{aligned}
\mathsf{Req}^{\omega}_{\varsigma} =& \{(a_i, a_j \Rsh^{*} a_i) \in \mathbf{Proc} \times \mathbf{Obj} \mid a_j \in \varsigma[a] \wedge (\exists (P, ps) \in \mathsf{Sol}^{\omega}_{\varsigma}, a_i \in ps \\
& \qquad \vee \exists n \in \mathbb{I}^{\omega}, \mathsf{bond}(\omega) = a_i)\} \\
\mathsf{Sol}^{\omega}_{\varsigma} =& \{(P, ps) \in \mathbf{Obj} \times \wp(\mathbf{Proc}) \mid \exists (a_i, P) \in \mathsf{Req}^{\omega}_{\varsigma} \wedge ps \in \mathbf{BS}^{\wedge}(P)\} \\
\mathsf{Cont}^{\omega}_{\varsigma} =& \{(P, Q) \in \mathbf{Obj} \times \mathbf{Obj} \mid \exists (P, ps) \in \mathsf{Sol}^{\omega}_{\varsigma} \wedge Q \in \mathsf{minCont}_{\varsigma}(p)\} \quad .
\end{aligned}$$

Lemme 9.5. *Étant donné un objectif P référencé dans $\mathcal{A}^{\omega}_{\varsigma}$,*

- $\omega' \oplus P \in \rho^{\wedge}(P, \mathbf{BS}^{\wedge}(P)) \iff (P, \{\mathsf{bond}(\omega'_n) \mid n \in \mathbb{I}^{\omega'}\}) \in \mathsf{Sol}^{\omega}_{\varsigma} \wedge \forall n \in \mathbb{I}^{\omega'}, a_j = \mathsf{bond}(\omega'_n)$, $\forall a_i \in \varsigma[a], (a_j, a_i \Rsh^{*} a_j) \in \mathsf{Req}^{\omega}_{\varsigma}$;
- $Q \in \mathsf{minCont}_{\varsigma}(P) \iff (P, Q) \in \mathsf{Cont}^{\omega}_{\varsigma}$.

Démonstration. Par construction de $\mathcal{A}^{\omega}_{\varsigma}$. □

$\mathcal{A}^{\omega}_{\varsigma}$ possède une structure de graphe, contenant potentiellement des cycles. Nous remarquons que, comme $|\mathbf{Obj}| = \sum_{a \in \Sigma} |L_a|^2$, la taille des ensembles $\mathsf{Req}^{\omega}_{\varsigma}$ et $\mathsf{Cont}^{\omega}_{\varsigma}$ est polynomiale suivant le nombre de processus dans les Frappes de Processus. La taille de l'ensemble $\mathsf{Sol}^{\omega}_{\varsigma}$ dépend également de la cardinalité de $\mathbf{BS}^{\wedge}$, bornée par le nombre de combinaisons de $|L_a|$ parmis l'ensemble des processus $\binom{|\mathsf{Proc}|}{|L_a|}$, où a est une sorte.

Enfin, l'algorithme 9.1 détaille le calcul de la décision du théorème 9.1.

Algorithme 9.1 (Approximation inf. désordonnée)**.** Étant donnés un contexte ς, une séquence d'objectifs $\omega \in \mathbf{OS}$ et la structure abstraite $\mathcal{A}^{\omega}_{\varsigma}$:

1. Initialiser $\Theta = \{P \in \mathbf{Obj} \mid (P, \emptyset) \in \mathsf{Sol}^{\omega}_{\varsigma}\}$.
2. Répéter jusqu'à atteinte d'un point fixe :
   (a) $\Upsilon = \{p \in \mathbf{Proc} \mid \exists P \in \Theta, (p, P) \in \mathsf{Req}^{\omega}_{\varsigma}\}$;
   (b) $\Theta = \{P \in \mathbf{Obj} \mid \exists (P, ps) \in \mathsf{Sol}^{\omega}_{\varsigma}, ps \subseteq \Upsilon \wedge \forall (P, Q) \in \mathsf{Cont}^{\omega}_{\varsigma}, Q \in \Theta\}$.
3. $\gamma_{\varsigma}(\omega) \neq \emptyset \Longrightarrow \forall n \in \mathbb{I}^{\omega}, \exists P \in \Theta, \mathsf{cible}(P) \in \varsigma \wedge \mathsf{bond}(P) = \mathsf{bond}(\omega_n)$.

Complexité. Le calcul de $\mathcal{A}^{\omega}_{\varsigma}$ est effectué en ajoutant itérativement les processus requis et objectifs associés. Les étapes de l'algorithme 9.1 sont calculées polynomialement selon la taille de $\mathcal{A}^{\omega}_{\varsigma}$. Ainsi, en mettant de côté le calcul de l'ensemble $\mathbf{BS}^{\wedge}$, l'approximation sup. proposée peut être acquise en un nombre d'opérations polynomial en la taille de la structure abstraite.

Exemples.

La figure 9.4 représente graphiquement la structure abstraite extraite de l'exemple de Frappes de Processus de la figure 9.1 page 138 pour une séquence d'objectifs et un contexte particuliers. Trois types de nœuds sont mis en relation dans ce graphe : les objectifs, les solutions et les processus. Un processus d_2 est lié à un objectif $d_0 \Rsh^{*} d_2$ si et seulement si d_0 appartient au contexte de la structure abstraite. Chaque solution d'un objectif P donné (c.-à-d. chaque ensemble de processus appartenant à $\mathbf{BS}(P)$) est liée à tous les processus la composant. Deux types particuliers de nœuds apparaissent alors : les solutions sans enfants, dénotant les solutions triviales ; et les objectifs sans enfants, dénotant les objectifs irréalisables (pas de séquence de bonds existante).

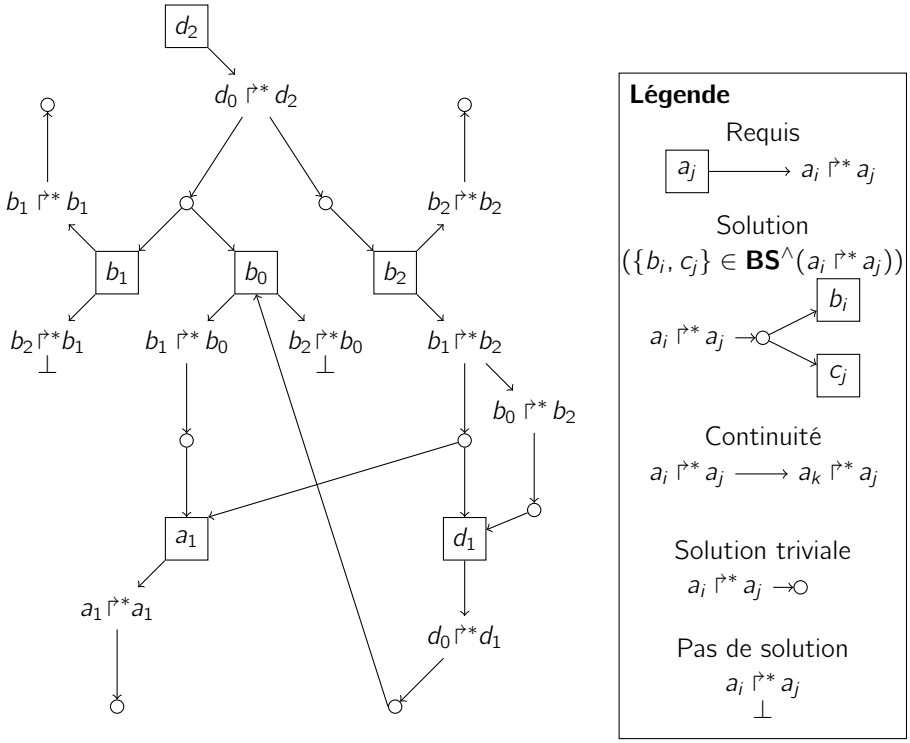


Figure 9.4 – Représentation graphique de la structure abstraite $\mathcal{A}^{\omega}_{\varsigma}$ extraite des Frappes de Processus de la figure 9.1 page 138, avec $\omega = d_0 \Rsh^* d_2$ et $\varsigma = \langle a_1, b_1, b_2, c_0, d_0 \rangle$.

Ainsi, partant d'un objectif à réaliser, ce graphe renseigne les étapes nécessaires permettant d'atteindre le processus voulu. Dans le cas présenté dans la figure 9.4, le théorème 9.1 est satisfait : partant de d_2, il existe bien un parcours sans cycle se terminant sur des solutions triviales, et où tous les enfants d'une solution sont sélectionnés, et au moins une solution par objectif est sélectionnée.

La figure 9.5 applique le théorème 9.1 à l'exemple de Frappes de Processus de la figure 9.1 pour la décision de concrétisabilité de l'objectif $d_1 \Rsh^* d_2$. Dans ce cas, la condition nécessaire n'est pas satisfaite : d_2 n'est pas atteignable depuis le contexte imposé. En effet, toutes les solutions de l'objectif $d_1 \Rsh^* d_2$ nécessitent l'atteinte du processus a_0, ce qui est impossible depuis a_1 (aucune solution, $\mathbf{BS}(a_1 \Rsh^* a_0) = \emptyset$).

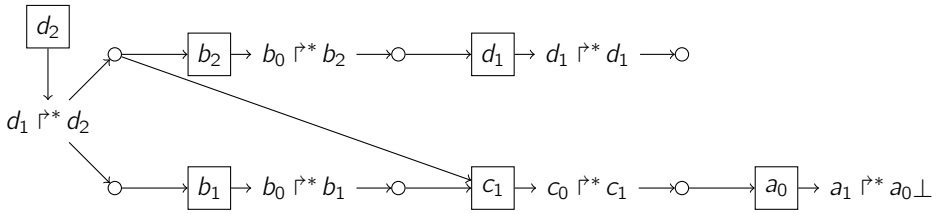


Figure 9.5 – Structure abstraite $\mathcal{A}^{\omega}_{\varsigma}$ pour l'exemple de Frappes de Processus de la figure 9.1 avec $\omega = d_1 \Rsh^* d_2$ et $\varsigma = \langle a_1, b_0, c_0, d_1 \rangle$. D'après le théorème 9.1, l'objectif $d_1 \Rsh^* d_2$ *n'est pas concrétisable*.

9.4.2 Approximation sup. ordonnée

Dans cette sous-section, nous exploitons l'ordre des objectifs dans une séquence d'objectifs afin d'augmenter les cas concluants pour l'approximation sup. de sa concrétisabilité.

Étant donné un objectif P qui n'est pas trivial dans le contexte ς donné, nous dénotons par $\mathsf{fins}(P)$ l'ensemble des processus vers lequel peut mener le jeu d'un scénario concrétisant P (définition 9.22, proposition 9.3). Cet ensemble est calculé depuis $\mathbf{BS}(P)$ en prenant le frappeur et le bond de la dernière action pour chaque séquence de bonds.

Définition 9.22 ($\mathsf{fins}_\varsigma : \mathbf{Obj} \mapsto \wp(\wp(\mathbf{Proc}))$)**.**

$$\mathsf{fins}_\varsigma(\star \Rsh^* a_i) = \{\mathsf{fin}(h) \mid \exists a_j \in \varsigma[a], \exists \zeta \in \mathbf{BS}(a_j \Rsh^* a_i), \zeta \neq \varepsilon \wedge h = \zeta_{|\zeta|}\} \ .$$

Proposition 9.3. $\gamma_\varsigma(\star \Rsh^* a_i) \neq \emptyset \wedge a_i \notin \varsigma[a] \implies \exists \delta \in \gamma_\varsigma(\star \Rsh^* a_i), \exists eps \in \mathsf{fins}_\varsigma(\star \Rsh^* a_i)$ *tel que* $eps \subseteq \mathsf{fin}(\delta)$.

Démonstration. Comme $a_i \notin \varsigma[a]$, il existe $n \in \mathbb{I}^\delta$ tel que $\mathsf{bond}(\delta_n) = a_i$; ainsi $\delta_{1..n} \in \gamma_\varsigma(\star \Rsh^* a_i)$ et $\mathsf{fin}(\delta_n) \subseteq \mathsf{fin}(\delta_{1..n})$. Par la définition 9.22, $\mathsf{fin}(\delta_n) \in \mathsf{fins}_\varsigma(\star \Rsh^* a_i)$. □

Étant donnée une structure abstraite $\mathcal{A}^\omega_\varsigma \in \mathbb{A}$, $\mathsf{procs}(\mathcal{A}^\omega_\varsigma)$ est l'ensemble des processus référencés dans $\mathcal{A}^\omega_\varsigma$ (définition 9.23) et nous dérivons la proposition 9.4. Nous définissons alors $\mathsf{maxprocs}_\varsigma(\omega)$ (définition 9.24) comme étant l'ensemble des processus présents dans la structure abstraite dont le contexte a été saturé (c.-à-d., tel que $\varsigma \Cap \mathsf{procs}(\mathcal{A}^\omega_\varsigma) = \varsigma$). Nous effectuons une optimisation particulière pour $\omega = P$: dans un tel cas, le processus $\mathsf{bond}(P)$ peut être ignoré par le contexte.

Définition 9.23 ($\mathsf{procs} : \mathbb{A} \mapsto \wp(\mathbf{Proc})$)**.**

$$\begin{aligned}\mathsf{procs}((\mathsf{Sol}^\omega_\varsigma, \mathsf{Req}^\omega_\varsigma, \mathsf{Cont}^\omega_\varsigma)) = \{p \in \mathbf{Proc} \mid & \exists (P, ps) \in \mathsf{Sol}^\omega_\varsigma, p \in ps \\ & \vee\ p = \mathsf{cible}(P) \\ & \vee\ (P \neq \omega \Rightarrow p = \mathsf{bond}(P))\} \ .\end{aligned}$$

Proposition 9.4. *Pour chaque objectif $Q \neq \omega$ testé par la proposition 9.2 au cours de la vérification du théorème 9.1 pour $\gamma_\varsigma(\omega)$,* $\mathsf{bond}(Q) \in \mathsf{procs}(\mathcal{A}^\omega_\varsigma)$ *et* $\varsigma[\Sigma(Q)] \subseteq \mathsf{procs}(\mathcal{A}^\omega_\varsigma)$.

Démonstration. D'après le lemme 9.5. □

Définition 9.24 ($\mathsf{maxprocs}_\varsigma : \mathbf{OS} \mapsto \wp(\mathbf{Proc})$)**.**

$$\mathsf{maxprocs}_\varsigma(\omega) = \mathsf{procs}(\lceil \mathcal{A}^\omega_\varsigma \rceil) \qquad \text{où } \lceil \mathcal{A}^\omega_\varsigma \rceil = \mathsf{pppf}\{\mathcal{A}^\omega_\varsigma\}\left(\mathcal{A}^\omega_\varsigma \mapsto \mathcal{A}^\omega_{\varsigma \Cap \mathsf{procs}(\mathcal{A}^\omega_\varsigma)}\right) \ .$$

En intersectant $\mathsf{maxprocs}_\varsigma(\omega)$ et $\mathsf{fins}_\varsigma(\omega_1)$, nous obtenons un contexte dans lequel la concrétisabilité de $\omega_{2..|\omega|}$ doit satisfaire le théorème 9.1, si ω est concrétisable dans ς. Ceci est établi par le théorème 9.2, qui produit un raffinement direct du théorème 9.1 en prenant en compte la séquentialité des objectifs dans ω.

Théorème 9.2 (Approximation sup. ordonnée)**.** *Étant donnés un contexte* ς *et* $\omega = P :: \omega' \in$ **OS** *tels que* $\mathsf{bond}(P) \notin \varsigma$, $\gamma_\varsigma(P :: \omega') \neq \emptyset \Longrightarrow$ *le théorème 9.1 est satisfait avec* $\gamma_{max\varsigma}(\omega') \neq \emptyset$, *où* $max\varsigma = \varsigma \Cap \mathsf{maxprocs}_\varsigma(\omega) \Cap eps$ *et* $eps \in \mathsf{fins}_\varsigma(P)$.

Démonstration. $\delta \in \gamma_\varsigma(P :: \omega') \Rightarrow \exists n \in \mathbb{I}^\delta$ tel que $\mathsf{bond}(\delta_n) = \mathsf{bond}(P), eps \subseteq \mathsf{fin}(\delta_{1..n})$, et $\delta_{n+1..|\delta|} \in \gamma_{\varsigma'}(\omega')$ avec $\varsigma' = \varsigma \Cap \mathsf{support}(\delta)$. Ainsi, le théorème 9.1 est satisfait avec $\gamma_{\varsigma'}(\omega') \neq \emptyset$. Soit Q un objectif testé par la proposition 9.2 lors de la vérification du théorème 9.1 avec $\gamma_{max\varsigma}(\omega') \neq \emptyset$. Par la proposition 9.2, $\mathsf{cible}(Q) \in max\varsigma$. Si $\mathsf{cible}(Q) \in eps$ alors $\mathsf{cible}(Q) \in \varsigma'$, et donc par hypothèse, la proposition 9.2 est satisfaite. Dans la cas où $\mathsf{cible}(Q) \notin eps$, par la définition 9.24 et la proposition 9.4, $\mathsf{bond}(Q) \in \mathsf{maxprocs}_\varsigma(\omega)$, donc la proposition 9.2 est vérifiée (Q est alors un objectif trivial). □

En définissant $\mathsf{maxCtx}(\varsigma, \omega, n)$ comme le contexte maximum après a résolution de $\omega_{1..n}$ (définition 9.25), le précédent théorème peut être directement étendu au corollaire 9.1.

Définition 9.25 ($\mathsf{maxCtx} : \mathbf{Ctx} \times \mathbf{OS} \times \mathbb{N} \mapsto \mathbf{Ctx}$)**.** (nous supposons $n \in \{0\} \cup \mathbb{I}^n$) :

$$\mathsf{maxCtx}(\varsigma, \omega, n) = \begin{cases} \varsigma & \text{si } n = 0, \\ \varsigma \Cap \mathsf{maxprocs}_\varsigma(\omega) & \text{si } \mathsf{bond}(\omega_n) \in \mathsf{maxCtx}(\varsigma, \omega, n-1) \\ \varsigma \Cap \mathsf{maxprocs}_\varsigma(\omega) \Cap eps & \text{sinon, où } eps \in \mathsf{fins}_\varsigma(\omega_n) \end{cases}$$

Corollaire 9.1. $\gamma_\varsigma(\omega) \neq \emptyset \Longrightarrow \forall n \in \mathbb{I}^\omega, \gamma_{max\varsigma}(\omega_n) \neq \emptyset$, *avec* $max\varsigma = \mathsf{maxCtx}(\varsigma, \omega, n-1)$.

Implémentation.

Le calcul de $\mathsf{maxprocs}_\varsigma(\omega)$ requiert au plus $|\mathbf{Proc}|$ itérations, donnant un nombre d'étapes polynomial suivant le nombre de processus. Concernant le calcul de $\mathsf{fins}_\varsigma(\star \Rsh^* a_i)$, il peut soit être dérivé des calculs précédents de $\mathbf{BS}(\star \Rsh^* a_i)$; ou, si **BS** est trop coûteux à calculer, il peut être approximé par $\{\{\mathsf{bond}(P)\}\}$, ou bien par $\{\mathsf{fin}(h) \mid h \in \mathcal{H} \wedge \mathsf{bond}(h) = a_i \wedge \exists a_j \in \varsigma[a], \exists ps \in \mathbf{BS}^\wedge(a_j \Rsh^* a_i), \mathsf{cible}(h) \in ps\}$.

Exemple.

Étant données les Frappes de Processus définies dans la figure 9.6, avec $\varsigma = \langle a_1, b_0 \rangle$, le théorème 9.1 n'est pas concluant sur la concrétisabilité de $\omega = a_1 \Rsh^* a_0 :: b_0 \Rsh^* b_1$. En effet, la structure abstraite correspondante, représentée dans la figure 9.6(droite) amène à conclure que ω satisfait les conditions du théorème 9.1, rendant non-concluante la décision de concrétisabilité de ω.

Le théorème 9.2 permet alors de prendre en compte la séquentialité des objectifs dans ω pour raffiner l'approximation sup. de sa concrétisabilité. Il apparaît qu'après résolution de l'objectif $a_1 \Rsh^* a_0$, les processus a_0 et b_0 sont présent. En effet, d'après les Frappes de Processus de la figure 9.6(gauche), $\mathsf{fins}_\varsigma(a_1 \Rsh^* a_0 = \{\{a_0, b_0\}\}$: le bond en a_0 est conditionné par la présence du processus b_0, donc juste après l'atteinte de a_0, b_0 est également présent.

Nous pouvons alors déduire que la condition nécéssaire, établie par le théorème 9.2, imposant que l'objectif $b_0 \Rsh^* b_1$ soit concrétisable dans le contexte $max\varsigma = \mathsf{maxCtx}(\varsigma, \omega, 1) = \langle a_0, b_0 \rangle$, n'est

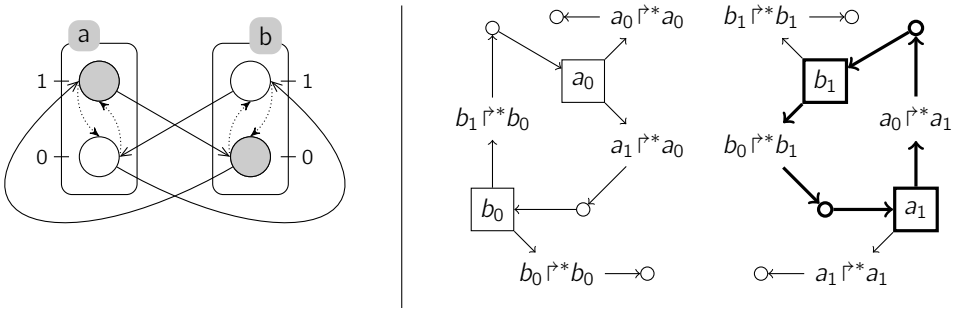


Figure 9.6 – (droite) Structure abstraite saturée $\lceil \mathcal{A}^{\omega}_{\varsigma} \rceil$ des Frappes de Processus de (gauche) avec $\varsigma = \langle a_1, b_0 \rangle$ et $\omega = a_1 \Rsh^* a_0 :: b_0 \Rsh^* b_1$. La partie en ligne épaisse représente la structure abstraite calculée pour la décision du théorème 9.2 : ce théorème permet de conclure que cette séquence d'objectifs n'est pas concrétisable.

pas satisfaite. En effet, la structure abstraite $\mathcal{A}^{b_0 \Rsh^* b_1}_{max_\varsigma}$ représentée en gras dans la figure 9.6(droite) forme un unique cycle entre b_1 et a_1 : les conditions du théorème 9.1 ne sont donc pas satisfaites, la concrétisation de ω dans ς est ainsi impossible.

9.4.3 Approximation Sup. avec Contraintes sur l'Ordre d'Occurrence des Processus

Un objectif $a_i \Rsh^* a_j$ ne possédant aucune solution ($\mathbf{BS}^{\wedge}(a_i \Rsh^* a_j) = \emptyset$) informe que le processus a_j n'apparaît jamais après le processus a_i. Cette contrainte d'ordre entre les occurrences des processus est référée par $a_j \lhd a_i$ (définition 9.26, propriété 9.5).

Définition 9.26 ($\lhd$)**.** La relation binaire $\lhd \subset \mathbf{Proc} \times \mathbf{Proc}$ est un pré-ordre partiel tel que $a_j \lhd a_i$ (c.-à-d. $(a_j, a_i) \in \lhd$) si et seulement si il n'existe aucun scénario où a_j apparaît après a_i : $a_j \lhd a_i \iff \nexists \delta \in \mathbf{Sce}$ tel que $\exists n, m \in \mathbb{I}^{\delta}, n \leq m, \mathsf{cible}(\delta_n) = a_i \wedge \mathsf{bond}(\delta_m) = a_j$.

Propriété 9.5 (Découverte d'une contrainte d'ordre)**.** $\mathbf{BS}(a_i \Rsh^* a_j) = \emptyset \Longrightarrow a_j \lhd a_i$.

Ayant pris connaissance de plusieurs contraintes d'ordre entre les occurrences des processus, nous voulons vérifier que certaines séquences d'objectifs ne contredisent pas ces contraintes. Ceci peut être effectué en calculant les processus apparaissant dans toutes les résolutions possibles d'un objectif : étant donnée une séquence d'objectifs ω, si un processus a_i est requis par ω_n et un processus a_j par ω_m, $n < m$, alors la contrainte $a_j \lhd a_i$ ne doit pas exister. Ce raisonnement est illustré dans la figure 9.7.

Similairement à $\mathsf{minCont}_{\varsigma}$ (définition 9.20), $\mathsf{minProc}_{\varsigma}(P)$ réfère à l'ensemble des processus de *toute* sorte apparaissant dans tous les raffinements de P dans le contexte ς (définition 9.27, lemme 9.6). En utilisant la définition précédente de maxCtx (définition 9.25), nous définissons $\mathsf{minProc}(\varsigma, \omega, n)$ (définition 9.28) l'ensemble des processus apparaissant au cours de la résolution de ω_n après avoir résolu $\omega_{1..n-1}$. À partir de cette définition, le théorème 9.3 établit formellement l'approximation sup. illustrée dans la figure 9.7.

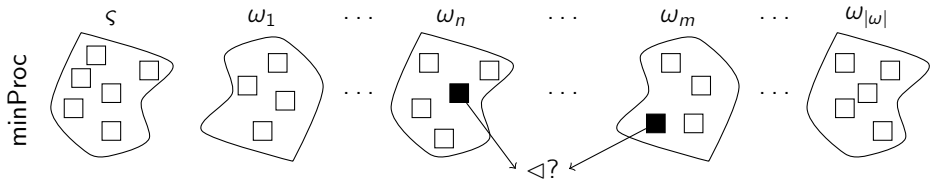


Figure 9.7 – Illustration de la méthode développée dans la sous-section 9.4.3 pour approximer supérieurement la concrétisabilité d'une séquence d'objectifs ω dans le contexte ς. Pour chaque objectif de la séquence, l'ensemble minimal des processus apparaissant lors de sa résolution (représentés par des carrés) est calculé en utilisant minProc (définition 9.28) ; la séquence n'est pas concrétisable si il existe deux processus (ici, en noir) apparaissant au cours de la résolution d'objectifs distincts contredisant une contrainte d'ordre d'occurrence entre les processus dans $\lhd$.

Définition 9.27 ($\text{minProc}_\varsigma : \textbf{Obj} \mapsto \wp(\textbf{Proc})$)**.** Étant donné un contexte ς,

$$\begin{aligned}
&\text{minProc}_\varsigma(\star \Rsh^* a_i) = \{p \in \textbf{Proc} \mid \forall a_j \in \varsigma[a], \\
&\qquad\qquad \textbf{BS}(a_j \Rsh^* a_i) \neq \emptyset \Rightarrow p \in \text{minProc}_\varsigma^{Obj}(a_j \Rsh^* a_i)\} \\
&\text{minProc}_\varsigma^{Obj} : \textbf{Obj} \mapsto \wp(\textbf{Proc}) \\
&\text{minProc}_\varsigma^{Obj}(a_j \Rsh^* a_i) = \{a_i\} \cup \{p \in \textbf{Proc} \\
&\qquad\qquad \mid \forall ps \in \textbf{BS}^\wedge(a_j \Rsh^* a_i), \exists q \in ps, p \in \text{minProc}_\varsigma(\star \Rsh^* q) \\
&\qquad\qquad \vee \exists a_k \Rsh^* a_i \in \text{minCont}_\varsigma^{Obj}(a, a_j \Rsh^* a_i), \\
&\qquad\qquad\quad p \in \text{minProc}_\varsigma^{Obj}(a_j \Rsh^* a_k) \cup \text{minProc}_\varsigma^{Obj}(a_k \Rsh^* a_i)))\} \ .
\end{aligned}$$

Lemme 9.6. $\forall \delta \in \gamma_\varsigma(P), \forall p \in \text{minProc}_\varsigma(P), p \in \delta$.

Démonstration. Par induction sur minProc_ς, p apparaît dans tous les raffinements récursifs de P par $\rho^\wedge$. □

Définition 9.28 ($\text{minProc} : \textbf{Ctx} \times \textbf{OS} \times \mathbb{N} \mapsto \wp(\textbf{Proc})$)**.** Nous supposons $n \in \{0\} \cup \mathbb{I}^\omega$:

$$\text{minProc}(\varsigma, \omega, n) = \begin{cases} \{a_i \in \varsigma\} & \text{si } n = 0 \\ \text{minProc}_{max\varsigma}(\omega_n) & \text{sinon, avec } max\varsigma = \text{maxCtx}(\varsigma, w, n-1) \ . \end{cases}$$

Théorème 9.3 (Approximation sup. ordonnée raffinée par $\lhd$)**.** $\gamma_\varsigma(\omega) \neq \emptyset \Longrightarrow \nexists n, m \in \{0\} \cup \mathbb{I}^\omega, n < m, \exists p \in \text{minProc}(\varsigma, \omega, n), \exists q \in \text{minProc}(\varsigma, \omega, m), q \lhd p$.

Démonstration. Par le lemme 9.6, le corollaire 9.1 et la définition 9.28. □

Implémentation.

L'implémentation est vraiment similaire à celle présentée dans la sous-section précédente. La découverte de $\lhd$ est effectuée linéairement selon la taille de la structure abstraite saturée $\lceil \mathcal{A}_\varsigma^\omega \rceil$.

Exemple.

Soient les Frappes de Processus définies dans la figure 9.8 et leur structure abstraite saturée $\lceil \mathcal{A}^{\omega}_{\varsigma} \rceil$, avec le contexte $\varsigma = \langle a_1, b_1, z_1 \rangle$ et la séquence d'objectifs $\omega = z_1 \Rsh^* z_2 :: z_2 \Rsh^* z_1 :: z_1 \Rsh^* z_2$ pour laquelle la concrétisabilité doit être décidée.

La satisfaction des conditions du théorème 9.3 se base sur celles du théorème 9.2 page 152 : partant du contexte ς, un procédé itératif calcule le contexte maximal requis pour la résolution de chaque objectif dans ω. Ce contexte est donné par $\mathsf{maxCtx}(\varsigma, \omega, n)$ où n est l'indice de l'objectif dans la séquence ω (définition 9.25). Pour chacun de ces objectifs, $\mathsf{minCont}(\varsigma, \omega, n)$ (définition 9.27) extrait alors les processus nécessaires à leur atteinte. Le tableau suivant récapitule les valeur de minCont et maxCtx en partant de $n = 0$, représentant le contexte ς de départ, puis en itérant sur les objectifs de la séquence ω.

	$n = 0 - \varsigma$	$n = 1 - z_1 \Rsh^* z_2$	$n = 2 - z_2 \Rsh^* z_1$	$n = 3 - z_1 \Rsh^* z_2$
$\mathsf{minProc}(\varsigma, \omega, n)$	a_1, b_1, z_1	$\mathbf{a}_0, a_1, b_0, z_0, z_2$	b_1, z_1	$a_0, \mathbf{a}_1, b_0, z_0, z_2$
$\mathsf{maxCtx}(\varsigma, \omega, n)$	a_1, b_1, z_1	a_0, a_1, b_0, z_2	a_0, a_1, b_1, z_1	-

L'analyse des Frappes de Processus représentées dans la figure 9.8(haut) indique que $a_1 \lhd a_0$ (il n'est pas possible d'atteindre a_1 depuis a_0). Or, le tableau précédent indique que le processus a_0 est requis pour l'objectif $\omega_1 = z_1 \Rsh^* z_2$ et le processus a_1 pour l'objectif $\omega_3 = z_1 \Rsh^* z_2$. Ainsi, d'après le théorème 9.3, la séquence d'objectifs ω n'est pas concrétisable dans ς.

9.4.4 Approximation Inf.

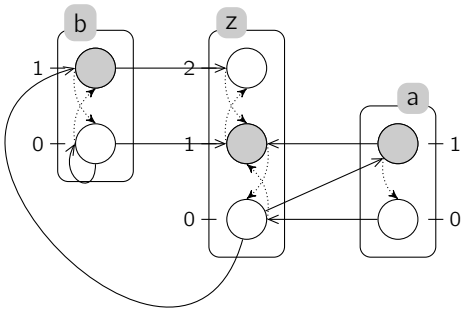
La procédure d'approximation inf. présentée dans cette sous-section prend avantage d'une variante de la structure abstraite présentée précédemment pour les approxamations sup. Si certaines conditions sont vérifiées sur cette structure abstraite, alors, nous montrons qu'un scénario concrétisant la séquence d'objectifs donnée existe. La construction proposée du scénario est effectuée par une résolution dite *de haut en bas* : étant donnée une séquence d'objectifs ω, nous construisons dans un premier temps le scénario concrétisant ω_1 en ignorant la résolution de la séquence d'objectifs $\omega_{2..|\omega|}$ (et ignorant ainsi tout entrelacement d'objectifs pouvant être nécessaire à la concrétisation complète de ω). Comme établi par les opérateurs de raffinement dans la sous-section 9.3.4, la concrétisation de ω_1 induit la concrétisation d'un raffinement de ω_1, résultant en une procédure récursive de construction du scénario.

Nous présentons au préalable une définition alternative de l'ensemble des scénarios concrétisant une séquence d'objectifs ω dans un contexte ς : $\ell_\varsigma(\omega)$ est vide sauf si, pour chaque état $s \in L$ inclus dans ς, il existe un scénario $\delta \in \gamma_\varsigma(\omega)$ tel que δ est jouable dans s ; dans ce cas, $\ell_\varsigma(\omega) = \gamma_\varsigma(\omega)$ (définition 9.29). La propriété 9.6 est directement dérivée de cette définition et l'extension de ℓ_ς pour un ensemble de séquences d'objectifs est donnée dans la définition 9.30.

Définition 9.29 ($\ell_\varsigma : \mathbf{OS} \mapsto \wp(\mathbf{Sce})$)**.**

$$\ell_\varsigma(\omega) = \begin{cases} \gamma_\varsigma(\omega) & \text{si } \forall s \in L, s \subseteq \varsigma, \exists \delta \in \gamma_\varsigma(\omega), \mathsf{support}(\delta) \subseteq s \\ \emptyset & \text{sinon.} \end{cases}$$

Propriété 9.6. $\varsigma' \subseteq \varsigma \wedge \ell_\varsigma(\omega) \neq \emptyset \Longrightarrow \ell_{\varsigma'}(\omega) \neq \emptyset$.



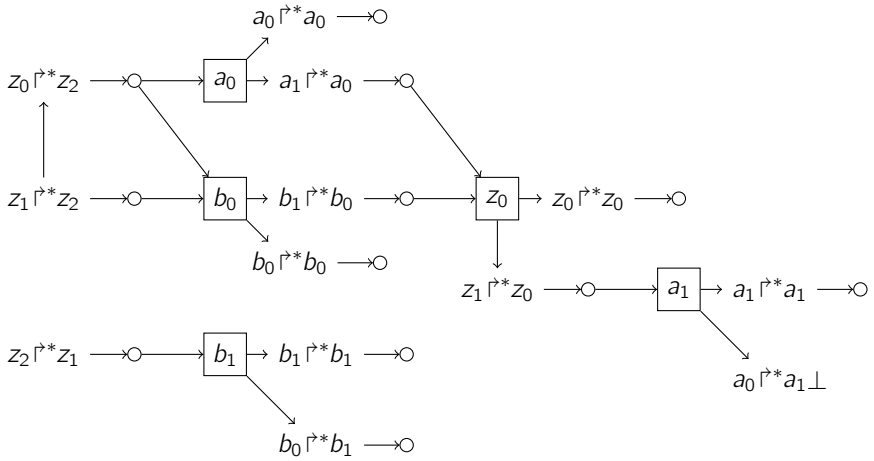


Figure 9.8 – (bas) Structure abstraite saturée $\lceil \mathcal{A}^\omega_\varsigma \rceil$ des Frappes de Processus (haut) avec $\varsigma = \langle a_1, b_1, z_1 \rangle$ et $\omega = z_1 \Rsh^* z_2 :: z_2 \Rsh^* z_1 :: z_1 \Rsh^* z_2$. Le théorème 9.3 conclut que cette séquence d'objectifs n'est pas concrétisable.

Définition 9.30 ($\ell_\varsigma : \wp(\mathbf{OS}) \mapsto \wp(\mathbf{Sce})$)**.** $\ell_\varsigma(\Omega) = \{\delta \in \ell_\varsigma(\omega) \mid \omega \in \Omega\}$

Étant donné un objectif P, $\mathsf{maxCont}_\varsigma(\Sigma(P), P)$ (définition 9.31) est l'ensemble des processus de sorte $\Sigma(P)$ pouvant être rencontrés lors de la résolution de P. Nous définissons alors la structure abstraite saturée $\lceil \mathcal{B}^\omega_\varsigma \rceil = (\lceil \mathsf{Req}^\omega_\varsigma \rceil, \lceil \mathsf{Sol}^\omega_\varsigma \rceil, \lceil \mathsf{Cont}^\omega_\varsigma \rceil)$ (définition 9.32) similairement à $\lceil \mathcal{A}^\omega_\varsigma \rceil$ (définition 9.21), excepté pour $\lceil \mathsf{Sol}^\omega_\varsigma \rceil$ pour lequel nous pouvons sélectionner arbitrairement des séquences de bonds pour la résolution d'un objectif, et pour $\lceil \mathsf{Cont}^\omega_\varsigma \rceil$ qui reflète $\mathsf{maxCont}_\varsigma$ au lieu de $\mathsf{minCont}_\varsigma$. Il apparaît que si $\lceil \mathcal{B}^\omega_\varsigma \rceil$ ne contient aucun cycle, et si tous les objectifs référencés possèdent au moins une solution, alors une résolution *de haut en bas* est possible pour chacun des objectifs référencés à partir de tous les états de ς. Ceci est établi par le théorème 9.4 qui produit une condition suffisante pour la concrétisation d'une séquence d'objectifs dans un contexte donné.

Définition 9.31 ($\mathsf{maxCont}_\varsigma : \Sigma \times \mathbf{Obj} \mapsto \wp(\mathbf{Proc})$)**.**

$$\mathsf{maxCont}_\varsigma(a, P) = \{p \in \mathbf{Proc} \mid \exists ps \in \mathbf{BS}^\wedge(P), \exists b_i \in ps, b = a \wedge p = b_i \\ \vee\, b \neq a \wedge p \in \mathsf{maxCont}_\varsigma(a, b_j \Rsh^* b_i) \wedge b_j \in \varsigma[b]\} \ .$$

Définition 9.32 ($\lceil\mathcal{B}^\omega_\varsigma\rceil$)**.** La structure abstraite $\lceil\mathcal{B}^\omega_\varsigma\rceil = (\lceil\mathsf{Req}^\omega_\varsigma\rceil, \lceil\mathsf{Sol}^\omega_\varsigma\rceil, \lceil\mathsf{Cont}^\omega_\varsigma\rceil)$ est définie comme $\lceil\mathcal{B}^\omega_\varsigma\rceil = \mathsf{pppf}\{\mathcal{B}^\omega_\varsigma\}\left(\mathcal{B}^\omega_\varsigma \mapsto \mathcal{B}^\omega_{\varsigma \Cap \mathsf{procs}(\mathcal{B}^\omega_\varsigma)}\right)$, avec $\mathcal{B}^\omega_\varsigma = (\overline{\mathsf{Req}^\omega_\varsigma}, \overline{\mathsf{Sol}^\omega_\varsigma}, \overline{\mathsf{Cont}^\omega_\varsigma})$:

$$\begin{aligned}
\overline{\mathsf{Req}^\omega_\varsigma} =& \{(a_i, a_j \Rsh^* a_i) \in \mathbf{Proc} \times \mathbf{Obj} \mid a_j \in \varsigma[a] \wedge (\exists (P, ps) \in \overline{\mathsf{Sol}^\omega_\varsigma}, a_i \in ps \\
& \vee \exists n \in \mathbb{I}^\omega, \mathsf{bond}(\omega) = a_i\} \\
\overline{\mathsf{Sol}^\omega_\varsigma} \subseteq& \{(P, ps) \in \mathbf{Obj} \times \wp(\mathbf{Proc}) \mid \exists (a_i, P) \in \overline{\mathsf{Req}^\omega_\varsigma} \wedge ps \in \mathbf{BS}^\wedge(P)\} \\
\overline{\mathsf{Cont}^\omega_\varsigma} =& \{(P, q \Rsh^* \mathsf{bond}(P)) \in \mathbf{Obj} \times \mathbf{Obj} \mid \exists (P, ps) \in \overline{\mathsf{Sol}^\omega_\varsigma} \\
& \wedge q \in \mathsf{maxCont}_\varsigma(\Sigma(P), P)\} \ .
\end{aligned}$$

Théorème 9.4 (Approximation inf.)**.** *Si le graphe $\lceil\mathcal{B}^\omega_\varsigma\rceil$ ne contient aucun cycle et si tous les objectifs référencés possèdent au moins une solution, alors $\ell_\varsigma(\omega) \neq \emptyset$.*

Démonstration. Nous dénotons par $max\varsigma = \varsigma \Cap \mathsf{procs}(\lceil\mathcal{B}^\omega_\varsigma\rceil)$ le contexte supporté par $\lceil\mathcal{B}^\omega_\varsigma\rceil$. Par induction sur le graphe acyclique $\lceil\mathcal{B}^\omega_\varsigma\rceil$, nous prouvons que $\forall s \in L, s \subseteq max\varsigma$, pour tout objectif P référencé dans $\lceil\mathcal{B}^\omega_\varsigma\rceil$ tel que $\mathsf{cible}(P) \in s$, $\exists \delta \in \ell_s(P)$ et $\mathsf{fin}(\delta) \subseteq max\varsigma$.

- $(P, \emptyset) \in \lceil\mathsf{Sol}^\omega_\varsigma\rceil \Rightarrow$ soit $\mathsf{cible}(P) = \mathsf{bond}(P)$ (donc $\delta = \varepsilon$), ou $\forall \zeta \in \mathbf{BS}(P), \zeta \in \mathbf{Sce} \wedge \Sigma(\zeta) = \{\Sigma(P)\}$, donc $\delta = \zeta$.
- nous supposons que tous les objectifs enfants de P sont concrétisables (aucun cycle). Si $\exists Q \in \lceil\mathsf{Cont}^\omega_\varsigma\rceil$, alors par hypothèse, $\ell_s(\mathsf{cible}(P) \Rsh^* \mathsf{cible}(Q) :: Q) \neq \emptyset$, donc $\ell_s(P) \neq \emptyset$. Sinon, par la définition 9.31, les concrétisations des enfants de P ne requièrent aucun processus de sorte $\Sigma(P)$. De plus, il existe $\zeta \in \mathbf{BS}(P)$ tel que $(P, \zeta^\wedge) \in \lceil\mathsf{Sol}^\omega_\varsigma\rceil$. Par hypothèse $\forall n \in \mathbb{I}^\zeta, \exists \delta^n \in \ell_{s^{n-1}}(\star \Rsh^* \mathsf{frappeur}(\zeta_n))$ avec soit $s^{n-1} = s$ si $n = 1$, ou $s^{n-1} = s \cdot \delta^1 \cdot \dots \cdot \delta^{n-1}$; et $\Sigma(P) \notin \Sigma(\delta^n)$ (par la définition 9.31). Ainsi, $\delta = \delta^1 :: \zeta^1 :: \dots \delta^n :: \zeta^n \in \ell_s(P)$ et $\mathsf{fin}(\delta) \subseteq max\varsigma$.

Enfin, comme $\ell_{max\varsigma}(\omega) \neq \emptyset$, $\ell_\varsigma(\omega) \neq \emptyset$ (propriété 9.6). □

Partant de la preuve du théorème ci-dessus, nous définissons $\mathsf{finProc}(\lceil\mathcal{B}^\omega_\varsigma\rceil, P)$ (définition 9.33) comme l'ensemble maximum des processus avec lesquels un scénario construit par le théorème 9.4 peut terminer (corollaire 9.2). Ceci permet une extension directe du théorème 9.4 pour prendre avantage de la séquentialité des objectifs (corollaire 9.3).

Définition 9.33 ($\mathsf{finProc} : \mathbb{B} \times \mathbf{Obj} \mapsto \wp(\mathbf{Proc})$)**.**

$$\begin{aligned}
\mathsf{finProc}(\lceil\mathcal{B}^\omega_\varsigma\rceil, P) = \{p \in \mathbf{Proc} \mid & \Sigma(p) = \Sigma(P) \wedge p = \mathsf{bond}(P) \\
& \vee \Sigma(p) \neq \Sigma(P) \wedge (\exists (P, Q) \in \lceil\mathsf{Cont}^\omega_\varsigma\rceil, p \in \mathsf{finProc}(\lceil\mathcal{B}^\omega_\varsigma\rceil, \mathsf{cible}(P) \Rsh^* \mathsf{cible}(Q)) \\
& \quad \vee p \in \mathsf{finProc}(\lceil\mathcal{B}^\omega_\varsigma\rceil, Q) \\
& \quad \vee \exists (P, ps) \in \lceil\mathsf{Sol}^\omega_\varsigma\rceil, \exists b_i \in ps, \exists b_j \in \varsigma[b], p \in \mathsf{finProc}(\lceil\mathcal{B}^\omega_\varsigma\rceil, b_j \Rsh^* b_i))\}
\end{aligned}$$

Corollaire 9.2. *Étant donné $P \in \mathbf{Obj}$, si le théorème 9.4 est satisfait avec $\ell_\varsigma(P) \neq \emptyset$, alors $\forall s \in L, s \subset \varsigma, \exists \delta \in \ell_\varsigma(P)$ tel que $\mathsf{fin}(\delta) \subseteq \mathsf{finProc}(\mathcal{B}^P_\varsigma, P)$.*

Corollaire 9.3. *Étant donnés $P \in \mathbf{Obj}$ et $\omega \in \mathbf{OS}$, si le théorème 9.4 est satisfait avec $\ell_\varsigma(P) \neq \emptyset$, et si le théorème 9.4 est satisfait avec $\ell_{\varsigma'}(\omega) \neq \emptyset$, où $\varsigma' = \varsigma \Cap \mathsf{finProc}(\mathcal{B}^P_\varsigma, P)$, alors le théorème 9.4 est satisfait avec $\ell_\varsigma(P \oplus \omega) \neq \emptyset$.*

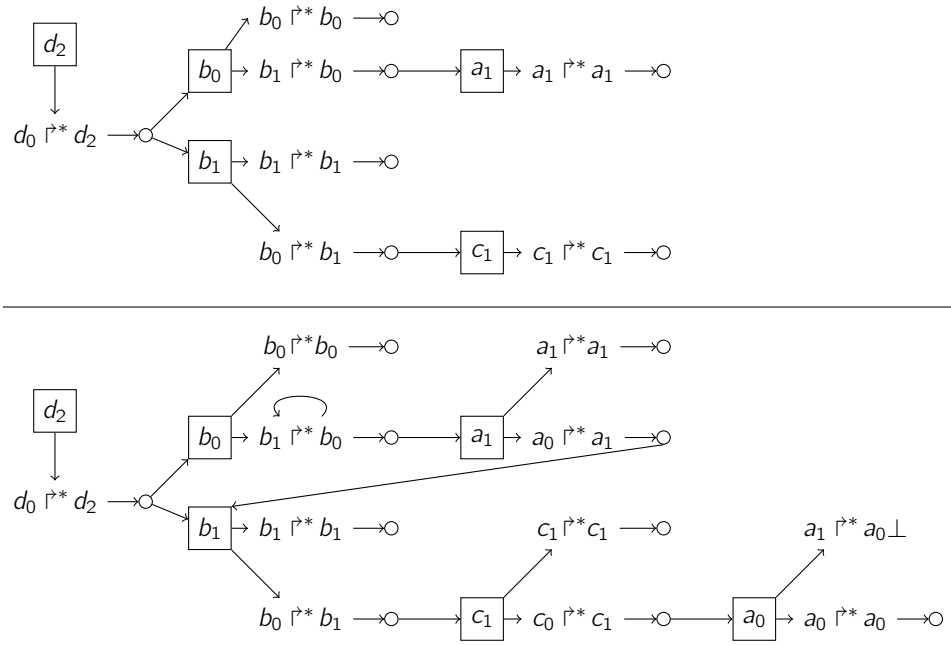


Figure 9.9 – Structure abstraite saturée $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ des Frappes de Processus de la figure 9.1 avec $\omega = d_0 \Rsh^* d_2$ et $\varsigma = \langle a_1, b_1, c_1, d_0 \rangle$ (haut), et $\varsigma = \langle a_0, b_1, c_0, d_0 \rangle$ (bas). D'après le théorème 9.4, ω est concretisable dans $\langle a_1, b_1, c_1, d_0 \rangle$. À ce stade, notre procédure reste non-concluante sur la concrétisabilité de ω dans $\langle a_0, b_1, c_0, d_0 \rangle$.

Implémentation.

Le calcul de la structure abstraite saturée $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ s'effectue par une addition progressive des relations entre les objectifs et les processus, découvertes par plusieurs explorations de la structure abstraite, résultant en une complexité maximale polynomiale en la taille de la structure abstraite. La vérification des conditions du théorème 9.4 s'effectue linéairement en la taille de la structure abstraite obtenue. Nous remarquons que la sélection arbitraire de solutions lors de la construction de $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ prévient des saturations superflues et peut ainsi augmenter la satisfaction de la vérification des conditions du théorème. Toutefois, ces sélections peuvent accroître la complexité de la vérification car plusieurs combinaisons de choix de solutions peuvent alors être testées.

Exemples.

La figure 9.9 montre deux exemples de l'application du théorème 9.4 sur l'exemple de Frappes de Processus de la figure 9.1.

La figure 9.9(haut) représente la structure abstraite $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ obtenue pour la décision de la concrétisation de $\omega = d_0 \Rsh^* d_2$ dans le contexte $\varsigma = \langle a_1, b_1, c_1, d_0 \rangle$. $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ ne contient aucun cycle, et tous les objectifs référencés ont une solution, le théorème 9.4 permet donc de conclure que ω est concrétisable dans ς. Intuitivement, la topologie de $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ nous indique qu'il existe un scénario permettant d'atteindre successivement b_0 et b_1 dans n'importe quel ordre et n'importe quel nombre de fois. Alors, comme b_0 et b_1 *suffisent* à résoudre l'objectif $d_0 \Rsh^* d_2$ (nous avons simplement oublié

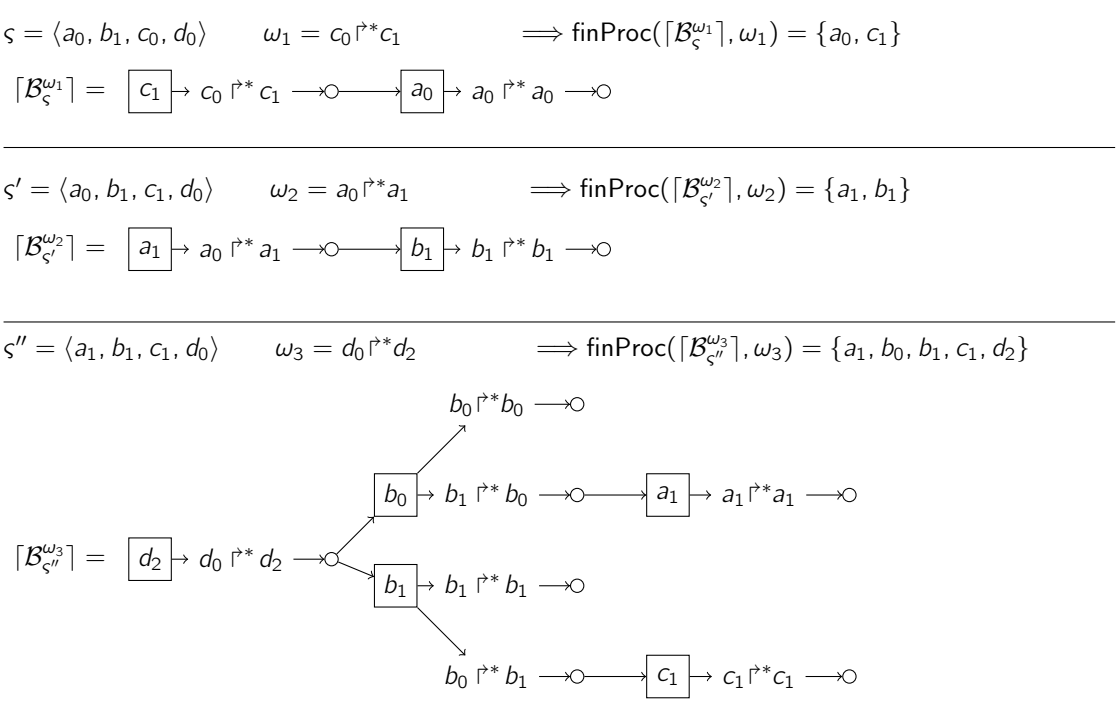


Figure 9.10 – Application du corollaire 9.3 pour la concrétisabilité de $\omega = c_0 \Rsh^* c_1 :: a_0 \Rsh^* a_1 :: d_0 \Rsh^* d_2$ dans le contexte $\varsigma = \langle a_0, b_1, c_0, d_0 \rangle$: corollaire 9.3 conclut que ω est bien concrétisable.

l'ordre et la répétition), nous pouvons conclure qu'il existe un scénario permettant d'atteindre d_2.

La figure 9.9(bas) représente la structure abstraite $\lceil \mathcal{B}_\varsigma^\omega \rceil$ obtenue pour la décision de la concrétisation de $\omega = d_0 \Rsh^* d_2$ dans le contexte $\varsigma = \langle a_0, b_1, c_0, d_0 \rangle$. Il apparaît que $\lceil \mathcal{B}_\varsigma^\omega \rceil$ ne satisfait pas les conditions du théorème 9.4, ne permettant donc pas de conclure sur la concrétisabilité de ω. En effet, l'objectif $a_1 \Rsh^* a_0$ est sans solution, indiquant qu'un scénario atteignant d_2 (si il existe) doit suivre un certain ordre dans la résolution des objectifs.

La figure 9.10 illustre alors l'application du corollaire 9.3 pour la concrétisabilité de $\omega = c_0 \Rsh^* c_1 :: a_0 \Rsh^* a_1 :: d_0 \Rsh^* d_2$ dans le même contexte $\varsigma = \langle a_0, b_1, c_0, d_0 \rangle$. Pour chaque élément de la séquence d'objectifs ω, nous vérifions par le théorème 9.4 sa concrétisabilité, et calculons le contexte de l'objectif suivant à l'aide la fonction finProc (définition 9.33) : ce contexte représente les états possibles après application d'un scénario induit par le théorème 9.4 (corollaire 9.2). Le corollaire 9.3 nous permet alors de conclure que $\omega = c_0 \Rsh^* c_1 :: a_0 \Rsh^* a_1 :: d_0 \Rsh^* d_2$ est concrétisable et donc que d_2 est atteignable depuis ς.

Nous pouvons remarquer que le choix de la séquence d'objectifs ω de ce dernier exemple peut être compris en analysant la structure abstraite donnée dans la figure 9.9(bas) : les objectifs ajoutés ($c_0 \Rsh^* c_1$ puis $a_0 \Rsh^* a_1$) engendrent indépendamment une structure abstraite possédant les propriétés requises pour le théorème 9.4.

Discussion.

Le corollaire 9.3 suggère que tester une séquence d'objectifs raffinée peut se révéler plus con-

cluant que le test sur la séquence d'objectifs originale donnée ; ceci a été démontré dans le dernier exemple. Un travail futur pourrait utiliser une analyse du graphe $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ afin de déterminer automatiquement quels raffinements de séquences pourraient se révéler comme de bons candidats à la satisfaction du théorème 9.4, et ainsi augmenter la conclusion de notre méthode.

9.5 Discussion

En s'aidant de la structure particulière des modèles en Frappes de Processus, une analyse statique efficace par interprétation abstraite a pu être développée afin de décider de la faisabilité d'une succession d'atteintes de processus (ou de niveaux de composants dans le cadre de la modélisation des RRB). Le calcul est effectué par approximations sup. et inf. de la décision, et peut se révéler non-concluant. Nous exploitons la complémentarité de deux abstractions des scénarios de Frappes de Processus (par séquences d'objectifs et de bonds). Plusieurs raffinements d'une séquence d'objectifs sont alors définis afin de détailler les étapes nécessaires à sa concrétisation. Ces raffinements sont exploités pour dériver des conditions nécessaires ou suffisantes pour la satisfaction de l'atteignabilité des processus. Des travaux futurs pourraient améliorer la conclusion de notre méthode, comme discuté en fin de sous-section 9.4.4. De même, le lien entre la conclusion de la méthode développée et la structure du graphe des interactions du RRB modélisé pourrait être étudié.

Les implémentations des approximations présentées se basent sur une structure abstraite dont la taille est quasiment polynomiale selon le nombre total de processus. D'une part, le calcul des raffinements peut être exponentiel selon le nombre de processus au sein d'une seule sorte ; d'une autre part, le calcul des décisions est polynomial en la taille de la structure abstraite. Ainsi, nos analyses se veulent efficaces quand le nombre de processus au sein d'une sorte est limité, alors qu'un très grand nombre de sortes devrait être supporté.

Cette approche nouvelle et originale est appliquée à l'analyse de grands RRB (notamment de 94 et 104 composants) dans le chapitre 11. Les temps de calcul sont très courts (autour du centième de seconde sur un ordinateur de bureau classique), ce qui montre une robustesse prometteuse pour des analyses à très grande échelle. L'utilisation de techniques standards de vérification formelle symbolique échoue très souvent pour de tels modèles, du fait de l'explosion combinatoire de l'espace des états. *À notre connaissance, notre méthode apporte la première application réussie de vérification formelle sur des RRB de telles tailles.*

Des travaux futurs pourraient rechercher de nouvelles applications des raffinements présentés sur les interprétations abstraites des Frappes de Processus. En particulier, ceux-ci révélant une causalité entre les changements de processus, ils exhibent des processus requis par l'atteignabilité d'un processus donné. Ce type d'analyse peut permettre d'amener à un *contrôle* du système étudié en agissant sur ces processus *clés* : par exemple en désactivant un gène clé afin de prévenir une cascade d'activation de gènes (ce qui est notamment recherché par les thérapies géniques).

Une autre direction de recherche est l'incorporation d'aspects quantitatifs au sein des décisions présentées sur l'atteignabilité des processus, telles que la probabilité d'atteindre un processus donné dans un intervalle de temps donné.

Quatrième partie

Mise en Œuvre et Applications Biologiques

Cette partie illustre l'application des résultats présentés dans les parties I, II et III à l'étude de systèmes biologiques concrets.
Le chapitre 10 introduit le logiciel PINT, développé dans le cadre de cette thèse, implémentant la spécification, la simulation et l'analyse des Frappes de Processus.
Le chapitre 11 détaille à la fois la modélisation quantitative de systèmes biologiques en Frappes de Processus et l'analyse statique de propriétés d'atteignabilité discrète au sein de grands Réseaux de Régulation Biologique.
En particulier, la mise en pratique des résultats du chapitre 9 montre une avancée significative pour la vérification formelle de systèmes complexes de grande taille.

Chapitre **10**

Implémentation : le Logiciel PINT

Nous présentons le logiciel PINT développé dans le cadre de cette thèse afin de faciliter la mise en pratique des résultats développés dans ce manuscrit. En particulier, PINT définit un langage de spécification des Frappes de Processus avec leur paramétrage temporisé et stochastique. Les méthodes d'analyses statiques développées dans la partie III sont également implémentées. PINT s'articule autour d'une bibliothèque de fonctions, programmée avec le langage OCaml, permettant son intégration dans d'autres outils.

10.1 Préliminaires

Afin d'apprécier et de faciliter la mise en œuvre des résultats apportées par cette thèse, nous avons mis au point le logiciel PINT (Paulevé, PINT, *Process hittINg related Tools*) implémentant la spécification discrète, stochastique et temporelle des Frappes de Processus, ainsi que les analyses statiques introduites dans la partie III. PINT apporte notamment :

- un langage de spécification pour les Frappes de Processus supportant le paramétrage stochastique et temporel présenté dans les chapitres 5 et 7 ;
- un simulateur non-markovien implémentant la machine abstraite développée dans les chapitres 6 et 7 ;
- un énumérateur des points fixes utilisant la méthode décrite dans le chapitre 8 ;
- un vérificateur statique de propriétés d'atteignabilité reposant sur les contributions du chapitre 9 ;
- des outils permettant l'importation et l'exportation des Frappes de Processus pour divers outils existants.

En sus de son orientation claire pour la modélisation des systèmes biologiques, nous prétendons que PINT peut être appliqué à l'étude d'autres systèmes complexes moins spécifiques. Les analyses des dynamiques fournies par PINT, reposant notamment sur les résultats du chapitre 9, évitent de construire le graphe complet des états, ce dernier pouvant devenir intraitable. Ces analyses aspirent à être très efficaces quand le nombre d'états pour chaque agent (c.-à-d. le nombre de processus par sorte) est petit, alors qu'un nombre potentiellement très grand d'agents est supporté.

Le site internet `http://process.hitting.free.fr` regroupe les documentations, des exemples de modèles, des références et des informations sur le développement de PINT. PINT est un logiciel libre et gratuit distribué sous la licence *open-source* CeCILL [*] ; il est programmé avec le

*. `http://www.cecill.info`

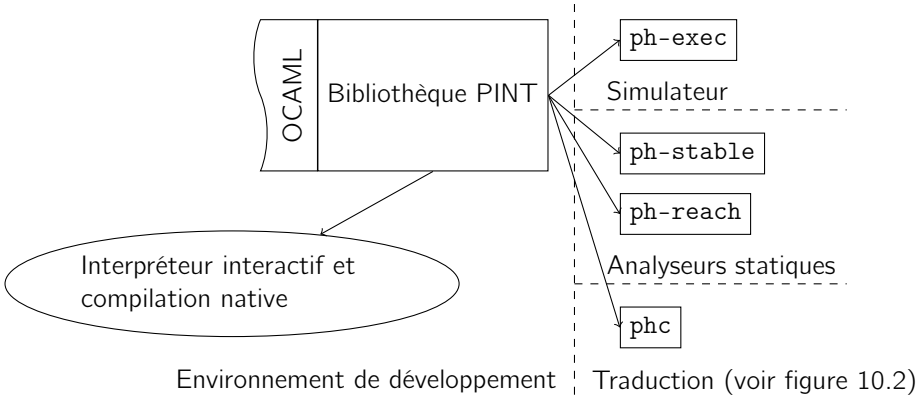


Figure 10.1 – Architecture de PINT : une bibliothèque OCaml et plusieurs outils en ligne de commande.

langage fonctionnel OCaml [†].

Nous présentons les fonctionnalités et l'architecture du logiciel PINT dans la section 10.2. Dans la section 10.3, nous présentons les éléments de base du langage PINT permettant la spécification de Frappes de Processus. Enfin, nous discutons des apports de ce logiciel dans la section 10.4.

10.2 Fonctionnalités et Architecture du Logiciel

L'architecture générale du logiciel PINT est décrite dans la figure 10.1 : elle consiste en une bibliothèque de fonctions programmée en OCaml, fournissant un interpréteur interactif et une compilation native des programmes, ainsi que d'une suite d'utilitaires en ligne de commandes prenant en entrée une représentation textuelle des Frappes de Processus :

- `ph-exec` est un simulateur non-markovien implémentant la machine abstraite générique (chapitre 6) instanciée aux Frappes de Processus munies de paramètres stochastiques et temporels (chapitre 7).
- `ph-stable` liste les points fixes des Frappes de Processus en utilisant la méthode décrite dans le chapitre 8.
- `ph-reach` vérifie formellement la satisfaction des Frappes de Processus pour des atteignabilités successives de processus (chapitre 9). L'énumération des processus clés à cette satisfaction est également possible.
- `phc` traduit les Frappes de Processus dans de nombreux formalismes (voir figure 10.2).

PINT implémente notamment les fonctionnalités suivantes :

Modélisation des RRB en Frappes de Processus. La spécification des Frappes de Processus s'effectue via une représentation textuelle des actions. Le langage PINT permet le raffinement progressif des dynamiques des RRB (chapitre 3) en spécifiant dans un premier temps le graphe des interactions à partir duquel la dynamique généralisée est calculée. Puis, des macros permettent de raffiner les dynamiques obtenues via la spécification manuelle d'actions supplémentaires, l'incorporation de coopérations entre processus ou encore la suppression d'actions automatiquement crées. La section 10.3 détaille les éléments clés de ce langage.

†. `http://caml.inria.fr`

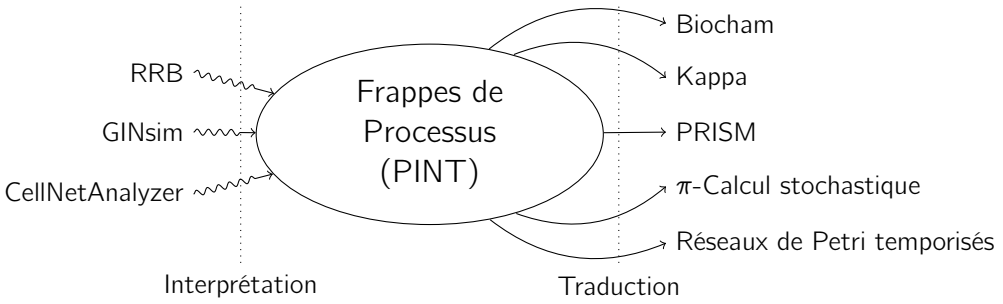


Figure 10.2 – Interopérabilité actuelle entre PINT et les autres outils existants.

Paramétrage stochastique et temporel. Les actions spécifiées en PINT peuvent se voir attribuer un délai qui est une variable aléatoire suivant une distribution d'Erlang avec un taux d'utilisation et un facteur d'absorption donné (chapitre 7). Il est également possible de spécifier l'intervalle de tir dans lequel l'action a lieu ; PINT calculera alors les paramètres stochastiques correspondants à l'aide des outils développés dans le chapitre 5.

Énumération des points fixes de la dynamique. L'ensemble des points fixes des Frappes de Processus s'effectue via une analyse structurelle du modèle : le calcul est réduit à la recherche des n-cliques dans un graphe n-parti, où n est le nombre de sortes (chapitre 8). La complexité est exponentielle selon le nombre de sortes.

Décision d'atteignabilité de processus. Étant données des Frappes de Processus et un état initial, PINT implémente l'analyse statique par interprétation abstraite des dynamiques afin de calculer des approximations supérieures et inférieures de l'atteignabilité d'une succession de processus (chapitre 9). Ce calcul possède une complexité théorique limitée (polynomiale selon le nombre de processus et exponentielle selon le nombre de processus au sein d'une seule sorte) et peut se révéler non-concluante.

Extraction des processus clés. Un sous-ensemble des processus clés (indépendamment nécessaires) à la satisfaction d'une propriété d'atteignabilité peut également être calculé pendant le processus de décision (et donc avec la même complexité).

Afin de faciliter la comparaison et la complémentarité avec les autres outils de l'état de l'art, la traduction de et vers PINT a été particulièrement développée. La figure 10.2 résume les conversions actuellement implémentées entre les Frappes de Processus et les différents outils existants. Ainsi, différents modèles formels (tels que les RRB) peuvent être interprétés en Frappes de Processus, résultant généralement en une abstraction du modèle de départ ; et les Frappes de Processus peuvent être traduites (exactement) dans un grand nombre de formalismes (voir notamment les chapitres 4 et 7), comme Biocham, Kappa, PRISM, le π-calcul stochastique et les réseaux de Petri temporisés.

10.3 Éléments du Langage PINT

La représentation textuelle des Frappes de Processus s'effectue via le langage PINT permettant la spécification progressive de l'ensemble des actions, via une succession d'instructions ajoutant ou supprimant des actions. Nous présentons dans cette section les instructions importantes de ce langage. Des exemples peuvent être trouvés dans le chapitre 11 et l'appendice B.

Définition d'une sorte. Une sorte est déclarée en donnant le processus ayant le niveau le plus élevé :

process `a X`

définit une sorte a contenant les processus $a_0, \ldots, a_X$.

Spécification d'une action. L'action $a_i \to b_j \not\uparrow b_k$ suivant un taux r et un facteur d'absorption de stochasticité sa est ajoutée par l'instruction suivante :

```
a i -> b j k @ r ~ sa
```

Il est également possible de spécifier un intervalle de tir $[d; D]$ pour un niveau de confiance c (par défaut à 99%).

```
a i -> b j k @ [d;D]#c
```

Dynamique généralisée d'un RRB. La macro **GRN** calcul la dynamique généralisée du graphe des interactions spécifié (section 3.4). L'activation (resp. inhibition) d'un composant b par un composant a avec un seuil X s'écrit `a X -> + b` (resp. `a X -> - b`).

Voici un exemple d'utilisation de la macro **GRN** appliqué au graphe des interactions possédant les activations $a \xrightarrow{1} b$ et $a \xrightarrow{2} a$, et l'inhibition $b \xrightarrow{1} a$:

GRN`([a 1 -> + b; b 1 -> - a; a 2 -> + a ])`

Raffinement par coopération. La macro **COOPERATIVITY**`([a1;...;an] -> b j k, [s1;...;sp])` crée une action coopérative entre les sortes $a^1, \ldots, a^n$ pour le bond $b_j \not\uparrow b_k$. Cette coopération est effective pour chaque état $s_1, \ldots, s_p$. Les actions provenant des sortes $a^1, \ldots, a^n$ vers b_j sont également supprimées.

L'instruction suivante crée une coopération entre les sortes a et b pour faire bondir le processus c_0 vers c_1 seulement si a_1 et b_0 ou a_0 et b_1 sont présents.

COOPERATIVITY`([a;b] -> c 0 1, [[1;0];[0;1]])`

Suppression d'une action Une action $a_i \to b_j \not\uparrow b_k$ peut être supprimée via la macro **RM** :

RM`({a i -> b j k})`

10.4 Discussion

Ce chapitre présente le logiciel PINT développé dans le cadre de cette thèse pour implémenter les analyses se basant sur les Frappes de Processus, introduit également dans cette thèse.

Le site internet `http://process.hitting.free.fr` regroupe la documentation complète du langage PINT, les manuels des utilitaires ainsi que les fonctions implémentées dans la bibliothèque OCaml. Un répertoire de modèles PINT est également disponible.

Autres logiciels existants. Il existe un grand nombre de logiciels dédiés à la modélisation et l'analyse de systèmes biologiques. Nous citons ici quelques un de ces outils offrant des fonctionnalités complémentaires à PINT : CellNetAnalyser (Klamt, Saez-Rodriguez & Gilles, 2007) propose un ensemble d'analyses structurelles et fonctionnelles des RRB. SMBioNet (I3S, SMBioNet) permet l'inférence des paramètres discrets de René Thomas d'un Réseau Discret à partir d'une formule CTL : étant donné un Réseau Discret dont les paramètres discrets sont partiellements connus, SMBioNet calcule tous les paramètres restants permettant au modèle de satisfaire la propriété CTL donnée. GINsim (Gonzalez et coll., 2006) est dédié à la modélisation et la simulation discrète des RRB, dont les interactions peuvent recevoir une classe de priorité. Il est également possible d'exporter des modèles GINsim en Réseau de Petri pour effectuer des analyses dynamiques avec des outils standards. Kappa (Danos et coll., 2008) est un langage basé sur des règles de réécriture dédié à la modélisation d'un grand nombre de phénomènes biologiques. Plusieurs méthodes d'abstractions ont été développées afin d'analyser ces règles, notamment une approximation supérieure efficace de l'ensemble des complexes atteignables. Biocham (Fages & Soliman, 2008*b*) apporte un langage simple de description des réactions biologiques. Il supporte de nombreuses analyses qualitatives et quantitatives, dont la vérification symbolique de propriétés CTL via le logiciel NuSMV (Cimatti, Clarke, Giunchiglia, Giunchiglia, Pistore, Roveri, Sebastiani & Tacchella, 2002).

Le Logiciel PINT s'établit comme une base pour de futurs développements autour des Frappes de Processus et propose ainsi d'assurer une continuité avec les nouvelles contributions à venir. Des travaux futurs pourraient doter PINT d'une interface graphique, notamment pour la spécification des Frappes de Processus.

L'application de PINT à l'étude de systèmes biologiques est détaillée dans le chapitre 11.

Chapitre **11**

Application des Frappes de Processus aux Réseaux de Régulation Biologique

Nous appliquons dans un premier temps les Frappes de Processus à la modélisation quantitative du processus de segmentation chez le métazoaire et du cycle circadien chez les mammifères : alors que le premier modèle illustre l'intérêt de la modélisation par raffinement et du facteur d'absorption de stochasticité afin de contrôler la variance temporelle des actions, le second modèle prend avantage de la spécification par intervalle de temps de la durée aléatoire des actions.
Enfin, nous appliquons les Frappes de Processus à la vérification formelle de propriétés dynamiques discrètes sur des grands Réseaux de Régulation Biologique (regroupant jusqu'à plus de cent composants). Les méthodes développées durant cette thèse rendent traitable l'analyse formelle de tels systèmes complexes à grande échelle. Ceci illustre la contribution importante du chapitre 9, à savoir l'analyse statique par interprétation abstraites des dynamiques des Frappes de Processus.

11.1 Préliminaires

Dans ce chapitre, nous donnons des exemples d'application des résultats apportés par cette thèse à la modélisation de Réseaux de Régulation Biologiques (RRB) en Frappes de Processus.

Dans la section 11.2, nous illustrons les bénéfices apportés par l'introduction conjointe du temps et de l'aléatoire dans les modèles (partie II) afin de modéliser quantitativement le comportement de systèmes biologiques :

- La modélisation en Frappes de Processus de *la segmentation chez le métazoaire* proposée en sous-section 11.2.1 prend avantage du facteur d'absorption de stochasticité afin de reproduire à la fois un comportement temporel et stochastique. Nous utiliserons également la traduction des Frappes de Processus en PRISM afin de vérifier formellement des propriétés quantitatives sur le modèle obtenu.
- La modélisation en Frappes de Processus du *cycle circadien chez les mammifères* proposée en sous-section 11.2.2 utilise abondamment la spécification des actions via les intervalles de temps afin de reproduire un comportement temporel précis (tout en gardant un cadre général stochastique).

Les RRB étudiés dans cette section 11.2 mettent en relation un nombre limité de composants. Bien que la méthode générique de simulation non-markovienne développée dans le chapitre 6 autorise un

grand nombre de composants, la vérification formelle en PRISM souffre rapidement de l'explosion combinatoire des transitions engendrées par nos modèles avec de grands facteurs d'absorption de stochasticité. Notamment la vérification formelle de propriétés via PRISM de la modélisation du cycle circadien (sous-section 11.2.2), regroupant cinq composants, n'est plus traitable lorsque l'on demande une précision temporelle appréciable.

Dans le cadre de la vérification de propriétés discrètes, nous illustrons dans la section 11.3 l'importante contribution de nos méthodes par analyse statique présentées dans la partie III de cette thèse, et plus particulièrement le chapitre 9 pour la vérification formelle de propriétés d'atteignabilité. Les systèmes étudiés regroupant jusqu'à plus de cent composants, nous pouvons parler de *grands* RRB.

- La modélisation en Frappes de Processus du *récepteur du facteur de croissance épidermique* proposée en sous-section 11.3.1 se base sur un RRB regroupant 20 composants. Pour ce modèle, nous détaillons l'obtention des états stables, la vérification de propriétés d'atteignabilité, ainsi que l'énumération des processus clés pour la satisfaction de ces propriétés.
- Les modélisations en Frappes de Processus du *récepteur des cellules T* dans une version à *40* et *94* composants, ainsi qu'un version étendue à *104* composants du récepteur du facteur de croissance épidermique sont proposées en sous-section 11.3.2, sur lesquelles nous vérifions des propriétés d'atteignabilité.

Alors que la vérification formelle de ces modélisations échoue avec les outils classiques à cause de l'importante explosion combinatoire des transitions possibles, nos méthodes répondent instantanément aux propriétés d'atteignabilité discrètes au sein de ces modèles. Ceci illustre une contribution majeure de cette thèse, à savoir l'analyse statique par interprétation abstraite des dynamiques présentée dans le chapitre 9 dont la complexité théorique est très faible (polynomiale selon le nombre de processus et exponentielle selon le nombre de processus au sein d'une seule sorte) comparés aux méthodes de vérification classiques (généralement exponentielles selon le nombre de processus).

L'application à la segmentation chez les métazoaires est publiée dans (Paulevé, Magnin & Roux, 2011*b*) et l'application au récepteur des cellules T dans (Paulevé, Magnin & Roux, 2011*a*) pour sa version à 40 composants.

11.2 Modélisations Quantitatives

11.2.1 La Segmentation chez les Métazoaires

Dans cette sous-section, nous illustrons les bénéfices de notre méthode pour la modélisation des phénomènes de segmentation chez les métazoaires (animaux pluricellulaires), notre but étant le contrôle de l'état final du système tout en reproduisant quantitativement son évolution. Nous nous basons sur le RRB établi *in silico* par François et coll. (2007) et étudié dans le cadre d'équations différentielles. Ce RRB tend à généraliser un motif commun aux processus de segmentation biologiques, comme par exemple chez la drosophile, où un nombre fixe de rayures régulières sont formées sur son corps. Notre modélisation fait écho à celle présentée dans le chapitre 5 (section 5.6 page 81) dans le cadre du π-calcul stochastique, la figure 5.9 page 82 illustre notamment l'évolution de ce système tel modélisé par François et coll..

Le RRB, dont le graphe des interactions est reproduit dans la figure 11.1(gauche), met en relation trois gènes. Le gène f (jouant le rôle d'un front d'onde) active le gène a dont les produits sont responsables de l'apparition de rayures. Le gène f active également un gène c (jouant le rôle

d'une horloge) dont les produits inhibent le gène a. L'auto-inhibition de c généralise la présence d'une chaîne d'inhibiteurs de a. La dynamique qualitative recherchée est la suivante : tant que f est exprimé, le système fonctionne : c change périodiquement de niveau d'expression, ce qui a pour effet d'activer puis inhiber périodiquement a, formant ainsi des rayures. Lorsque f ne s'exprime plus, c devient inactif, et a peut alors soit terminer actif ou inactif.

En outre de reproduire ce comportement qualitatif, nous cherchons à contrôler la fin du gène a (arbitrairement, nous voulons maximiser les chances pour que a termine dans l'état actif) ; et également à reproduire le comportement quantitatif de ce système, à savoir l'activation et inhibition régulière de a pour former un nombre fixe de rayures (fixé arbitrairement à 7). Dans un premier temps, nous détaillons notre modélisation en Frappes de Processus du comportement qualitatif ; puis nous discutons et vérifions les paramètres quantitatifs à attacher aux actions afin de contrôler la fin de a et la régularité des rayures produites.

Nous construisons dans un premier temps la dynamique généralisée du RRB en Frappes de Processus (section 3.4 page 28), en attachant à chaque composant deux niveaux qualitatifs (absent ou présent). Cette dynamique généralisée est représentée dans la figure 11.1(droite). Il apparaît alors que la spécification de la dynamique recherchée implique une frappe coopérative dans les Frappes de Processus : a doit atteindre a_1 uniquement si à la fois f l'active (f_1 est présent) et c ne l'inhibe pas (c_0 est présent). Ainsi, nous créons une sorte coopérative fc reflétant l'état du couple de f et c et nous remplaçons les frappes depuis c_0 et f_1 vers a_0 par une frappe depuis fc_{10} (représentant alors la présence de a_1 et c_0). Enfin, le processus c_0 n'atteignant c_1 que si f_1 est présent, nous supprimons l'auto-frappe de c_0 ; f_0 étant l'état final de f, nous supprimons également l'auto-frappe de f_0. Les Frappes de Processus résultantes sont représentées dans la figure 11.2(a).

Au regard du Graphe Sans-Frappe de nos Frappes de Processus (figure 11.2(b)), seul un état stable est présent : $\langle f_0, c_0, fc_{00}, a_0\rangle$. Afin de rendre stable l'état $\langle f_0, c_0, fc_{00}, a_1\rangle$, nous supprimons la répression de a_1 par f_0 en supprimant l'action $f_0 \to a_1 \not\Rightarrow a_0$ des Frappes de Processus. Ce procédé est illustré dans la figure 11.2(b). À titre d'information, nous avons inféré les paramètres discrets de René Thomas pour les gènes a et c depuis les Frappes de Processus obtenues (sous-section 3.6.1 page 33) :

$$\begin{array}{lll} K_{a,\emptyset,\{a,c,f\}} = 0 & K_{a,\{a,c\},\{f\}} = 1 & K_{c,\emptyset,\{c,f\}} = 0 \\ K_{a,\{a\},\{c,f\}} = 0 & K_{a,\{a,f\},\{c\}} = 0 & K_{c,\{c\},\{f\}} = 0 \\ K_{a,\{c\},\{a,f\}} = 0 & K_{a,\{c,f\},\{a\}} = 1 & K_{c,\{f\},\{c\}} = 0 \\ K_{a,\{f\},\{a,c\}} = 0 & K_{a,\{a,c,f\},\emptyset} = 1 & K_{c,\{c,f\},\emptyset} = 1 \ . \end{array}$$

La dynamique qualitative étant définie, nous nous intéressons au contrôle, via un paramétrage stochastique et temporel, du processus final de sorte a — soit a_0 soit a_1 — lorsque f_1 devient f_0. Au regard des Frappes de Processus de la figure 11.2(a), et en considérant que le processus f_0 est présent, nous pouvons déduire qu'au plus c_1 est présent, au plus a_1 peut se faire frapper par c_1 pour bondir vers a_0 ; de manière similaire, au plus fc_{10} est présent, au plus a_0 peut se faire frapper pour devenir a_1. Afin d'augmenter la probabilité que le processus final soit a_1, et en se concentrant uniquement sur le paramétrage des actions dont f_0 est le frappeur, nous pouvons réduire la présence de c_1 en augmentant le taux de l'action $f_0 \to c_1 \not\Rightarrow c_0$ et étendre la présence de fc_{10} en réduisant le taux de l'action $f_0 \to fc_{10} \not\Rightarrow fc_{00}$.

Finalement, l'obtention de rayures régulières est acquise en fixant un fort facteur d'absorption de stochasticité pour les actions responsables des frappes des processus a et c lorsque f_1 est présent. La figure 11.3 trace l'évolution des niveaux des gènes a, c et f au cours d'une simulation des Frappes de Processus obtenues depuis l'état initial $\langle f_1, c_0, fc_{10}, a_0\rangle$. Le taux de l'action $f_0 \to c_1 \not\Rightarrow c_0$

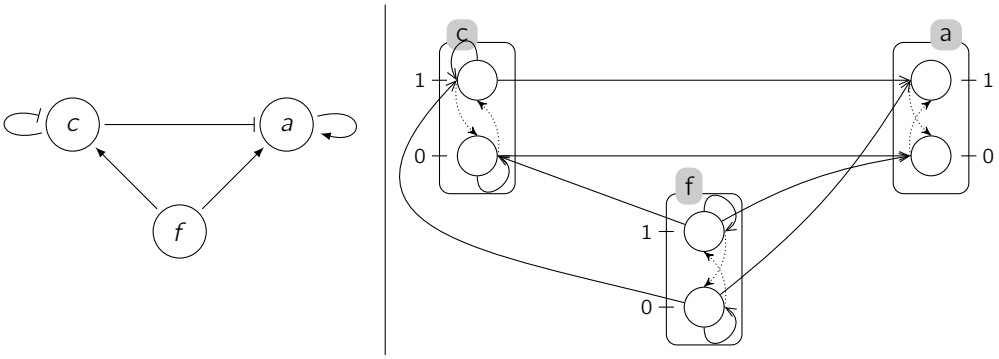


Figure 11.1 – (gauche) Graphe des interactions du RRB modélisation le processus de segmentation chez les métazoaires. Tous les composants ont deux niveaux qualitatifs et les seuils des régulations sont supposées à 1. (droite) Frappes de Processus de la dynamique généralisée du RRB de gauche.

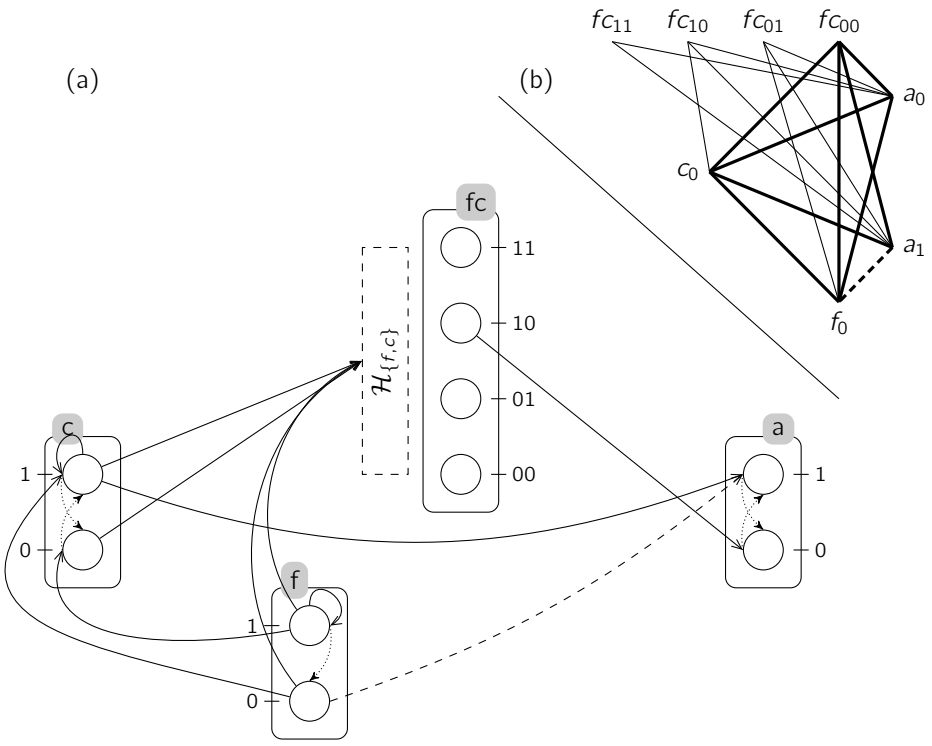


Figure 11.2 – (a) Frappes de Processus modélisant la segmentation chez les métazoaires où $\mathcal{H}_{\{f,c\}}$ représente les actions sur la sorte coopérative fc ; (b) Graphe Sans-Frappe correspondant. L'absence de frappe de f_0 vers a_1 (lignes pointillées) contrôle la présence de la relation entre f_0 et a_1 dans le Graphe Sans-Frappe (b). Si une telle relation existe, deux 4-cliques sont présentes (c.-à-d. deux états stables) : $\langle c_0, f_0, fc_{00}, a_0 \rangle$ et $\langle c_0, f_0, fc_{00}, a_1 \rangle$ (lignes épaisses).

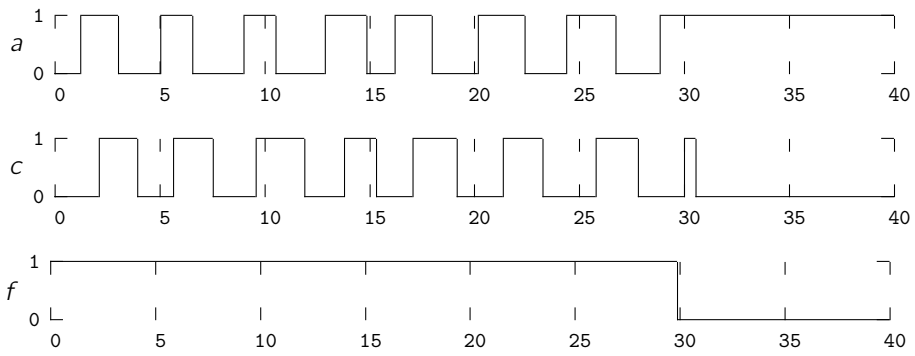


Figure 11.3 – Simulation des Frappes de Processus modélisant la segmentation : évolution de l'expression du gène a (haut), c (milieu) et f (bas).

est grand devant le taux de l'action $c_1 \to a_1 \not\uparrow a_0$ et les couples des taux et facteurs d'absorption des frappes sur a, c et f ont été fixés de sorte que c change de niveau toutes les 2 unités de temps, les frappes de c sur a prennent 1 unité de temps et f se désactive au bout de 30 unités de temps, avec une très faible variance. Nous observons sur cette simulation que f_0 frappe c_1 avant que c_1 n'ait le temps de frapper a_1 : l'état final est alors $\langle f_0, c_0, fc_{00}, a_1 \rangle$.

Nous remarquons que le modèle obtenu en Frappes de Processus est très proche de l'application en π-calcul stochastique étudiée en section 5.6 page 81), une étude sur la vérification formelle des propriétés quantitatives via le logiciel PRISM donnerait des résultats similaires. Le modèle complet des Frappes de Processus se trouve dans l'appendice B.1.

11.2.2 Le Cycle Circadien chez les Mammifères

Nous illustrons ici l'utilité de la spécification par intervalle de temps pour le tir des actions en Frappes de Processus via une modélisation du cycle circadien chez les mammifères inspirée par Leloup & Goldbeter (2003). Ce système, dans sa version pour la drosophile, a été étudié par Fromentin, Eveillard & Roux (2010) pour la découverte de contraintes temporelles strictes gouvernant la présence de cycles. Dans notre modélisation, nous nous intéressons seulement à la simulation stochastique d'un tel système.

Partant de la modélisation de Leloup & Goldbeter (2003), nous extrayons un modèle simplifié du cycle circadien, dont le graphe des interactions, regroupant 5 composants, est représenté dans la figure 11.4. Deux protéines PER et CRY, produites respectivement par la transcription de per_mRNA et cry_mRNA, peuvent s'assembler en un complexe $PERCRY$ qui, à son tour s'associe avec d'autres composants (non détaillés ici), produisant finalement le complexe noté Cn, inhibant alors la transcription de per_mRNA et cry_mRNA. Ainsi, la production de PER et CRY devient cyclique, et la période de ce cycle est généralement de 24 heures.

Notre modélisation en Frappes de Processus de ce système raffine la dynamique généralisée du graphe des interactions de la figure 11.4 en construisant une coopération entre PER et CRY activant Cn. La sorte coopérative obtenue, $PERCRY$, correspond ainsi au complexe $PERCRY$ cité précédemment. Enfin, en accord avec les observations de Leloup & Goldbeter (2003), nous affectons aux différentes actions différents intervalles de temps afin de reproduire le cycle de production sur 24

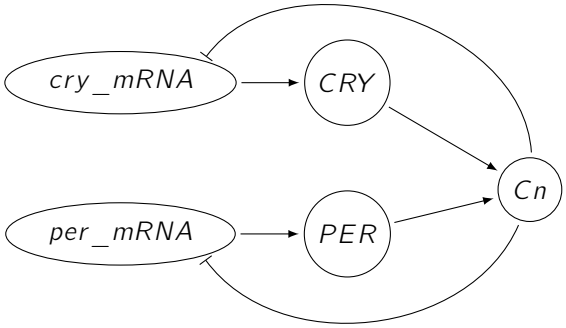


Figure 11.4 – Graphe des interactions illustrant les régulations entre les composants impliqués dans le cycle circadien des mammifères.

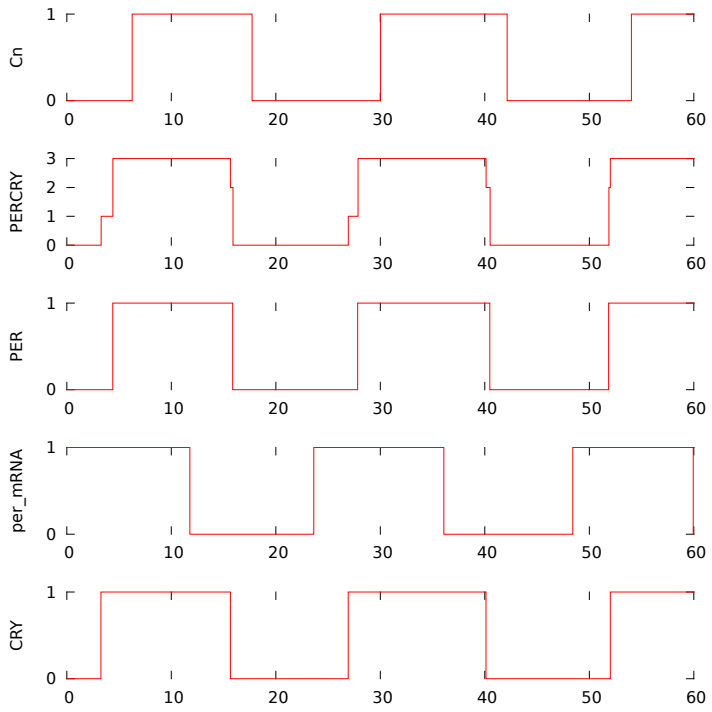
heures. Ces intervalles de temps sont alors traduits automatiquement en paramètres stochastiques à l'aide des outils développés dans le chapitre 5. La simulation est obtenue à l'aide de la machine abstraite développée dans le chapitre 6. La figure 11.5(haut) montre une l'évolution de différents composants de ce système au cours d'une simulation, où $PERCRY$ est la sorte coopérative entre PER et CRY, $PERCRY_0$ correspond à PER_0 et CRY_0 ; $PERCRY_1$ à PER_0 et CRY_1 ; $PERCRY_2$ à PER_1 et CRY_0 ; $PERCRY_3$ à PER_1 et CRY_1.

Leloup & Goldbeter étudient également l'impact de la phosphorylation de PER sur les cycles obtenus afin de comprendre certaines pathologies, telles que le déphasage du cycle circadien, ou le changement de sa période. En résumé, la phosphorylation de PER l'empêche de se complexifier avec CRY. Ainsi, si le taux de phosphorylation est trop élevé, le procédé global de formation du complexe se trouve ralenti entraînant alors une augmentation de la période. Nous illustrons ceci en augmentant l'intervalle de temps des actions de PER sur la sorte coopérative $PERCRY$, ralentissant alors sa progression vers le niveau actif. La figure 11.5(bas) trace l'évolution d'une simulation avec un tel paramétrage : le ralentissement apparaît dans le passage de $PERCRY_1$ à $PERCRY_3$ retardant alors l'activation de Cn. Le modèle complet en Frappes de Processus est reproduit dans le listing 11.1.

11.3 Analyse Statique de Propriétés Qualitatives

Nous commençons cette section pour l'étude d'un RRB de taille modérée (20 composants) modélisant une partie du récepteur du facteur de croissance épidermique. Nous nous comparons notamment à de précédents travaux étudiant les états stables, des propriétés d'atteignabilité ainsi que les composants clés pour cette atteignabilité, utilisant des méthodes plus « manuelles » et reposant sur la présence d'un état stable ; hypothèse non requise par nos méthodes.

Enfin, la modélisation de RRB de plus grande échelle est abordée dans la sous-section 11.3.2, où nous illustrons la grande efficacité de nos méthodes de vérification de propriétés d'atteignabilité par analyse statique, montrant une approche très prometteuse pour l'analyse de réseaux d'encore plus grande échelle.



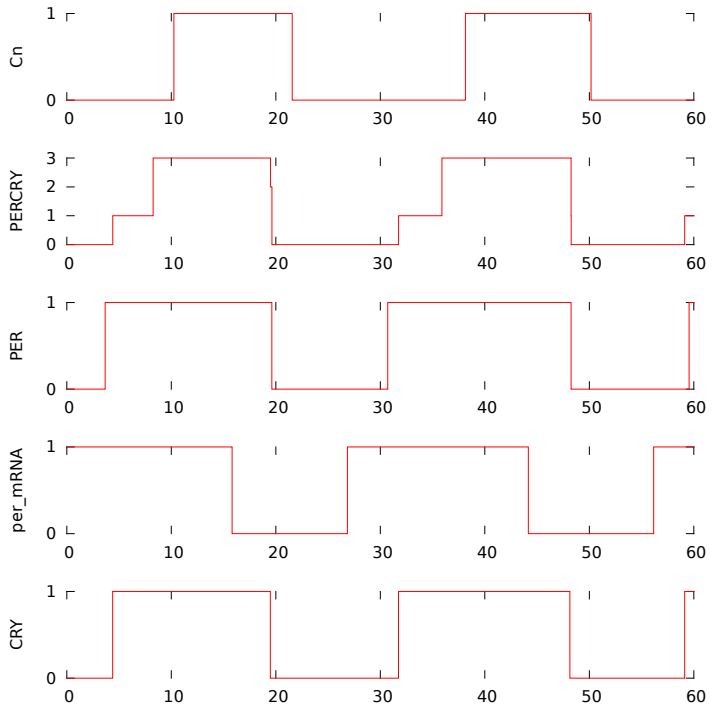


Figure 11.5 – Simulations des Frappes de Processus résultant de la modélisation du cycle circadien (figure 11.4). En bas, la formation du complexe entre CRY et PER est ralentie par PER afin de simuler une phosphorylation accrue de ce dernier, et entraînant alors un décalage du cycle et une augmentation de sa période.

```
directive default_rate 60.

process PER 1
process CRY 1
process Cn 1
process cry_mRNA 1
process per_mRNA 1

GRN([
    PER 1 -> + Cn @ [1.8;2.2];
    CRY 1 -> + Cn @ [1.8;2.2];
    cry_mRNA 1 -> + CRY @ [3.5;4.5]; (* 4h *)
    per_mRNA 1 -> + PER @ [3.5;4.5]; (* 4h *)
    Cn 1 -> - cry_mRNA @ [5.5;6.5];
    Cn 1 -> - per_mRNA @ [5.5;6.5]
])
COOPERATIVITY([PER;CRY] -> Cn 0 1 @ [1.8;2.2], [[1;1]]) (* 2h *)

(* ralentissement de la formation du complexe quand PER se phosphorylise
   rapidement *)
{ PER 1 -> PERCRY * * } @ [3.;5.]

initial_state cry_mRNA 1, per_mRNA 1
```

Listing 11.1 – Code source PINT de la modélisation en Frappes de Processus du cycle circadien avec forte phosphorylation de PER.

11.3.1 Le récepteur du facteur de croissance épidermique (20 composants)

Nous partons du modèle du récepteur de facteur de croissance épidermique publié par Sahin, Frohlich, Lobke, Korf, Burmester, Majety, Mattern, Schupp, Chaouiya, Thieffry, Poustka, Wiemann, Beissbarth & Arlt (2009) et étudié dans le cadre de recherches autour du cancer du sein. Le graphe des interactions regroupant 20 composants est reproduit dans la figure 11.6. Ce réseau décrit une cascade d'activations en amont du composant *pRB* contrôlant la transition G1/S impliquée dans le processus de division cellulaire. Le composant EGF agit alors comme l'entrée de cette cascade : quand il est exprimé *pRB* sera potentiellement activé. En se basant sur la littérature, Sahin et coll. ont établi un ensemble de règles logiques régissant l'activation des composants de ce réseaux.

En partant du graphe des interactions, nous modélisons en Frappes de Processus sa dynamique généralisée que nous raffinons par l'introduction de coopérations codant les différents règles logiques entre les composants. Les Frappes de Processus obtenues regroupent 670 actions et engendrent un total de 2^{64} états. Nous notons que ce nombre d'états inclut l'ajout des sortes coopératives supplémentaires. Le modèle complet en Frappes de Processus est donnée dans l'appendice B.4.

La recherche des états stables s'effectue à l'aide des algorithmes présentés dans le chapitre 8 et sont obtenus en moins d'une seconde sur un ordinateur de bureau classique. Les états stables obtenus sont les suivants, où seuls les composants dont le niveau est 1 sont écrits :

- AKT1, CDK2, CDK4, CDK6, CycD1, CycE1, EGF, ERBB1, ERBB1_2, ERBB1_3, ERBB2, ERBB2_3, ERBB3, ERalpha, MEK1, MYC, pRB.
- AKT1, CDK2, CDK4, CDK6, CycD1, CycE1, ERalpha, IGF1R, MEK1, MYC, pRB.
- AKT1, CDK2, CycE1, EGF, ERBB1, ERBB1_2, ERBB1_3, ERBB2, ERBB2_3, ERBB3, ERalpha, MEK1, MYC.

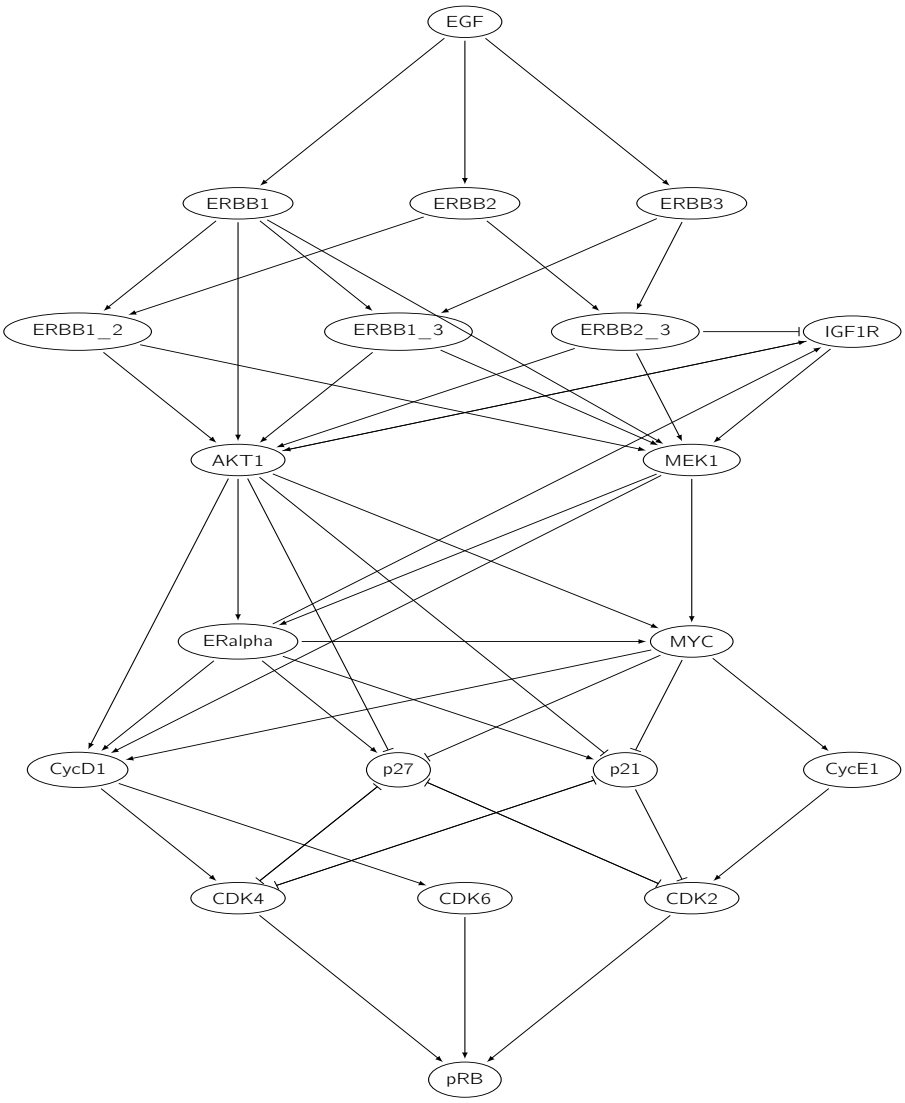


Figure 11.6 – Graphe des interactions modélisant le récepteur du facteur de croissance épidermique reproduit depuis (Sahin et coll., 2009).

– AKT1, CDK2, CycE1, ERalpha, IGF1R, MEK1, MYC.
– $\emptyset$ (tous les composants sont au niveau 0).

Comme résumé dans le tableau 11.1 en fin de section (ligne egfr20), la vérification formelle de propriétés d'atteignabilité s'effectue en quelques millisecondes. Nous nous intéressons particulièrement à l'atteignabilité du processus pRB_1 pour différents états initiaux. Sahin et coll. (2009) étudient notamment les composants clés pour l'activation de pRB en partant de l'état initial où tous les composants sont à 0, hormis EGF : pour chaque composant, ils simulent sa désactivation en prévenant l'atteinte de son niveau 1 et vérifient si le niveau 1 de pRB est toujours atteignable.

En utilisant la méthode développée dans le chapitre 9, nous pouvons calculer directement l'ensemble des processus indépendamment requis pour l'atteinte du processus pRB_1. Nous obtenons ainsi, toujours en quelques millisecondes, le même ensemble de composants clés ($CDK4$, $CDK6$, $CycD1$, $ERalpha_1$, MYC) avec en supplément le processus coopératif $CycD1p21p27_4$ (représentant la présence de $CycD1$ et l'absence de $p21$ et $p27$).

11.3.2 Modélisations de Grands RRB (40, 94 et 104 composants)

Récepteur des cellules T (40 et 94 composants). Le récepteur des cellules T est responsable de la reconnaissance de certaines molécules qui, si détectées, vont déclencher via une cascade de signalisations une réponse de la cellule T à la molécule étrangère (pouvant impliquer la mort de la cellule T). Nous nous basons sur deux modèles de RRB de ce récepteur regroupant 40 composants (Klamt et coll., 2006) et 94 composants (Saez-Rodriguez, Simeoni, Lindquist, Hemenway, Bommhardt, Arndt, Haus, Weismantel, Gilles, Klamt & Schraven, 2007), dont leur graphe des interactions est représenté respectivement dans les figures 11.7 et 11.8. Selon la molécule détectée, les composants d'entrées $CD45$, $CD8$ et $TCRlig$ sont activés ou non ; et nous nous intéressons alors à l'activation potentielle des composants de sortie CRE, $AP1$, $NFkB$ et $NFAT$ (version à 40 composants) plus SRE, $Cyc1$, $p21c$, $p27k$ $FKHR$ et $BclXL$ (version à 94 composants).

Les Frappes de Processus obtenues (appendices B.2 page 205 et B.3 page 207) regroupent, pour la version à 40 composants, 301 actions entre 156 processus séparés en 54 sortes (la sorte la plus grande contient 8 processus), générant un nombre total de 2^{73} ($\approx 10^{22}$) états ; et pour la version à 94 composants, 1124 actions entre 448 processus séparés en 133 sortes (la sorte la plus grande contient 16 processus), générant un nombre total de 2^{194} ($\approx 2 \cdot 10^{58}$) états. Ces modèles ont été obtenus en partant d'une traduction automatique du modèle fourni au format GINsim et CellNetAnalyser, à laquelle la spécification des coopérations a été manuellement optimisée afin de réduire le nombre de processus par sortes (en utilisant la factorisation des sortes coopératives expliquée dans la section 3.5 page 30).

Nous avons alors vérifié formellement les atteintes des différents composants de sortie en fonction des différentes configurations des entrées possibles. Notre méthode par analyse statique est toujours concluante, et répond en quelques centièmes de secondes (tableau 11.1, lignes tcrsig40 et tcrsig94).

Récepteur du facteur de croissance épidermique (104 composants). Nous utilisons ici une version étendue à 104 composants du modèle étudié dans la sous-section 11.3.1 par Samaga, Saez-Rodriguez, Alexopoulos, Sorger & Klamt (2009). La figure 11.9 représente son graphe des interactions.

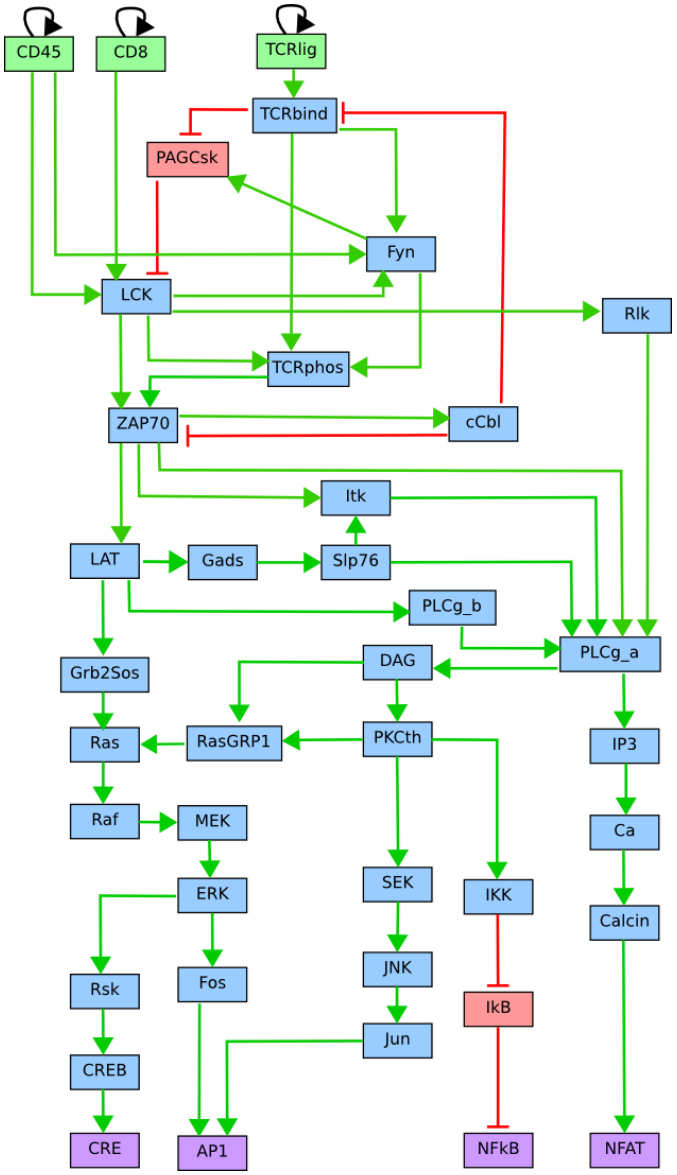


Figure 11.7 – Graphe des interactions du récepteur des cellules T (40 composants) reproduit depuis (Klamt et coll., 2006).

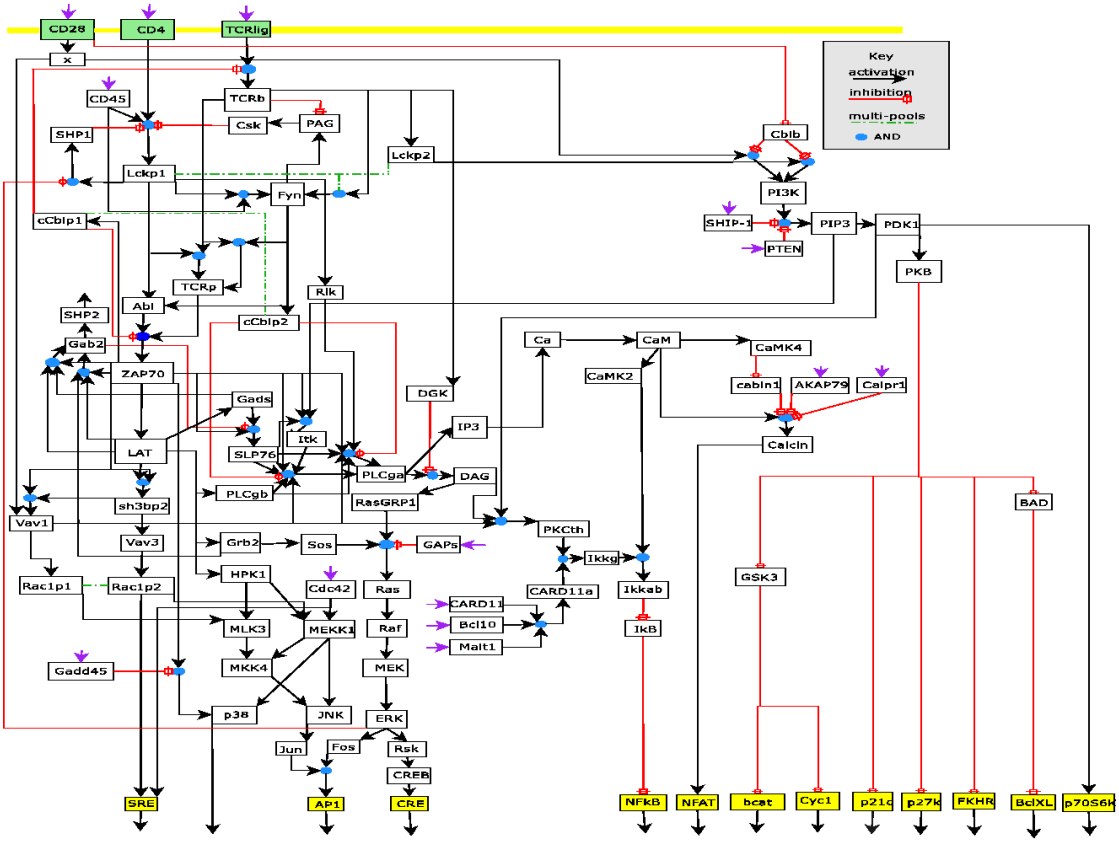


Figure 11.8 – Graphe des interactions du récepteur des cellules T (94 composants) reproduit depuis (Saez-Rodriguez et coll., 2007).

Les Frappes de Processus obtenues (appendice B.5) regroupent 2356 actions entre 748 processus séparés en 193 sortes (la sorte la plus grande contient 16 processus), générant un nombre total de 2^{320} ($\approx 2 \cdot 10^{96}$) états. Ce modèle résulte de la traduction automatique en Frappes de Processus du modèle original au format CellNetAnalyser, à laquelle la spécification des coopérations a été manuellement optimisée afin de réduire le nombre de processus par sortes.

Une fois encore, notre méthode de vérification formelle des atteignabilités par analyse statique est toujours concluante et s'exécute en quelques centièmes de secondes (tableau 11.1, ligne egfr104).

Comparaison du temps d'exécution des vérifications formelles. Afin de souligner l'importante contribution de notre méthode de vérification par analyse statique des propriétés d'atteignabilité, nous comparons les temps d'exécution obtenus avec d'autres outils classiques reposant sur une représentation symbolique de l'espace des états accessibles : Biocham (Fages & Soliman, 2008*b*) qui utilise le vérificateur symbolique NuSMV (Cimatti et coll., 2002) (la traduction des Frappes de Processus en Biocham est directe et s'effectue en temps linéaire) ; et libddd (LIP6/Move, libDDD), un vérificateur symbolique basé sur les diagrammes hiérarchiques de décision (Hamez et coll., 2009), auquel nous donnons en entrée la traduction (exacte) en Réseaux de Petri des Frappes de Processus présentée dans le chapitre 4. Nous rappelons que cette traduction est d'une complexité linéaire suivant la taille des Frappes de Processus.

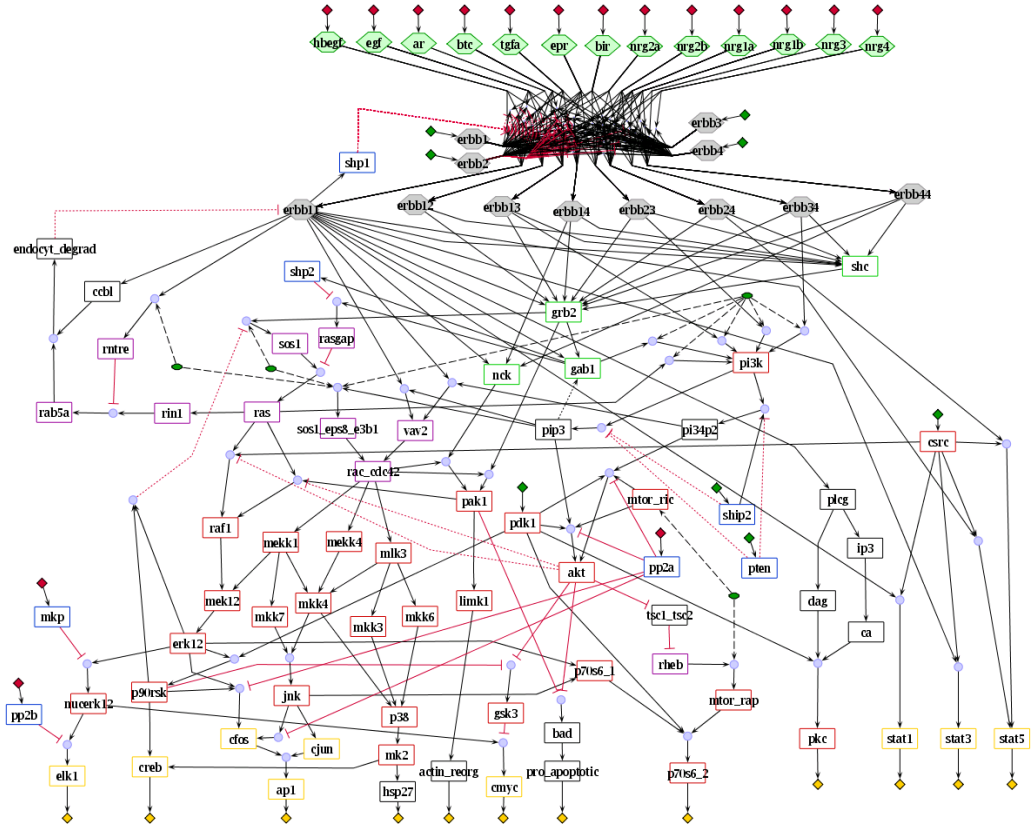


Figure 11.9 – Graphe des interactions du récepteur du facteur de croissance épidermique (104 composants) reproduit depuis (Samaga et coll., 2009).

Modèle	Nb sortes	Nb procs	Nb actions	Nb états	Biocham	libddd	PINT
egfr20	35	196	670	2^{64}	[3s-out]	[1s-150s]	0.007s
tcrsig40	54	156	301	2^{73}	[1s-out]	[0.6s-out]	0.004s
tcrsig94	133	448	1124	2^{194}	out	out	0.030s
egfr104	193	748	2356	2^{320}	out	out	0.050s

Tableau 11.1 – Comparaison des temps d'exécution sur les études de cas entre Biocham (utilisant NuSMV), libddd et PINT. « out » marque une exécution demandant trop de temps ou de mémoire. Lorsque les temps d'exécution varient de manière significative selon les différents états initiaux, les temps minimum et maximum sont écrits entre crochets.

Le tableau 11.1 résume les temps d'exécution obtenus lors de l'analyse des modèles présentés dans cette section sur un ordinateur de bureau classique (processeur 3GHz avec 2GB de mémoire vive). Pour la majorité des décisions, les méthodes utilisant une représentation symbolique de l'espace des états manquent de mémoire et ne répondent pas. Ceci montre l'efficacité remarquable de nos méthodes basées sur l'analyse statique par interprétation abstraite des dynamiques.

11.4 Discussion

Alors que les Frappes de Processus peuvent être vues comme un formalisme restreint pour modéliser un système dynamique, la spécification conjointe de paramètres temporels et stochastiques pour les durées des actions permet de reproduire des phénomènes quantitatifs assez précis. Ceci est illustré par l'application des Frappes de Processus à la modélisation de la segmentation chez les métazoaires et du cycle circadien chez les mammifères : le premier prend avantage du facteur de stochasticité pour réduire la variance temporelle des actions et ainsi reproduire un comportement régulier ; le second prend avantage de la spécification des paramètres par un intervalle de temps afin d'apporter une modélisation compacte et précise d'un phénomène temporel, tout en gardant un cadre général stochastique.

Nous notons toutefois que les résultats sur la modélisation quantitative en Frappes de Processus ne s'appliquent aux grands RRB que dans le cadre de la simulation stochastique du modèle. En effet, la vérification formelle des modèles avec facteur d'absorption de stochasticité reste très coûteuse en pratique (du moins avec le logiciel PRISM).

Concernant la vérification formelle de propriétés sur les dynamiques discrètes, les applications présentées dans la section 11.3 indiquent un apport notable des méthodes par analyses statique des Frappes de Processus développées dans le chapitre 9. Alors que les techniques classiques de vérification formelle échouent à vérifier formellement les modèles étudiés à cause de l'explosion combinatoire des états atteignables, notre analyse statique décide quasiment instantanément la validité de propriétés d'atteignabilité successive de processus.

Ces résultats très encourageants suggèrent une possible application des Frappes de Processus à l'analyse de modèles encore plus grands que ceux étudiés dans ce chapitre.

Plus encore, nos applications démontrent une réelle contribution des analyses statiques pour la vérification de propriétés dynamiques et appellent au développement de ces méthodes pour diversifier le type de propriétés décidables. Le chapitre 12 discute de différentes pistes de recherches envisagées.

Cinquième partie

Perspectives et Conclusion

Le chapitre 12 présente des ouvertures importantes vers lesquelles les résultats apportés dans cette thèse peuvent amener. Nous abordons notamment des pistes de recherche pour obtenir une analyse statique efficace de propriétés quantitatives sur les modèles en Frappes de Processus. Enfin, le chapitre 13 apporte une discussion générale et une conclusion quant aux apports de cette thèse.

Chapitre **12**

Ouvertures

Ce chapitre présente des ouvertures potentielles vers lesquelles amènent les résultats présentés dans cette thèse. Nous commençons par discuter de différentes pistes de recherche afin d'obtenir des analyses statiques efficaces de propriétés quantitatives sur les Frappes de Processus. Nous évoquons notamment l'intérêt d'une analyse statique sur les Frappes de Processus avec Priorités comme étape intermédiaire au développement d'une analyse statique de propriétés quantitatives. Nous détaillons également d'autres perspectives de recherche comme l'aide à l'inférence des paramètres discrets et au contrôle des RRB et l'extension de l'applicabilité des Frappes de Processus, avec notamment l'application aux systèmes de réactions biochimiques.

12.1 Vers l'Analyse Statique de Propriétés Quantitatives des Frappes de Processus

L'introduction de dimensions stochastiques ou temporelles dans les modèles permet une reproduction plus précise du comportement du système étudié. Nous avons présenté dans la sous-section 2.2.3 page 13 différents formalismes connus permettant une modélisation hybride des RRB ; et nous avons étudié dans la partie II l'introduction conjointe du temps et de l'aléatoire dans les modèles et nous l'avons appliqué aux Frappes de Processus.

Les propriétés, dites quantitatives, recherchées sur ces modèles sont généralement du type « quelle est la probabilité d'observer un tel comportement ? », « quel est le temps moyen pour arriver à tel comportement ? », etc. La vérification formelle de telles propriétés a de nombreuses applications lors de l'analyse des modèles de RRB : les comportements engendrés étant très nombreux, leur quantification (au sens des probabilités ou du temps moyen) permet de se concentrer sur les plus vraisemblables. De plus, les aspects temporels, liés aux vitesses des réactions par exemple, peuvent avoir un fort impact sur le devenir des systèmes étudiés ; identifier les composants dont ces paramètres influent particulièrement sur la probabilité ou la durée d'un comportement donné peut se révéler intéressant pour contrôler un tel système.

La quantification exacte de propriétés implique une connaissance précise des comportements satisfaisant la propriété donnée et des comportements ne la satisfaisant pas. Dans le cas stochastique markovien, les modèles sont usuellement traduits en Chaîne de Markov à Temps Continu (CMTC), et des algorithmes classiques de vérification formelle sont appliqués. Cependant une telle approche souffre rapidement de l'explosion combinatoire des comportements à observer. Différentes

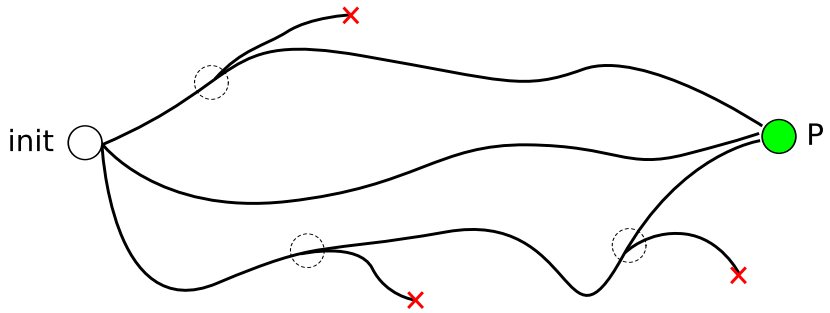


Figure 12.1 – Illustration des comportements satisfaisants (reliés à P) et ne satisfaisant pas (reliés à une croix rouge) une propriété d'atteignabilité P en partant d'un état initial donné (marqué *init*). Les embranchements sont représentés par des cercles en traits discontinus.

techniques peuvent alors être déployées comme la réduction de la CMTC engendrée (par exemple (Feret, Henzinger, Koeppl & Petrov, 2010)) ou l'exploration incomplète des comportements générés (par exemple (Heiner, Rohr, Schwarick & Streif, 2010)) donnant alors une solution approchée de la quantification d'une propriété.

Dans le cadre des Frappes de Processus, une piste de recherche serait l'incorporation de données quantitatives lors de l'analyse statique présentée dans le chapitre 9 pour extraire, par exemple, la probabilité d'observer une succession d'atteignabilités donnée, ou d'en calculer son temps moyen. Une telle approche résulterait en des approximations des quantifications recherchées.

Comme illustré dans la figure 12.1, la quantification des comportements satisfaisant une propriété d'atteignabilité P requiert l'identification des « embranchements » amenant à la non-satisfiabilité de P, ceux-ci résultant d'une concurrence entre plusieurs actions.

Une première étape pourrait alors étendre l'analyse statique du chapitre 9 aux Frappes de Processus *avec Priorités* (sous-section 4.3.1 page 49). L'introduction de priorités peut être considérée comme analogue à l'introduction d'une concurrence entre les actions basée sur leur vitesse (ou probabilité), à la différence notable que cette compétition s'effectue de manière discrète et fixe (c'est toujours la même action qui gagne, celle de plus forte priorité). Afin d'obtenir des approximations supérieures et inférieures efficaces, la notion de priorité devrait apparaître dans la structure abstraite sur laquelle notre analyse statique s'effectue, et ainsi donner des indices sur la présence d'une concurrence entre des actions pouvant mener à la non-satisfiabilité d'une propriété.

Nous illustrons cette idée par un petit exemple de Frappes de Processus (figure 12.2), où deux actions sont en concurrence : $a_0 \to b_0 \not\upharpoonright b_1$ et $b_0 \to a_0 \not\upharpoonright a_1$; nous nous intéressons alors à l'atteignabilité de b_1. Une analyse sans prise en compte des priorités aurait simplement extrait la structure abstraite contenant l'objectif $b_0 \not\upharpoonright^* b_1$ et $a_0 \not\upharpoonright^* a_0$ et aurait conclu positivement sur l'atteignabilité de b_1. En prenant en compte les priorités, nous mettons en concurrence la présence du processus a_0 et l'objectif $a_0 \not\upharpoonright^* a_1$ (représenté par un arc terminé par une barre) : si l'objectif $a_0 \not\upharpoonright^* a_1$ se réalise avec une priorité supérieure à celle de l'action de a_0 sur b_0, alors l'objectif $b_0 \not\upharpoonright^* b_1$ n'est pas réalisable.

Après la généralisation de ce raisonnement, le défi sera d'établir une extension de l'analyse des Frappes de Processus avec Priorité à l'analyse des Frappes de Processus markoviennes (c.-à-d. sans facteur d'absorption de stochasticité), puis à celle des Frappes de Processus avec absorption de stochasticité (impliquant alors des phénomènes d'accumulation temporelle).

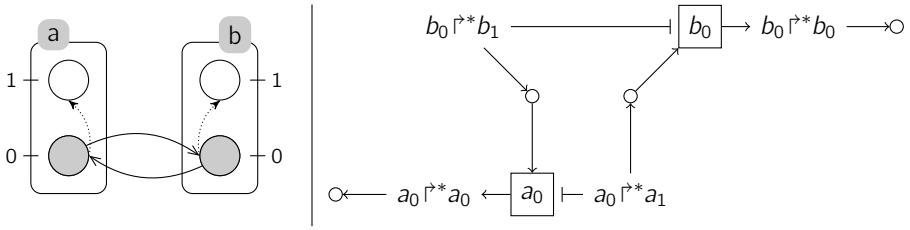


Figure 12.2 – (gauche) Exemple de Frappes de Processus ; (droite) Structure abstraite incorporant les concurrences entre la présence du processus a_0 et l'objectif $a_0 \Rightarrow^* a_1$ et la présence de b_0 et l'objectif $b_0 \Rightarrow^* b_1$.

12.2 Autres Perspectives

12.2.1 Vers l'Inférence des Paramètres Discrets et le Contrôle des RRB

Dans le chapitre 3, introduisant les Frappes de Processus et leur application aux RRB, nous avons exhibé une méthode de modélisation des RRB par raffinement : partant du seul graphe des interactions, nous construisons sa dynamique généralisée en Frappes de Processus ; puis à l'aide d'observations ou de connaissances sur les paramètres discrets régissant les fonctions entre les composants, nous raffinons cette dynamique en supprimant des actions et en spécifiant des coopérations entre des processus.

Ces raffinements peuvent alors être interprétés comme des contraintes ou des valeurs pour les paramètres discrets des composants (sortes) concernées par le procédé de raffinement. Dans cette thèse, nous avons notamment illustré ce raffinement lors de la modélisation de la segmentation chez les métazoaires (sous-section 11.2.1 page 170). Nous soulignons que l'approche employée est ici manuelle. Toutefois, un processus automatique de raffinement de Frappes de Processus pourrait être étudié, notamment lors de la présence de données expérimentales permettant d'attester l'absence de certains comportements. Le schéma général d'une telle inférence pourrait être le suivant :

1. Construction de la dynamique généralisée à partir du graphe des interactions (aucun paramètre discret connu).
2. Vérification de l'absence d'un comportement donné.
3. Si le comportement est présent, raffinement des Frappes de Processus afin de le supprimer.
4. Recommencer à l'étape 2 pour chaque propriété à vérifier.

Bien entendu, ce procédé très général soulève de nombreuses questions, notamment sur la construction du raffinement automatique et de son impact sur les comportements supprimés, et dont les réponses dépendent vraisemblablement du type de propriétés utilisées pour l'inférence.

Concernant les propriétés de point fixe, les méthodes développées dans le chapitre 8 (et plus particulièrement dans la section 8.3) peuvent permettre d'extraire les actions responsables de l'absence d'un point fixe voulu. Il est alors possible de construire une coopération entre les processus concernés afin de créer ce point fixe avec un minimum d'impact sur la dynamique obtenue. Ce procédé de raffinement a également été illustré dans l'application en sous-section 11.2.1.

Enfin, concernant les propriétés d'atteignabilité, les méthodes d'analyses développées dans le chapitre 9 permettent d'extraire les processus clés à la réalisation d'une propriété d'atteignabilité : l'absence d'un tel processus conduit à l'impossibilité de l'atteignabilité donnée. Alors, une

telle propriété peut être supprimée en bloquant l'activation d'un processus clé (impactant alors les paramètres discrets). Un tel procédé est une des bases de la thérapie génique, où certains traitements ont pour but de bloquer l'expression d'un certain gène (processus dit de *knockdown*). Nous soulignons que les processus clés susmentionnés sont requis indépendamment ; il pourrait alors être intéressant d'extraire des couples (ou plus généralement des tuples) de processus clés.

À plus long terme, et en complément des perspectives discutées dans la section 12.1, les analyses statiques opérées sur les Frappes de Processus devraient pouvoir extraire des processus clés à la réalisation de propriétés quantitatives. Ainsi, de manière analogue au cas discret, ces processus clés pourraient aider à l'inférence des paramètres quantitatifs des actions en Frappes de Processus afin de contrôler la réalisation d'une propriété.

12.2.2 Extension de l'Applicabilité des Frappes de Processus

Nous pouvons différencier deux voies permettant d'étendre le champs d'application des Frappes de Processus.

D'une part, les types de propriétés vérifiables statiquement pourraient être élargies, avec un accent particulier sur les propriétés ayant un intérêt pour la modélisation et la compréhension de systèmes biologiques. Parmi ces propriétés, nous pouvons citer, par exemple, la caractérisation des différents attracteurs des dynamiques (ensemble d'états clos par la relation de transitions) avec notamment l'extraction des attracteurs minimaux ; ou encore le temps attendu pour atteindre un attracteur particulier.

D'autre part, la généralisation des méthodes développées à des extensions des Frappes de Processus pourrait être envisagée. La vérification statique des Frappes de Processus avec Priorités (envisagée également dans la section 12.1) permettrait, par exemple, l'analyse exacte de modèles en Réseaux de Petri ou en Automates Finis Communicants (voir chapitre 4). Une autre approche pourrait tenter d'appliquer nos méthodes sur des formalismes plus complexes que les Frappes de Processus, supportant par exemple des actions avec une arité arbitraire, ou changeant plusieurs processus de manière synchrone.

12.2.3 Application aux Systèmes de Réactions Biochimiques

Toujours dans la perspective d'augmenter l'applicabilité des méthodes développées dans cette thèse autour des Frappes de Processus, nous montrons informellement dans cette sous-section une traduction d'un ensemble de réactions biochimiques (telles qu'elles seraient définies dans Biocham (Fages & Soliman, 2008*b*), par exemple) en Frappes de Processus. Intuitivement (nous ne prouvons pas cette assertion), les dynamiques obtenues sont proches des dynamiques obtenues en Biocham avec une sémantique booléenne des réactions (Fages & Soliman, 2008*a*). Une telle traduction présente donc un intérêt certain pour l'analyse de tels systèmes via les Frappes de Processus.

Un ensemble de réactions est défini par $E = \{e_1, \ldots, e_n\}$, où e_i est une réaction : on note $r(e_i)$ l'ensemble des réactants et $p(e_i)$ l'ensemble des produits de la réaction $e_i \in E$. En particulier, dans le cas de Biocham :

- $r(A + B \to C + D) = \{A, B\}$; $p(A + B \to C + D) = \{C, D\}$;
- $r(A + B[= X] \to C + D) = \{A, B, X\}$; $p(A + B[= X] \to C + D) = \{C, D, X\}$.

Interprétation en Frappes de Processus.

Sortes et processus :

- Pour chaque équation $e_i \in E$, nous créons une sorte (e_i) contenant deux processus $(e_i)_1$ et $(e_i)_0$ représentant respectivement l'activité et l'inactivité de l'équation e_i.
- Pour chaque entité A existante ($\exists e_i \in E : A \in r(e_i) \vee A \in p(e_i)$), nous créons une sorte A contenant deux processus A_1 et A_0 représentant respectivement la présence et l'absence de l'entité A.
- Pour chaque ensemble de réactants $\{A, B, \ldots\}$ ($\exists e_i \in E : r(e_i) = \{A, B, \ldots\}$), nous créons une sorte $\{A, B, \ldots\}$ représentant la fonction booléenne $A_1 \wedge B_1 \wedge \ldots$.

Frappes :

- (production) Pour chaque processus $(e_i)_1$, pour tout nouveau produit $A \in p(e_i) \setminus r(e_i)$, nous créons la frappe $(e_i)_1 \to A_0 \Rsh A_1$.
- (consommation) Pour chaque processus $(e_i)_1$, pour tout réactant consommé $A \in r(e_i) \setminus p(e_i)$, nous créons la frappe $(e_i)_1 \to A_1 \Rsh A_0$.
- (activation et désactivation) Pour tout ensemble de réactants $\{A, B, \ldots\}$, nous appelons $\{A, B, \ldots\}_1$ le processus représentant la valeur vraie de la fonction booléenne $A_1 \wedge B_1 \wedge \ldots$. Pour tout $e_i \in E : r(e_i) = \{A, B, \ldots\}$, nous créons la frappe $\{A, B, \ldots\}_1 \to (e_i)_0 \Rsh (e_i)_1$. De la même manière, pour tout processus $\{A, B, \ldots\}_0$ représentant la valeur fausse, nous créons la frappe $\{A, B, \ldots\}_0 \to (e_i)_1 \Rsh (e_i)_0$.
- (fonctions booléennes) Enfin, il faut ajouter les frappes des entités (A, B, etc.) vers les ensembles de réactants les contenant afin de mettre à jour la valeur de la fonction booléenne.

Remarque : afin d'optimiser la taille du modèle en Frappes de Processus, nous pouvons avoir recours à une factorisation des sortes coopératives, en suivant par exemple le procédé illustré dans la figure 3.3 page 32.

Exemple. Soit $E = \{e_1 = A + B \to C; e_2 = C \to A + B\}$.
L'ensemble des sortes obtenues par notre traduction est $\Sigma = \{(e_1), (e_2), A, B, C, \{A, B\}\}$, les processus sont $(e_1)_0, (e_1)_1$, $(e_2)_0, (e_2)_1$, A_0, A_1, B_0, B_1, C_0, C_1, $\{A, B\}_{00}, \{A, B\}_{01}, \{A, B\}_{10}$, et $\{A, B\}_{11}$. Enfin, les frappes obtenues sont :

- (production) $(e_1)_1 \to C_0 \Rsh C_1$, $(e_2)_1 \to A_0 \Rsh A_1$, $(e_2)_1 \to B_0 \Rsh B_1$;
- (consommation) $(e_1)_1 \to A_1 \Rsh A_0$, $(e_1)_1 \to B_1 \Rsh B_0$, $(e_2)_1 \to C_1 \Rsh C_0$;
- (activation et désactivation) $\{A, B\}_{11} \to (e_1)_0 \Rsh (e_1)_1$, $C_1 \to (e_2)_0 \Rsh (e_2)_1$, $\{A, B\}_{00} \to (e_1)_1 \Rsh (e_1)_0$, $\{A, B\}_{01} \to (e_1)_1 \Rsh (e_1)_0$, $\{A, B\}_{10} \to (e_1)_1 \Rsh (e_1)_0$, $C_0 \to (e_2)_1 \Rsh (e_2)_0$;
- (fonctions booléennes) $A_1 \to \{A, B\}_{00} \Rsh \{A, B\}_{10}$, $A_1 \to \{A, B\}_{01} \Rsh \{A, B\}_{11}$, $B_1 \to \{A, B\}_{00} \Rsh \{A, B\}_{01}$, $B_1 \to \{A, B\}_{10} \Rsh \{A, B\}_{11}$, $A_0 \to \{A, B\}_{10} \Rsh \{A, B\}_{00}$, $A_0 \to \{A, B\}_{11} \Rsh \{A, B\}_{01}$, $B_0 \to \{A, B\}_{01} \Rsh \{A, B\}_{00}$, $B_0 \to \{A, B\}_{11} \Rsh \{A, B\}_{10}$.

Chapitre **13**

Conclusion

Les Réseaux de Régulation Biologique (RRB) sont communément utilisés en biologie des systèmes afin de modéliser et de comprendre les dynamiques de production des protéines au sein des cellules (Thomas, 1973). Ces dynamiques sont généralement spécifiées par des fonctions discrètes définissant le prochain état d'un composant selon l'état courant de ses régulateurs (voir chapitre 2).

Bien qu'offrant une représentation très abstraite des systèmes biologiques, et en particulier se limitant à un espace fini des états, cette modélisation discrète souffre d'une explosion combinatoire des comportements engendrés limitant les outils classiques de vérification formelle à l'étude de RRB de taille restreinte.

Dans cette thèse, nous avons apporté une contribution pour la modélisation, la simulation et la vérification formelle de propriétés discrètes de grands RRB à travers la définition d'un nouveau formalisme, les *Frappes de Processus*. Nous avons tout juste abordé le domaine du contrôle de ces RRB, ce qui consitue un enjeu majeur dans la perspective des thérapies géniques. Nous pensons toutefois avoir ouvert une voie efficace et réaliste pour la prise en compte de réseaux à très grande échelle et par conséquent extrêmement complexes.

En effet, basée sur les Frappes de Processus, une analyse statique très efficace par interprétation abstraite a été développée afin de rendre possible la vérification formelle de propriétés d'atteignabilité au sein de Frappes de Processus. D'une complexité théorique limitée (au prix de potentielles absences de conclusion), cette approche promet de supporter l'analyse de très grands RRB, et supplante les outils actuels de l'état de l'art.

Une technique générique de modélisation a été également présentée afin de proposer un compromis entre une spécification stochastique et temporelle dans les modélisations hybrides. Ceci facilite une certaine flexibilité dans la modélisation de systèmes où les contraintes temporelles et le comportement stochastique sont des caractéristiques essentielles.

Ainsi, cette thèse apporte les contributions clés suivantes :

Introduction des Frappes de Processus (partie I). Le formalisme des Frappes de Processus se veut simple et permet la modélisation de systèmes complexes. En Frappes de Processus, les processus sont séparés en un ensemble fini de *sortes*. À tout temps, un et un seul processus de chaque sorte est présent ; un processus peut être remplacé par un autre processus de même sorte par la *frappe* d'un autre processus présent (chapitre 3).

Les Frappes de Processus sont particulièrement adaptées pour la modélisation des RRB en offrant une traduction directe de tels systèmes et en permettant différents niveaux d'abstraction

pour la spécification des dynamiques, autorisant notamment une connaissance partielle des fonctions entre les composants.

Au passage, nous pensons que ce type de formalisation pourrait également s'avérer adapté à d'autres systèmes complexes présentant des caractéristiques plus ou moins analogues aux RRB. En cela, la bio-informatique n'est pas seulement l'application de l'informatique aux sciences du vivant, mais elle contribue également à apporter des avancées significatives à la science informatique.

Les relations entre les Frappes de Processus et les formalismes existants sont étudiées dans le chapitre 4. Il apparaît, entre autres, que les Frappes de Processus sont une restriction de nombreux formalismes. La traduction proposée des Automates Finis Communicants en Frappes de Processus avec Priorités ouvre également la perspective d'encoder des formalismes classiques en Frappes de Processus.

En étant strictement moins expressives que les autres formalismes couramment employés, les Frappes de Processus permettent d'obtenir des vérifications formelles par des analyses statiques très efficaces.

Introduction de paramètres temporels et stochastiques dans les modèles (partie II). Dans le but de préciser des caractéristiques temporelles au sein des modèles stochastiques, nous avons introduit la notion de *facteur d'absorption de stochasticité* dans le cadre générique du π-calcul stochastique (chapitre 5). Ce facteur d'absorption, spécifié avec le taux d'utilisation d'une action, agit comme un réducteur de la variance temporelle : au plus ce facteur est élevé, au plus la variance autour du temps moyen imposé par le taux de l'action diminue. La durée des actions suit alors la distribution d'Erlang. Des estimateurs ont été produits afin de traduire un intervalle de temps en une spécification stochastique, pour un niveau de confiance donné.

En ce sens, la modélisation de systèmes possédant un comportement stochastique inhérent, tout en subissant des contraintes temporelles importantes, a été grandement simplifiée par le biais de cette technique. C'est notamment le cas pour la modélisation des RRB en Frappes de Processus (chapitre 7), où le niveau d'abstraction des modélisations requiert la spécification de contraintes temporelles tout en conservant un comportement global aléatoire.

Nous avons également montré une construction du π-calcul avec absorption de stochasticité dans le π-calcul stochastique markovien classique (chapitre 5), illustrant ainsi l'applicabilité des outils d'analyses se basant sur les chaînes de Markov à temps continu, tel PRISM (`http://www.prismmodelchecker.org`), un vérificateur de modèles probabilistes.

Enfin, nous avons présenté une machine abstraite générique pour la simulation markovienne et non-markovienne des calculs de processus (chapitre 6), développée en collaboration avec Andrew Phillips et Matthew Lakin (Microsoft Research, Cambridge, UK). Utilisant le principe de compilation « à la volée », cette machine peut être instanciée pour de nombreux calculs de processus en spécifiant seulement certaines fonctions propres à sa sémantique. Cette machine apporte ainsi la simulation efficace des Frappes de Processus avec absorption de stochasticité (chapitre 7).

Analyse statique des Frappes de Processus (partie III). La structure particulière des Frappes de Processus permet la dérivation statique de propriétés sur les dynamiques engendrées.

Nous avons montré dans le chapitre 8 que les points fixes de la dynamique peuvent être obtenus par une simple analyse structurelle (recherche de cliques dans le graphe complémentaire des frappes). En collaboration avec Adrien Richard (CNRS & I3S, Nice, France), ce résultat a été porté à la

dérivation de certains points fixes particuliers directement depuis le graphe des interactions des RRB.

Enfin, en utilisant des techniques d'interprétation abstraite, une analyse statique originale a pu être développée afin de dériver des propriétés d'atteignabilité successive de processus au sein des Frappes de Processus (chapitre 9). Notamment, cette analyse possède une complexité limitée (polynomiale selon le nombre de sortes et exponentielle selon le nombre de processus dans une seule sorte), apportant ainsi une gestion prometteuse de l'analyse de grands systèmes. L'extraction des processus nécessaires à la satisfaction d'une propriété d'atteignabilité vient directement avec l'analyse statique proposée.

Cette analyse extrait différentes abstractions des Frappes de Processus, et en exploitant leur complémentarité, raffine une spécification donnée d'une propriété d'atteignabilité, résultant finalement en des approximations supérieures et inférieures de la décision de satisfiabilité.

Application aux Réseaux de Régulation Biologique (partie IV). Le logiciel PINT (`http://process.hitting.free.fr`) implémente les résultats de cette thèse autour des Frappes de Processus (chapitre 10) et se veut être une base pour les futurs développements autour des Frappes de Processus.

Nous avons illustré dans le chapitre 11 l'application des Frappes de Processus à l'étude des RRB : utilisation des différents niveaux d'abstraction proposés en Frappes de Processus (méthode de modélisation par raffinement) ; spécification de contraintes stochastiques et temporelles ; analyse des points fixes et de propriétés d'atteignabilité discrète.

En particulier, l'analyse statique des atteignabilités a été appliquée avec succès sur des Frappes de Processus modélisant des RRB regroupant jusqu'à plus de cent composants. Les décisions sont calculées très rapidement (autour du centième de seconde) alors que les outils classiques (utilisant une représentation symbolique de l'espace des états) échouent régulièrement du fait de l'explosion combinatoire des comportements engendrés par les Frappes de Processus.

Cette thèse ouvre de nouvelles perspectives (étayées dans le chapitre 12) concernant l'analyse à grande échelle des RRB : en utilisant la simplicité des Frappes de Processus, l'analyse statique de propriétés dynamiques précises est rendue possible. Des nouveaux travaux s'attellent au développement de nouvelles analyses statiques permettant la dérivation d'autres propriétés des dynamiques, et notamment la dérivation efficace de propriétés quantitatives, ceci pouvant avoir un autre impact notable pour l'analyse de systèmes biologiques.

Annexe **A**

Instanciations Supplémentaires de la Machine Abstraite Générique

Nous détaillons ici l'instanciation et la correction de la machine abstraite générique présentée dans le chapitre 6 pour le calcul bioambiant et κ.

A.1 Instanciation avec le Calcul Bioambiant

Le calcul bioambiant a été présenté originellement par Regev et coll. (2004) pour la modélisation de compartiments mobiles dans les processus biologiques. Dans cette section, nous instancions la machine abstraite générique pour une restriction du calcul bioambiant stochastique : nous donnons les définitions complètes pour le calcul bioambiant sans l'action `fusion`, et nous discutons en fin de sous-section une extension de la machine pour incorporer cette action `fusion`.

A.1.1 Syntaxe et Sémantique du Calcul

La syntaxe et les règles de réduction du calcul bioambiant stochastique utilisées dans cette section sont présentées dans la définition A.1 et sont basées sur celles de Phillips (2009). Un processus P peut être un choix d'actions C, une instance $X(\tilde{n})$ d'une définition X avec les paramètres $\tilde{n}$, une composition parallèle de processus $P \mid Q$, un processus $\nu x\, P$ avec un canal privé x, ou un ambiant $\boxed{P}^{a}$ consistant en un processus P dans le compartiment nommé a. Un choix C consiste en une compétition entre zéro ou plusieurs actions $\pi^{i}.P$, comme en π-calcul stochastique. Une action π peut être un délai τ_r, un envoi $\gamma!x(\tilde{n})$ de valeurs $\tilde{n}$ sur un canal x, ou une réception $\gamma?x(\tilde{m})$ de valeurs $\tilde{m}$ sur un canal x, où γ dénote le type de communication. Ceci peut se dérouler à l'intérieur d'un seul ambiant (`local`), depuis un ambiant voisin vers un autre (`v-à-v`), d'un ambiant enfant vers son parent (`e-à-p`), ou d'un ambiant parent vers un enfant (`p-à-e`). En supplément, une action π peut être un déplacement $\mu!x$ sur un canal x ou une validation $\mu?x$ sur un canal x, où μ dénote le type de mouvement : un ambiant entrant dans un de ses voisins (`entrée`), un ambiant quittant son parent (`sortie`), ou la fusion de deux ambiants voisins (`fusion`). Un environnement E consiste en un ensemble de définitions $X(\tilde{m}) \mapsto P$, comme en π-calcul stochastique.

Nous dérivons une sémantique en CMTC pour le calcul bioambiant directement de sa relation de réduction, en définissant une *forme indicée* pour les processus, basée sur celle de Phillips (2009).

Définition A.1 (Syntaxe et règles de réduction pour le calcul bioambiant stochastique)**.** Nous écrivons r_x pour $\mathsf{taux}(x)$. Chaque réduction est libellée avec son taux r et un indice w qui est une liste d'identifiants dénotant les actions et les ambiants impliqués dans la réduction.

$$
\begin{array}{lll}
P, Q ::= & C \mid X(\tilde{n}) \mid P|Q \mid \nu x\, P \mid \boxed{P}^a & \text{Processus} \\
C ::= & \pi_1^{i_1}.P_1 + \ldots + \pi_N^{i_N}.P_N & \text{Choix} \\
E ::= & X_1(\tilde{m}_1) \mapsto P_1, \ldots, X_N(\tilde{m}_N) \mapsto P_N & \text{Environnement}
\end{array}
$$

$$
\begin{array}{llll}
\pi ::= & \tau_r & \text{Délai} & \\
\mid & \gamma!x(\tilde{n}) & \text{Envoi} & \\
\mid & \gamma?x(\tilde{m}) & \text{Réception} & \\
\mid & \mu!x & \text{Déplacement} & \\
\mid & \mu?x & \text{Validation} & \\
\lambda ::= & \gamma \mid \mu & \text{Actions binaires} &
\end{array}
\qquad
\begin{array}{llll}
\gamma ::= & \texttt{local} & \text{Local} & \\
\mid & \texttt{v-à-v} & \text{Voisin} & \\
\mid & \texttt{e-à-p} & \text{Parent} & \\
\mid & \texttt{p-à-e} & \text{Enfant} & \\
\mu ::= & \texttt{entrée} & \text{Entrer} & \\
\mid & \texttt{sortie} & \text{Quitter} & \\
\mid & \texttt{fusion} & \text{Fusionner} &
\end{array}
$$

$$
\begin{array}{rcl}
\tau_r^i.P + C & \xrightarrow{r,i} & P \\
\texttt{local}!x(\tilde{n})^i.P + C \mid \texttt{local}?x(\tilde{m})^{i'}.P' + C' & \xrightarrow{r_x,(i,i')} & P \mid P'_{\{\tilde{n}/\tilde{m}\}} \\
\boxed{Q \mid \texttt{e-à-p}!x(\tilde{n})^i.P + C}^a \mid Q' \mid \texttt{e-à-p}?x(\tilde{m})^{i'}.P' + C' & \xrightarrow{r_x,(i,a,i')} & \boxed{Q \mid P}^a \mid Q' \mid P'_{\{\tilde{n}/\tilde{m}\}} \\
Q \mid \texttt{p-à-e}!x(\tilde{n})^i.P + C \mid \boxed{Q' \mid \texttt{p-à-e}?x(\tilde{m})^{i'}.P' + C'}^a & \xrightarrow{r_x,(i,a,i')} & Q \mid P \mid \boxed{Q' \mid P'_{\{\tilde{n}/\tilde{m}\}}}^a \\
\boxed{Q \mid \texttt{v-à-v}!x(\tilde{n})^i.P + C}^a \mid \boxed{Q' \mid \texttt{v-à-v}?x(\tilde{m})^{i'}.P' + C'}^b & \xrightarrow{r_x,(i,a,i',b)} & \boxed{Q \mid P}^a \mid \boxed{Q' \mid P'_{\{\tilde{n}/\tilde{m}\}}}^b \\
\boxed{Q \mid \texttt{entrée}!x^i.P + C}^a \mid \boxed{Q' \mid \texttt{entrée}?x^{i'}.P' + C'}^b & \xrightarrow{r_x,(i,a,i',b)} & \boxed{\boxed{Q \mid P}^a \mid Q' \mid P'}^b \\
\boxed{\boxed{Q \mid \texttt{sortie}!x^i.P + C}^a \mid Q' \mid \texttt{sortie}?x^{i'}.P' + C'}^b & \xrightarrow{r_x,(i,a,i',b)} & \boxed{Q \mid P}^a \mid \boxed{Q' \mid P'}^b \\
\boxed{Q \mid \texttt{fusion}!x^i.P + C}^a \mid \boxed{Q' \mid \texttt{fusion}?x^{i'}.P' + C'}^b & \xrightarrow{r_x,(i,a,i',b)} & \boxed{Q \mid P \mid Q' \mid P'}^b \\
P \xrightarrow{r,w} P' \quad \Rightarrow \quad \boxed{P}^a & \xrightarrow{r,w} & \boxed{P'}^a \\
P \xrightarrow{r,w} P' \quad \Rightarrow \quad \nu x\, P & \xrightarrow{r,w} & \nu x\, P' \\
P \xrightarrow{r,w} P' \quad \Rightarrow \quad P \mid Q & \xrightarrow{r,w} & P' \mid Q \\
Q \equiv P \xrightarrow{r,w} P' \equiv Q' \quad \Rightarrow \quad Q & \xrightarrow{r,w} & Q'
\end{array}
$$

Définition A.2 (Forme indicée d'un processus)**.** Un processus P est sous *forme indicée* si il est de la forme $\nu x_1 .. \nu x_M (C_1 \mid .. \mid C_N \mid \boxed{P_1}^{a_1} \mid .. \mid \boxed{P_K}^{a_K})$ où chaque processus $P_1, .., P_K$ est sous forme indicée, chaque indice d'ambiant $a_1, .., a_K$ est unique, et chaque action non gardée π^i est associée à un indice unique i.

Nous pouvons montrer que tout processus est structurellement congruent à un processus sous forme indicée, au renommage des indices près (la preuve est directe (Phillips, 2009)). La sémantique de la réduction en CMTC est définie par la règle suivante, où P est un processus sous forme indicée :

$$\frac{a = \left(\sum_{\{\lambda, w \mid P \xrightarrow{\exp(\lambda), w} P'\}} \lambda\right) > 0}{P \xrightarrow{a} P'} \tag{A.1}$$

A.1.2 Extraire les espèces et les réactions des processus

La particularité la plus importante du calcul bioambiant est la compartimentation des processus dans une hiérarchie d'*ambiants*, prévenant certaines réactions locales entre les processus résidant dans différents ambiants. Nous extrayons un ensemble plat de réactions à partir d'un processus bioambiant en annotant les processus avec l'identifiant de l'ambiant dans lequel ils résident. Nous supposons donc que chaque ambiant est associé à un identifiant unique (définition A.2). La structure hiérarchique des ambiants est spécifiée par les espèces de *localisation*, où la localisation (a, b) signifie que l'ambiant a est un enfant de l'ambiant b. L'identifiant racine dénote l'ambiant de plus haut niveau (l'ambiant racine). Par exemple, le processus $\boxed{P_1 \mid \boxed{P_2}^a}^b$ est traduit par le multi-ensemble d'espèces $P_1^b, P_2^a, (a, b), (b, \text{racine})$ où racine est l'identifiant de l'ambiant racine, et a et b sont des identifiants supposés uniques. L'assignation des localisations des espèces dans la définition de la fonction espèces(P) est formellement présentée dans la définition A.3, où la fonction processus$(\overline{l})$ retourne le processus correspondant au multi-ensemble d'espèces $\overline{l}$.

Le calcul des réactions entre les espèces est présenté dans la définition A.4. Les règles de réduction du calcul bioambiant sont directement employées pour calculer l'ensemble des réactions entre les espèces l_1 et l_2 avec leur localisation correspondante. Par exemple,

$$\boxed{Q_1 \mid \texttt{entrée!}x.P_1}^a \mid \boxed{Q_2 \mid \texttt{entrée?}x.P_2}^b \quad \longrightarrow \quad \boxed{\boxed{Q_1 \mid P_1}^a \mid Q_2 \mid P_2}^b$$

est traduit dans la machine abstraite par la réaction suivante :

$$\texttt{entrée!}x.P_1^a + \texttt{entrée?}x.P_2^b + (a, \text{racine}) + (b, \text{racine}) \longrightarrow P_1^a + P_2^b + (a, b) + (b, \text{racine})$$

Comme la population d'une espèce de localisation est toujours 0 ou 1, l'application d'une telle réaction désactiverait les réactions impliquant l'ancienne localisation (a, racine) de l'ambiant a, et les réactions impliquant la nouvelle localisation (a, b) deviendraient possibles.

Nous notons que nous n'incluons pas explicitement les fonctions pour supporter l'opération `fusion` dans la définition A.3. Ceci peut être ajouté en étendant les espèces de localisation avec la notion d'*alias d'ambiant* $(a = b)$, qui spécifie que a est fusionné avec l'ambiant b. À la suite d'une fusion, tous les identifiants d'ambiant a doivent être remplacés par b. Ceci peut être effectué

Définition A.3 (Fonctions de conversion entre les espèces et les processus)**.**

$$
\begin{array}{rcll}
I & ::= & X(\tilde{n})^a & \text{Espèce processus} \\
 & | & L & \text{Espèce localisation} \\
L & ::= & (a, b) & \text{Localisation d'un ambiant}
\end{array}
$$

$$
\begin{array}{rcl}
\text{espèces}(\mathbf{0}) & \triangleq & \emptyset \\
\text{espèces}(P) & \triangleq & \text{espèces}(P, \text{racine}) \\
\text{espèces}(\boxed{P}^{a'}, a) & \triangleq & \text{espèces}(P, a') \uplus [(a', a)] \\
\text{espèces}(X(\tilde{n}), a) & \triangleq & [X(\tilde{n})^a] \quad \text{si } E(X(\tilde{n})) = C \\
\text{espèces}(X(\tilde{n}), a) & \triangleq & \text{espèces}(P, a) \quad \text{si } E(X(\tilde{n})) = P \neq C \\
\text{espèces}(\nu x\, P, a) & \triangleq & \text{espèces}(P_{\{y/x\}}, a) \quad \text{si unique}(y) \\
\text{espèces}(P_1 \mid P_2, a) & \triangleq & \text{espèces}(P_1, a) \uplus \text{espèces}(P_2, a)
\end{array}
$$

$$
\begin{array}{rcl}
\text{processus}(\overline{I}) & \triangleq & \text{processus}(\overline{I}, \text{racine}) \\
\text{processus}(\overline{I}, \text{racine}) & \triangleq & \text{parallèle}(\overline{J} \uplus [\boxed{\text{processus}((\overline{I} \uplus \overline{J}), b)}^{b} \mid (b, a) \in \overline{I}]) \\
 & & \text{si } \overline{J} = [X(\tilde{n}) \mid X(\tilde{n})^a \in \overline{I}]
\end{array}
$$

où $\text{parallèle}(I_1, \ldots, I_N)$ dénote le processus $I_1 \mid \cdots \mid I_N$ et $\overline{I} \uplus \overline{J}$ la différence entre deux multi-ensembles.

Définition A.4 (Instanciation de la machine générique avec le calcul bioambiant)**.**

$$
\begin{array}{rcl}
\text{réactions}(I, \widetilde{I'}) & \triangleq & \text{unaire}(I) \uplus \text{n-aire}(I, (\widetilde{I'} \cup \{I\})) \\
\text{unaire}(I) & \triangleq & [([I], r, \text{espèces}(P, a)) \mid [I] \xrightarrow{r,w} P \wedge I = X(\tilde{n})^a] \\
\text{n-aire}(I, \widetilde{I'}) & \triangleq & \text{2-aire}(I, \widetilde{I'}) \uplus \text{3-aire}(I, \widetilde{I'}) \uplus \text{4-aire}(I, \widetilde{I'}) \\
\text{2-aire}(I_1, \widetilde{I'}) & \triangleq & [(\overline{I}, r, \text{espèces}(P, a)) \mid \text{processus}(\overline{I}) \xrightarrow{r,(i_1,i_2)} P \\
 & & \quad \wedge \overline{I} = [I_1, I_2] = [X(\tilde{n})^a, Y(\tilde{m})^a] \wedge I_2 \in \widetilde{I'}] \\
\text{3-aire}(I_1, \widetilde{I'}) & \triangleq & [(\overline{I}, r, \text{espèces}(P, b)) \mid \text{processus}(\overline{I}) \xrightarrow{r,(i_1,a,i_2)} P \wedge a \neq b \\
 & & \quad \wedge \overline{I} = [I_1, I_2, I_3] = [X(\tilde{n})^a, Y(\tilde{m})^b, (a, b)] \wedge [I_2, I_3] \subseteq \widetilde{I'}] \\
\text{4-aire}(I_1, \widetilde{I'}) & \triangleq & [(\overline{I}, r, \text{espèces}(P, c)) \mid \text{processus}(\overline{I}) \xrightarrow{r,(i_1,a,i_2,c)} P \\
 & & \quad \wedge a \neq b \wedge a \neq c \wedge (d = b \vee d = c) \wedge [I_2, I_3, I_4] \subseteq \widetilde{I'} \\
 & & \quad \wedge \overline{I} = [I_1, I_2, I_3, I_4] = [X(\tilde{n})^a, Y(\tilde{m})^b, (a, d), (b, c)]]
\end{array}
$$

Nous notons que dans le cas de 2-aire, 3-aire et 4-aire, nous exploitons le fait que l'ordre des éléments à l'intérieur d'un multi-ensemble n'est pas fixé, par exemple dans 2-aire il est possible que $I_1 = Y(\tilde{m})^a$ et $I_2 = X(\tilde{n})^a$.

en définissant une *forme normale* pour les espèces qui supprime les alias d'ambiant en substituant les identifiants correspondant. Par exemple, considérons le processus $\boxed{A \mid X}^a \mid \boxed{B \mid Y}^b$ avec $X = \texttt{fusion}!x^i.X_2$ et $Y = \texttt{fusion}?x^{i'}.Y_2$. La réaction suivante est possible :

$$X^a + Y^b + (a, root) + (b, root) \longrightarrow X_2^b + Y_2^b + (a = b) + (b, root)$$

Après application de cette réaction, l'ensemble des espèces est $[X_2^b, Y_2^b, A^a, B^b, (a = b), (a, root), (b, root)]$, et est normalisé en $[X_2^b, Y_2^b, A^b, B^b, (b, root)]$.

Plusieurs optimisations spécifiques au calcul bioambiant pourraient également être incorporées. Par exemple, différents ambiants contenant les mêmes espèces pourraient être regroupés en une seule espèce, et les complexes pourraient être traités similairement aux complexes en π-calcul stochastique.

A.1.3 Exemple

Considérons les processus bioambiant définis comme suit :

$$\begin{array}{rclcrcl} E & = & \texttt{entrée}?s.EL(s) + \texttt{entrée}?p.EL(p) & & S & = & \texttt{entrée}!s.X + \tau_r \\ EL(x) & = & \texttt{sortie}?x.E & & P & = & \texttt{entrée}!p.X + \tau_r \\ & & & & X & = & \texttt{sortie}!s.S + \texttt{sortie}!p.P \end{array}$$

avec le processus initial $\boxed{E}^a \mid \boxed{S|P}^b$. Après initialisation de la machine, l'ensemble S de la population contient les espèces E^a, (a, racine), S^b, P^b, (b, racine) et les réactions liées sont :

1. $E^a + S^b + (a, \mathsf{racine}) + (b, \mathsf{racine}) \longrightarrow EL(s)^a + X^b + (a, \mathsf{racine}) + (b, a)$
2. $S^b \longrightarrow \mathbf{0}$
3. $E^a + P^b + (a, \mathsf{racine}) + (b, \mathsf{racine}) \longrightarrow EL(p)^a + X^b + (a, \mathsf{racine}) + (b, a)$
4. $P^b \longrightarrow \mathbf{0}$

Supposons que la réaction 1 est choisie. L'ambiant de localisation (b, racine) se déplace alors vers (b, a). Les espèces (b, racine), E^a, S^b sont mises à une population nulle et les nouvelles espèces $EL(s)^a, X^b$ sont créées. La propension des réactions 1, 2 et 3 est mise à zéro et les nouvelles réactions impliquant les espèces créées sont calculées :

5. $EL(s)^a + X^b + (a, \mathsf{racine}) + (b, a) \longrightarrow E^a + S^b + (a, \mathsf{racine}) + (b, \mathsf{racine})$.

A.1.4 Correction

Dans le cas du calcul bioambiant, la preuve de la proposition 6.2 est moins directe que pour le π-calcul (sous-section 6.5.2) car les définitions espèces et processus doivent traiter avec soin les localisations des ambiants. Bien que les résultats de cette sous-section s'appliquent pour le calcul bioambiant sans l'action `fusion`, ils peuvent être étendus facilement pour incorporer cette action.

Démonstration de la correction des espèces. Par induction sur les fonctions espèces et processus des définitions A.3 et A.4. En calculant espèces(P, a), l'identifiant a est ajouté à toutes les instances à la position courante dans la hiérarchie des compartiments afin d'indiquer leur localisation. Quand un ambiant $\boxed{P'}^{a'}$ est rencontré, l'espèce de localisation (a', a) est créée et a' est utilisé à la place

de a lors de la récursion dans le processus P'. Lors de la reconversion en processus, $\mathsf{processus}(\overline{I}, a)$ collecte toutes les instances à la localisation courante dans la hiérarchie des ambiants et utilise l'espèce de localisation pour reconstruire récursivement les compartiments enfants. □

Démonstration de la correction des réactions. Cette preuve est similaire à la précédente preuve de correction des réactions pour le π-calcul stochastique. La différence principale est que les définitions des fonctions source et cible doivent refléter le fait que les indices des réactions sur les bioambiants sont plus complexes que ceux utilisés en π-calcul stochastique. En particulier, nous ajoutons un argument additionnel $\widetilde{L}$ à source et cible pour l'ensemble de toutes les espèces de localisation présentes dans le système :

$$\mathsf{source}(w, \widetilde{L}) \triangleq \begin{cases} I & \text{si } w = i \wedge i \in I \\ I_1 \mid I_2 & \text{si } w = (i_1, i_2) \wedge i_1 \in I_1 \wedge i_2 \in I_2 \\ I_1 \mid I_2 \mid (a, b) & \text{si } w = (i_1, a, i_2) \wedge i_1 \in X(\widetilde{n})^a \\ & \quad \wedge i_2 \in Y(\widetilde{m})^b \wedge a \neq b \wedge (a, b) \in \widetilde{L} \\ I_1 \mid I_2 \mid (a, d) \mid (b, c) & \text{si } w = (i_1, a, i_2, b) \wedge i_1 \in X(\widetilde{n})^a \\ & \quad \wedge i_2 \in Y(\widetilde{m})^b \wedge \{(a, d), (b, c)\} \subseteq \widetilde{L} \\ & \quad \wedge a \neq b \wedge a \neq c \wedge (d = b \vee d = c) \end{cases}$$

$$\mathsf{cible}(w, \widetilde{L}) \triangleq P' \quad \text{si } \mathsf{source}(w, \widetilde{L}) \xrightarrow{\exp(\lambda), w} P'$$

Dans les deux derniers cas de $\mathsf{source}(w, \widetilde{L})$, nous autorisons le cas symétrique où $i_1 \in Y(\widetilde{m})^b$ et $i_2 \in X(\widetilde{n})^a$. Afin de reconstruire un processus depuis $\mathsf{cible}(w, \widetilde{L})$ nous avons également besoin de l'identifiant de l'ambiant à passer à la fonction espèces. Nous introduisons alors une nouvelle fonction $\mathsf{loc}(w, \widetilde{L})$ qui calcule la localisation correcte d'un indice donné w :

$$\mathsf{loc}(w, \widetilde{L}) \triangleq \begin{cases} a & \text{si } w = i \wedge i \in X(\widetilde{n})^a \\ a & \text{si } w = (i_1, i_2) \wedge i_1 \in X(\widetilde{n})^a \wedge i_2 \in Y(\widetilde{m})^a \\ b & \text{si } w = (i_1, a, i_2) \wedge i_1 \in X(\widetilde{n})^a \\ & \quad \wedge i_2 \in Y(\widetilde{m})^b \wedge a \neq b \wedge (a, b) \in \widetilde{L} \\ c & \text{si } w = (i_1, a, i_2, b) \wedge i_1 \in X(\widetilde{n})^a \\ & \quad \wedge i_2 \in Y(\widetilde{m})^b \wedge \{(a, d), (b, c)\} \subseteq \widetilde{L} \\ & \quad \wedge a \neq b \wedge a \neq c \wedge (d = b \vee d = c) \end{cases}$$

Comme précédemment, nous autorisons le cas symétrique où $i_1 \in Y(\widetilde{m})^b$ et $i_2 \in X(\widetilde{n})^a$ dans les deux cas finaux. Lors de la traduction de $\mathsf{cible}(w, \widetilde{L})$ en un multi-ensemble d'espèces, nous calculons $\mathsf{espèces}(\mathsf{cible}(w, \widetilde{L}), \mathsf{loc}(w, \widetilde{L}))$, en utilisant la fonction $\mathsf{espèces}(P, a)$ de la définition A.3, qui assure que les instances sont libellées par le bon identifiant de l'ambiant. Ces définitions suivent la définition A.4 et nous traitons les réactions fusionnées en sommant selon les indices w. Ainsi, nous pouvons utiliser une preuve similaire à celle de la correction des réactions en π-calcul stochastique présentée précédemment afin de montrer que $\mathsf{processus}(S) \xrightarrow{a} \mathsf{processus}(S')$ si et seulement si $a = \sum_{O \in \mathsf{ens\text{-}réactions}(S, S')} \mathsf{propension}(O, S)$ avec $a > 0$. □

A.2 Instanciation avec κ

Dans cette sous-section, nous décrivons une instanciation de la machine abstraite générique pour simuler κ.

A.2.1 Syntaxe et sémantique du calcul

κ (Danos, Feret, Fontana & Krivine, 2007; Danos et coll., 2008) est un langage à base de règles utilisé pour la modélisation d'interactions biologiques. κ propose un formalisme compact pour décrire les interactions diverses entre des *agents* présents dans une solution. Un agent est défini par son nom et un ensemble de *sites* qu'il peut utiliser pour interagir avec d'autres agents. Un site est soit libre, soit lié à un et un seul site d'un autre agent. Un site peut également recevoir un état interne, p. ex. *phosphorylé* ou *non-phosphorylé*. Les interactions entre les agents sont spécifiées par des *règles* décrivant les transformations à appliquer aux agents en présence d'un certain *contexte*. Le contexte d'application d'une règle consiste en un multi-ensemble d'agents spécifiés partiellement décrivant l'état de liaison et l'état interne des sites prenant part à la transformation ; les autres sites devant être ignorés de la spécification. Par exemple, la liaison entre deux agents $A(x, y), B(x, y)$ peut être spécifiée par la règle $A(x), B(y) \rightarrow A(x^1), B(y^1)$ si la liaison dépend seulement de la liberté des sites x et y de A et B, respectivement ; 1 dénote ici l'état de liaison des sites et est partagé par précisément deux sites appartenant à des instances différentes d'agents. La définition A.5 résume la syntaxe de κ. Nous dénotons par P une solution (un multi-ensemble) d'agents entièrement définis, et par G une solution d'agents partiellement définis. Les solutions sont considérées égales à un renommage des états de liaison près.

Étant donnée une solution P, l'application d'une règle (G, r, G') nécessite un *plongement* $\phi : G \mapsto P$ (définition A.7) des agents partiellement spécifiés vers leur spécification complète dans P. Si un tel plongement existe, la règle peut être appliquée (le contexte existe). Par abus de notation, la spécification des agents dans G' est étendue à l'aide du même plongement, noté $\phi(G')$. Comme toutes les instances d'agents présentes dans G' sont présentes dans G, $\phi(G')$ est défini pour tous les agents dans G', et les modifications apportées aux agents dans G' s'appliquent aux instances correspondantes. Le remplacement de ces instances d'agent dans P est noté $\phi(G')/P$ (définition A.6). Afin d'assurer le bien-fondé des solutions obtenues, les états de liaison non présents dans G sont transcrits par ϕ par des valeurs uniques. Le système de transition obtenu est établi par la définition A.8, et une sémantique en CMTC peut en être dérivée avec la règle suivante :

$$\frac{a = \left(\sum_{\{P^{r,\phi,(G,r,G')} \xrightarrow{} P'\}} r \right) > 0}{P \xrightarrow{a} P'} \tag{A.2}$$

Nous notons que la création et la suppression des agents n'est pas possible avec ces définitions ; toutefois, elles peuvent être étendues facilement pour gérer cette fonctionnalité.

A.2.2 Extraire les espèces et les réactions des processus

L'instanciation de notre machine abstraite générique avec κ est donnée dans la définition A.9. Une espèce I correspond à un agent complètement défini, donnant une relation simple entre le multi-ensemble des espèces $\overline{I}$ et un multi-ensemble d'agents complètement défini P (via les fonctions espèces et processus). Le multi-ensemble des réactions entre un agent I_1 et un ensemble d'agents

Définition A.5 (Syntaxe de κ)**.**

$P, G ::=$	$[a_1, .., a_M]$	Solution	$s ::=$	x_ι^λ		Site
$a ::=$	$A(\sigma)$	Agent	$\iota ::=$	$\epsilon \mid m \in \mathbb{V}$		État interne
$\sigma ::=$	$\{s_1, \ldots, s_M\}$	Interface	$\lambda ::=$	$\epsilon \mid i \in \mathbb{N}$		État de liaison

$E ::= \quad (G_1, r_1, G_1'), .., (G_N, r_N, G_N')$ Ensemble de règles

où r dénote le taux d'application d'une règle, A est le nom d'un agent et x est le nom d'un site.

Définition A.6 (Remplacement d'une solution κ)**.**

$$
\begin{array}{rcl@{\qquad}rcl}
x_{\iota_r}^{\lambda_r}/x_\iota^\lambda & \triangleq & x_{\iota_r}^{\lambda_r} & A(\sigma_r)/A(\sigma) & \triangleq & A(\sigma_r/\sigma) \\
x^{\lambda_r}/x_\iota^\lambda & \triangleq & x_\iota^{\lambda_r} & \emptyset/G & \triangleq & G \\
\emptyset/\sigma & \triangleq & \sigma & a_r, G'/a, G & \triangleq & a_r/a, G'/G \\
s_r, \sigma_r/s, \sigma & \triangleq & s_r/s, \sigma_r/\sigma & & &
\end{array}
$$

Définition A.7 (Plongement entre deux solutions κ)**.** $\phi \in \mathsf{plonge}(G, P) : G \mapsto P$ est un plongement d'une solution (partielle) G vers une solution P, si pour tout $a, b \in G$:

$$
\begin{array}{l}
\phi(a) = \phi(b) \Rightarrow a = b \\
Nom(a) = Nom(\phi(a)) \\
Site(a) \subseteq Site(\phi(a)) \\
x_\iota^\lambda \in Intf(a) \Rightarrow x_{\iota'}^{\lambda'} \in Intf(\phi(a)) \text{ avec } \lambda = \lambda' \wedge (\iota = \epsilon \vee \iota = \iota')
\end{array}
$$

où Nom, $Site$, et $Intf$ dénotent respectivement le nom A, les noms des sites $\{x_1, \ldots, x_M\}$ et l'interface σ de l'instance d'agent donnée.

Définition A.8 (Système de transition κ)**.** En considérant un ensemble global fixe de règles E :

$$
\frac{(G, r, G') \in E \quad \phi \in \mathsf{plonge}(G, P)}{P \xrightarrow{r, \phi, (G, r, G')} \phi(G')/P}
$$

Définition A.9 (Instanciation de la machine abstraite avec κ)**.**

$$
\begin{array}{rcl}
I & ::= & A(\sigma) \\
\mathsf{espèces}(P) & \triangleq & P \\
\mathsf{réactions}(I_1, \widetilde{I'}) & \triangleq & [(\bar{I}, r, \mathsf{espèces}(P')) \mid \exists \phi. \mathsf{processus}(\bar{I}) \xrightarrow{r, \phi, (G, r, G')} P' \\
 & & \qquad \wedge I_1 \in \bar{I} \wedge \widetilde{I} \subseteq (\{I_1\} \cup \widetilde{I'}) \wedge |\bar{I}| = |G|] \\
\mathsf{processus}(\bar{I}) & \triangleq & \bar{I}
\end{array}
$$

où E est un ensemble global fixe de règles. Un espèce I est un agent $A(\sigma)$. $|G|$ dénote le nombre d'agents dans G.

$\widetilde{I'}$ est calculé en utilisant le système de transitions de la définition A.8 : une réaction pour chaque règle (G, r, G') pour laquelle il existe au moins un plongement entre G et le multi-ensemble $\{I_1\} \cup \widetilde{I'}$ contenant I_1 et possédant le même nombre d'agents que dans G. La propension de la réaction obtenue $(\overline{I}, r, \overline{J})$ correspond au nombre de plongements possibles depuis G vers $\overline{I}$ résultant en $\overline{J}$ après application de la règle.

A.2.3 Exemple

Considérons les règles κ suivantes :

$$\begin{aligned} A(x, z_u), B(y) &\leftrightarrows A(x^1, z_p), B(y^1) \\ A(x^1), B(y^1), C(z_u) &\rightarrow A(x), B(y), C(z_p) \end{aligned}$$

avec la solution initiale $E = 100 \cdot A(x, z_u), 100 \cdot B(y), 100 \cdot C(z_u)$. Initialement, une réaction impliquant les espèces initiales est calculée :

1. $A(x, z_u), B(y) \rightarrow A(x^{\nu 1}, z_p), B(y^{\nu 1})$

Après application de cette réaction, un nouvel état de liaison (1) est créé donnant naissance à deux nouvelles espèces $A(x^1, z_p)$ et $B(y^1)$. Les populations de $A(x, z_u)$ et $B(y)$ sont décrémentées de 1. Deux réactions impliquant les nouvelles espèces sont alors extraites :

2. $A(x^1, z_p), B(y^1) \rightarrow A(x, z_u), B(y)$
3. $A(x^1, z_p), B(y^1), C(z_u) \rightarrow A(x, z_p), B(y), C(z_p)$

Si la même réaction est encore appliquée, un autre état de liaison (2) est créé entre les agents A et B, résultant en deux nouvelles espèces $A(x^2, z_p)$ et $B(y^2)$, et deux nouvelles réactions similaires aux précédentes seront calculées. Ceci montre une amélioration possible de l'instanciation de la machine pour gérer plus efficacement les états de liaison à la manière de la gestion des complexes dans l'instanciation avec le π-calcul stochastique (section 6.4, définition 6.11) : idéalement, nous aurions une seule espèce complexe : $A(x^1, z_p), B(y^1)$ au lieu de quatre espèces $A(x^1, z_p)$, $A(x^2, z_p)$, $B(y^1)$ et $B(y^2)$.

A.2.4 Correction

Nous présentons les preuves assurant pour la correction de l'instanciation avec κ.

Démonstration de la correction des espèces. Ceci est trivial car les fonctions espèces et processus (définition A.9) sont toutes deux la fonction identité. □

Démonstration de la correction des réactions. Nous savons que $\mathsf{processus}(S) \xrightarrow{a} \mathsf{processus}(S')$ si et seulement si $a = \sum_{\{\mathsf{processus}(S) \xrightarrow{r,\phi,(G,r,G')} \mathsf{processus}(S')\}} r$ avec $a > 0$. Étant donnée une règle (G, r, G'), chaque plongement différent $\phi \in \mathsf{plonge}(G, \mathsf{processus}(S))$ contribue à la propension de la réaction obtenue. Nous définissons le multi-ensemble des réactions F, où une réaction est présente pour chaque paire différente de plongement de règle :

$$\begin{aligned} F = [&(\mathsf{espèces}(\phi(G)), r, \mathsf{espèces}(\phi(G')/\phi(G))) \mid (G, r, G') \in E \\ &\wedge \phi \in \mathsf{plonge}(G, \mathsf{processus}(S)) \wedge \phi(G')/\mathsf{processus}(S) = \mathsf{processus}(S')] \end{aligned}$$

Pour tout $((\overline{I}, r, \overline{I'}), k) \in F$, nous savons que $(\overline{I}, r, \overline{I'}) \in \text{ens-réactions}(S, S')$. Étant donnée une règle (G, r, G') et un agent $a \in G$, nous écrivons $\{\phi_1, \ldots, \phi_L\} \subseteq \text{plonge}(G, \text{processus}(S))$ pour l'ensemble des plongements qui sont différents seulement dans l'association de l'agent a, et qui sont tels que $\phi_1(G) = \cdots = \phi_L(G) = \text{processus}(\overline{I})$. Nous obtenons $\text{espèces}(\phi_1(a)) = \cdots = \text{espèces}(\phi_L(a)) = I_a$ avec $I_a \in \overline{I}$; donc $L = S(I_a)$. Par induction, nous obtenons que $r \times k = \text{propension}((\overline{I}, r, \overline{I'}), S)$. Ainsi, il résulte que $\sum_{((\overline{I}, r, \overline{I'}), k) \in F} (r \times k) = \sum_{\{\text{processus}(S) \xrightarrow{r, \phi, (G, r, G')} \text{processus}(S')\}} r$, ce qui est équivalent à $a = \sum_{\{O \in \text{ens-réactions}(S, S')\}} \text{propension}(O, S)$ avec $a > 0$. □

Annexe **B**

Codes Sources PINT des Applications

B.1 La Segmentation chez les Métazoaires

```
process a 1
process c 1
process f 1
process fc 3 (* cooperative sort {f,c} *)

c 1 -> fc 0 1 @10.
c 1 -> fc 2 3 @10.
c 0 -> fc 1 0 @10.
c 0 -> fc 3 2 @10.
f 1 -> fc 0 2 @10.
f 1 -> fc 1 3 @10.
f 0 -> fc 2 0 @10.
f 0 -> fc 3 1 @10.

(* actions on c *)
f 1 -> c 0 1 @0.5~50 (* only if (f1,c0) *)
c 1 -> c 1 0 @0.5~50
f 0 -> c 1 0 @100.
(* actions on a *)
fc 2 -> a 0 1 @1.~50 (* only if (f1,c0) *)
c 1 -> a 1 0 @1.~50
(* actions on f *)
f 1 -> f 1 0 @0.034~100 (* auto-off *)

initial_state f 1, c 0, a 0
```

La commande `ph-exec -i metazoan.ph 40 exp` exécute le modèle enregistré dans le fichier `metazoan.ph` et crée pour chaque sorte un fichier décrivant son évolution dans le dossier `exp/`.

B.2 Le Récepteur des Cellules T - 40 composants

```
process Itk 1 process TCRbind 1 process Grb2Sos 1 process SEK 1 process
    PAGCsk 1 process IKK 1 process LCK 1 process PLCg_b 1 process Raf 1
    process NFAT 1 process CD45 1 process Fos 1 process TCRphos 1 process
    IkB 1 process LAT 1 process AP1 1 process Jun 1 process JNK 1 process
```

```
    IP3 1 process cCbl 1 process MEK 1 process Fyn 1 process NFkB 1
    process TCRlig 1 process CRE 1 process Ras 1 process ERK 1 process
    ZAP70 1 process Rsk 1 process Gads 1 process CREB 1 process Calcin 1
    process Ca 1 process RasGRP1 1 process PKCth 1 process Slp76 1
    process CD8 1 process Rlk 1 process PLCg_a 1 process DAG 1

GRN([
    Itk 1 -> + PLCg_a;
    TCRbind 1 -> + Fyn; TCRbind 1 -> + TCRphos; TCRbind 1 -> - PAGCsk;
    Grb2Sos 1 -> + Ras;
    SEK 1 -> + JNK;
    PAGCsk 1 -> - LCK;
    IKK 1 -> - IkB;
    LCK 1 -> + Fyn; LCK 1 -> + Rlk; LCK 1 -> + TCRphos; LCK 1 -> + ZAP70;
    PLCg_b 1 -> + PLCg_a;
    Raf 1 -> + MEK;
    CD45 1 -> + Fyn; CD45 1 -> + LCK; CD45 1 -> + CD45;
    Fos 1 -> + AP1;
    TCRphos 1 -> + ZAP70;
    IkB 1 -> - NFkB;
    LAT 1 -> + Gads; LAT 1 -> + Grb2Sos; LAT 1 -> + PLCg_b;
    Jun 1 -> + AP1;
    JNK 1 -> + Jun;
    IP3 1 -> + Ca;
    cCbl 1 -> - TCRbind; cCbl 1 -> - ZAP70;
    MEK 1 -> + ERK;
    Fyn 1 -> + PAGCsk; Fyn 1 -> + TCRphos;
    TCRlig 1 -> + TCRbind; TCRlig 1 -> + TCRlig;
    Ras 1 -> + Raf;
    ERK 1 -> + Fos; ERK 1 -> + Rsk;
    ZAP70 1 -> + PLCg_a; ZAP70 1 -> + Itk; ZAP70 1 -> + cCbl; ZAP70 1 ->
        + LAT;
    Rsk 1 -> + CREB;
    Gads 1 -> + Slp76;
    CREB 1 -> + CRE;
    Calcin 1 -> + NFAT;
    Ca 1 -> + Calcin;
    RasGRP1 1 -> + Ras;
    PKCth 1 -> + RasGRP1; PKCth 1 -> + IKK; PKCth 1 -> + SEK;
    Slp76 1 -> + PLCg_a; Slp76 1 -> + Itk;
    CD8 1 -> + LCK; CD8 1 -> + CD8;
    Rlk 1 -> + PLCg_a;
    PLCg_a 1 -> + DAG; PLCg_a 1 -> + IP3;
    DAG 1 -> + PKCth; DAG 1 -> + RasGRP1;
])

COOPERATIVITY([LCK;TCRbind] -> TCRphos 0 1, [[1;1]])
COOPERATIVITY([Fyn;LCK;TCRbind] -> TCRphos 1 0,
    [[0;0;0];[0;0;1];[0;1;0]])

COOPERATIVITY([CD45;LCK;TCRbind] -> Fyn 0 1, [[1;0;1];[1;1;0];[1;1;1]])
COOPERATIVITY([LCK;TCRbind] -> Fyn 1 0, [[0;0]])

COOPERATIVITY([ZAP70;Slp76;PLCg_b] in [[1;1;1]]
```

```
          and [Itk;Rlk] in [[0;1];[1;0];[1;1]],
            PLCg_a, 1, 0)

COOPERATIVITY([ZAP70;Slp76] -> Itk 0 1, [[1;1]])
COOPERATIVITY([Fos;Jun] -> AP1 0 1, [[1;1]])
COOPERATIVITY([Grb2Sos;RasGRP1] -> Ras 0 1, [[1;1]])
COOPERATIVITY([DAG;PKCth] -> RasGRP1 0 1, [[1;1]])

COOPERATIVITY([cCbl;TCRlig] -> TCRbind 0 1, [[0;1]])
COOPERATIVITY([PAGCsk;CD8;CD45] -> LCK 0 1, [[0;1;1]])
COOPERATIVITY([cCbl;TCRphos;LCK] -> ZAP70 0 1, [[0;1;1]])

COOPERATIVITY([TCRbind;Fyn] -> PAGCsk 0 1, [[0;0];[0;1]])
COOPERATIVITY([TCRbind;Fyn] -> PAGCsk 1 0, [[1;0];[1;1]])
```

B.3 Le Récepteur des Cellules T - 94 composants

```
process cd28 1 process itk 1 process akap79 1 process pkb 1 process jnk
    1 process bclxl 1 process gsk3 1 process nfat 1 process pten 1
    process creb 1 process ca 1 process dag 1 process cabin1 1 process
    fyn 1 process mekk1 1 process calcin 1 process calpr1 1 process
    sh3bp2 1 process ship1 1 process bad 1 process cre 1 process x 1
    process rac1p1 1 process rac1p2 1 process plcgb 1 process card11a 1
    process mlk3 1 process cyc1 1 process pdk1 1 process rasgrp 1 process
    fos 1 process pkcth 1 process abl 1 process tcrlig 1 process rlk 1
    process mek 1 process ikb 1 process p38 1 process grb2 1 process sre
    1 process camk2 1 process lckr 1 process camk4 1 process p70s 1
    process gads 1 process lckp2 1 process lckp1 1 process plcga 1
    process sos 1 process ccblp2 1 process slp76 1 process ccblp1 1
    process ikkg 1 process bcat 1 process shp2 1 process bcl10 1 process
    rsk 1 process vav1 1 process vav3 1 process ap1 1 process jun 1
    process gap 1 process ip3 1 process rac1r 1 process gadd45 1 process
    nfkb 1 process cam 1 process mkk4 1 process erk 1 process raf 1
    process cblb 1 process cd45 1 process tcrb 1 process ras 1 process
    tcrp 1 process p27k 1 process cdc42 1 process pip3 1 process gab2 1
    process pag 1 process card11 1 process ccblr 1 process dgk 1 process
    zap70 1 process lat 1 process shp1 1 process fkhr 1 process cd4 1
    process p21c 1 process csk 1 process pi3k 1 process hpk1 1 process
    ikkab 1 process malt1 1

COOPERATIVITY([slp76;pip3;zap70] -> itk 0 1, [[1;1;1]])
COOPERATIVITY([slp76;pip3;zap70] -> itk 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1]])
pdk1 1 -> pkb 0 1
pdk1 0 -> malt1 1 0
COOPERATIVITY([mekk1;mkk4] -> jnk 0 1, [[1;0];[0;1];[1;1]])
COOPERATIVITY([mekk1;mkk4] -> jnk 1 0, [[0;0]])
bad 1 -> bclxl 1 0
bad 0 -> malt1 0 1
pkb 1 -> gsk3 1 0
pkb 0 -> malt1 0 1
```

```
calcin 1 -> nfat 0 1
calcin 0 -> malt1 1 0
rsk 1 -> creb 0 1
rsk 0 -> malt1 1 0
ip3 1 -> ca 0 1
ip3 0 -> malt1 1 0
COOPERATIVITY([dgk;plcga] -> dag 0 1, [[0;1]])
COOPERATIVITY([dgk;plcga] -> dag 1 0, [[0;0];[1;0];[1;1]])
camk4 1 -> cabin1 1 0
camk4 0 -> malt1 0 1
COOPERATIVITY([cd45;lckp1;tcrb;lckr] -> fyn 0 1,
    [[1;1;0;0];[1;1;1;0];[1;1;0;1];[0;0;1;1];[1;0;1;1];[0;1;1;1];[1;1;1;1]])
COOPERATIVITY([cd45;lckp1;tcrb;lckr] -> fyn 1 0,
    [[0;0;0;0];[1;0;0;0];[0;1;0;0];[0;0;1;0];[1;0;1;0];[0;1;1;0];[0;0;0;1];
    [1;0;0;1];[0;1;0;1]])
COOPERATIVITY([rac1p2;hpk1;cdc42] -> mekk1 0 1,
    [[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1];[1;1;1]])
COOPERATIVITY([rac1p2;hpk1;cdc42] -> mekk1 1 0, [[0;0;0]])
COOPERATIVITY([cam;akap79;cabin1;calpr1] -> calcin 0 1, [[1;0;0;0]])
COOPERATIVITY([cam;akap79;cabin1;calpr1] -> calcin 1 0,
    [[0;0;0;0];[0;1;0;0];[1;1;0;0];[0;0;1;0];[1;0;1;0];[0;1;1;0];
    [1;1;1;0];[0;0;0;1];[1;0;0;1];[0;1;0;1];[1;1;0;1];[0;0;1;1];[1;0;1;1];
    [0;1;1;1];[1;1;1;1]])
COOPERATIVITY([lat;zap70] -> sh3bp2 0 1, [[1;1]])
COOPERATIVITY([lat;zap70] -> sh3bp2 1 0, [[0;0];[1;0];[0;1]])
pkb 1 -> bad 1 0
pkb 0 -> malt1 0 1
creb 1 -> cre 0 1
creb 0 -> malt1 1 0
cd28 1 -> x 0 1
cd28 0 -> malt1 1 0
COOPERATIVITY([vav1;rac1r] -> rac1p1 0 1, [[1;1]])
COOPERATIVITY([vav1;rac1r] -> rac1p1 1 0, [[0;0];[1;0];[0;1]])
COOPERATIVITY([vav3;rac1r] -> rac1p2 0 1, [[1;1]])
COOPERATIVITY([vav3;rac1r] -> rac1p2 1 0, [[0;0];[1;0];[0;1]])
lat 1 -> plcgb 0 1
lat 0 -> malt1 1 0
COOPERATIVITY([card11;bcl10;malt1] -> card11a 0 1, [[1;1;1]])
COOPERATIVITY([card11;bcl10;malt1] -> card11a 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1]])
COOPERATIVITY([rac1p1;hpk1] -> mlk3 0 1, [[1;0];[0;1];[1;1]])
COOPERATIVITY([rac1p1;hpk1] -> mlk3 1 0, [[0;0]])
gsk3 1 -> cyc1 1 0
gsk3 0 -> malt1 0 1
pip3 1 -> pdk1 0 1
pip3 0 -> malt1 1 0
dag 1 -> rasgrp 0 1
dag 0 -> malt1 1 0
erk 1 -> fos 0 1
erk 0 -> malt1 1 0
COOPERATIVITY([pdk1;vav1;dag] -> pkcth 0 1, [[1;1;1]])
COOPERATIVITY([pdk1;vav1;dag] -> pkcth 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1]])
COOPERATIVITY([lckp1;fyn] -> abl 0 1, [[1;0];[0;1];[1;1]])
```

```
COOPERATIVITY([lckp1;fyn] -> abl 1 0, [[0;0]])
lckp1 1 -> rlk 0 1
lckp1 0 -> malt1 1 0
raf 1 -> mek 0 1
raf 0 -> malt1 1 0
ikkab 1 -> ikb 1 0
ikkab 0 -> malt1 0 1
COOPERATIVITY([mekk1;gadd45;zap70] -> p38 0 1,
    [[1;0;0];[1;1;0];[0;0;1];[1;0;1];[1;1;1]])
COOPERATIVITY([mekk1;gadd45;zap70] -> p38 1 0, [[0;0;0];[0;1;0];[0;1;1]])
lat 1 -> grb2 0 1
lat 0 -> malt1 1 0
COOPERATIVITY([cdc42;rac1p2] -> sre 0 1, [[1;0];[0;1];[1;1]])
COOPERATIVITY([cdc42;rac1p2] -> sre 1 0, [[0;0]])
cam 1 -> camk2 0 1
cam 0 -> malt1 1 0
cam 1 -> camk4 0 1
cam 0 -> malt1 1 0
pdk1 1 -> p70s 0 1
pdk1 0 -> malt1 1 0
lat 1 -> gads 0 1
lat 0 -> malt1 1 0
COOPERATIVITY([tcrb;lckr] -> lckp2 0 1, [[1;1]])
COOPERATIVITY([tcrb;lckr] -> lckp2 1 0, [[0;0];[1;0];[0;1]])

COOPERATIVITY(
            [cd45;cd4;lckr] in [[1;1;1]]
        and [shp1;csk] in [[0;0]],
        lckp1, 1, 0)

COOPERATIVITY(
        [plcgb;itk;rlk] in [[1;0;1];[1;1;0];[1;1;1]]
    and [vav1;zap70;ccblp2;slp76] in [[1;1;0;1]],
    plcga, 1, 0)

grb2 1 -> sos 0 1
grb2 0 -> malt1 1 0
COOPERATIVITY([fyn;ccblr] -> ccblp2 0 1, [[1;1]])
COOPERATIVITY([fyn;ccblr] -> ccblp2 1 0, [[0;0];[1;0];[0;1]])
COOPERATIVITY([gab2;zap70;gads] -> slp76 0 1, [[0;1;1]])
COOPERATIVITY([gab2;zap70;gads] -> slp76 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[1;1;1]])
COOPERATIVITY([zap70;ccblr] -> ccblp1 0 1, [[1;1]])
COOPERATIVITY([zap70;ccblr] -> ccblp1 1 0, [[0;0];[1;0];[0;1]])
COOPERATIVITY([card11a;pkcth] -> ikkg 0 1, [[1;1]])
COOPERATIVITY([card11a;pkcth] -> ikkg 1 0, [[0;0];[1;0];[0;1]])
gsk3 1 -> bcat 1 0
gsk3 0 -> malt1 0 1
gab2 1 -> shp2 0 1
gab2 0 -> malt1 1 0
erk 1 -> rsk 0 1
erk 0 -> malt1 1 0
COOPERATIVITY([x;zap70;sh3bp2] -> vav1 0 1,
    [[1;0;0];[1;1;0];[1;0;1];[0;1;1];[1;1;1]])
```

```
COOPERATIVITY([x;zap70;sh3bp2] -> vav1 1 0, [[0;0;0];[0;1;0];[0;0;1]])
sh3bp2 1 -> vav3 0 1
sh3bp2 0 -> malt1 1 0
COOPERATIVITY([jun;fos] -> ap1 0 1, [[1;1]])
COOPERATIVITY([jun;fos] -> ap1 1 0, [[0;0];[1;0];[0;1]])
jnk 1 -> jun 0 1
jnk 0 -> malt1 1 0
plcga 1 -> ip3 0 1
plcga 0 -> malt1 1 0
ikb 1 -> nfkb 1 0
ikb 0 -> malt1 0 1
ca 1 -> cam 0 1
ca 0 -> malt1 1 0
COOPERATIVITY([mekk1;mlk3] -> mkk4 0 1, [[1;0];[0;1];[1;1]])
COOPERATIVITY([mekk1;mlk3] -> mkk4 1 0, [[0;0]])
mek 1 -> erk 0 1
mek 0 -> malt1 1 0
ras 1 -> raf 0 1
ras 0 -> malt1 1 0
cd28 1 -> cblb 1 0
cd28 0 -> malt1 0 1
COOPERATIVITY([tcrlig;ccblp1] -> tcrb 0 1, [[1;0]])
COOPERATIVITY([tcrlig;ccblp1] -> tcrb 1 0, [[0;0];[0;1];[1;1]])
COOPERATIVITY([rasgrp;sos;gap] -> ras 0 1, [[1;1;0]])
COOPERATIVITY([rasgrp;sos;gap] -> ras 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[0;0;1];[1;0;1];[0;1;1];[1;1;1]])
COOPERATIVITY([lckp1;tcrb;fyn] -> tcrp 0 1, [[1;1;0];[0;1;1];[1;1;1]])
COOPERATIVITY([lckp1;tcrb;fyn] -> tcrp 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[0;0;1];[1;0;1]])
pkb 1 -> p27k 1 0
pkb 0 -> malt1 0 1
COOPERATIVITY([pten;ship1;pi3k] -> pip3 0 1, [[0;0;1]])
COOPERATIVITY([pten;ship1;pi3k] -> pip3 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[1;0;1];[0;1;1];[1;1;1]])
COOPERATIVITY([lat;grb2;zap70;gads] -> gab2 0 1,
    [[1;1;1;0];[1;0;1;1];[1;1;1;1]])
COOPERATIVITY([lat;grb2;zap70;gads] -> gab2 1 0,
    [[0;0;0;0];[1;0;0;0];[0;1;0;0];[1;1;0;0];[0;0;1;0];[1;0;1;0];
    [0;1;1;0];[0;0;0;1];[1;0;0;1];[0;1;0;1];[1;1;0;1];[0;0;1;1];[0;1;1;1]])
COOPERATIVITY([tcrb;fyn] -> pag 0 1, [[0;0];[0;1];[1;1]])
COOPERATIVITY([tcrb;fyn] -> pag 1 0, [[1;0]])
tcrb 1 -> dgk 0 1
tcrb 0 -> malt1 1 0
COOPERATIVITY([tcrp;abl;ccblp1] -> zap70 0 1, [[1;1;0]])
COOPERATIVITY([tcrp;abl;ccblp1] -> zap70 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[0;0;1];[1;0;1];[0;1;1];[1;1;1]])
zap70 1 -> lat 0 1
zap70 0 -> malt1 1 0
COOPERATIVITY([lckp1;erk] -> shp1 0 1, [[1;0]])
COOPERATIVITY([lckp1;erk] -> shp1 1 0, [[0;0];[0;1];[1;1]])
pkb 1 -> fkhr 1 0
pkb 0 -> malt1 0 1
pkb 1 -> p21c 1 0
pkb 0 -> malt1 0 1
```

```
pag 1 -> csk 0 1
pag 0 -> malt1 1 0
COOPERATIVITY([cblb;lckp2;x] -> pi3k 0 1, [[0;1;0];[0;0;1];[0;1;1]])
COOPERATIVITY([cblb;lckp2;x] -> pi3k 1 0,
    [[0;0;0];[1;0;0];[1;1;0];[1;0;1];[1;1;1]])
lat 1 -> hpk1 0 1
lat 0 -> malt1 1 0
COOPERATIVITY([ikkg;camk2] -> ikkab 0 1, [[1;1]])
COOPERATIVITY([ikkg;camk2] -> ikkab 1 0, [[0;0];[1;0];[0;1]])

initial_state lckr 1, bcl10 1, rac1r 1, cd45 1, card11 1, ccblr 1, malt1
    1
```

B.4 Le Récepteur du Facteur de Croissance Épidermique - 20 composants

```
process AKT1 1
process CDK2 1 process CDK4 1 process CDK6 1
process CycD1 1 process CycE1 1
process EGF 1
process ERalpha 1
process ERBB1 1 process ERBB1_2 1 process ERBB1_3 1
process ERBB2 1 process ERBB2_3 1 process ERBB3 1
process IGF1R 1
process MEK1 1
process MYC 1
process p21 1 process p27 1
process pRB 1

GRN([
    ERBB2_3 1 -> + AKT1; ERBB2_3 1 -> + MEK1; ERBB2_3 1 -> - IGF1R;
    ERBB2 1 -> + ERBB2_3; ERBB2 1 -> + ERBB1_2; ERBB3 1 -> + ERBB2_3;
    ERBB3 1 -> + ERBB1_3;
    CycE1 1 -> + CDK2;
    MEK1 1 -> + CycD1; MEK1 1 -> + ERalpha; MEK1 1 -> + MYC;
    CDK4 1 -> + pRB; CDK4 1 -> - p21; CDK4 1 -> - p27;
    ERalpha 1 -> + CycD1; ERalpha 1 -> + IGF1R; ERalpha 1 -> + p21;
       ERalpha 1 -> + MYC; ERalpha 1 -> + p27;
    MYC 1 -> + CycE1; MYC 1 -> - p21; MYC 1 -> + CycD1; MYC 1 -> - p27;
    CDK6 1 -> + pRB;
    ERBB1 1 -> + ERBB1_2; ERBB1 1 -> + ERBB1_3; ERBB1 1 -> + AKT1; ERBB1
       1 -> + MEK1;
    IGF1R 1 -> + AKT1; IGF1R 1 -> + MEK1;
    ERBB1_3 1 -> + AKT1; ERBB1_3 1 -> + MEK1;
    p27 1 -> - CDK2; p27 1 -> - CDK4;
    CDK2 1 -> - p27; CDK2 1 -> + pRB;
    p21 1 -> - CDK2; p21 1 -> - CDK4;
    CycD1 1 -> + CDK4; CycD1 1 -> + CDK6;
    EGF 1 -> + ERBB1; EGF 1 -> + ERBB2; EGF 1 -> + ERBB3;
```

```
    AKT1 1 -> + CycD1; AKT1 1 -> + MYC; AKT1 1 -> - p27; AKT1 1 -> +
        ERalpha; AKT1 1 -> + IGF1R; AKT1 1 -> - p21;
    ERBB1_2 1 -> + AKT1; ERBB1_2 1 -> + MEK1;
])

COOPERATIVITY([ERBB1;ERBB2] -> ERBB1_2 0 1, [[1;1]])
COOPERATIVITY([ERBB1;ERBB3] -> ERBB1_3 0 1, [[1;1]])
COOPERATIVITY([ERBB2;ERBB3] -> ERBB2_3 0 1, [[1;1]])

COOPERATIVITY([ERBB2_3;AKT1] -> IGF1R 0 1, [[0;1]])
COOPERATIVITY([ERBB2_3;ERalpha] -> IGF1R 0 1, [[0;1]])
COOPERATIVITY([AKT1;ERalpha] -> IGF1R 1 0, [[0;0]])

COOPERATIVITY([AKT1;MEK1] -> ERalpha 1 0, [[0;0]])
COOPERATIVITY([AKT1;MEK1;ERalpha] -> MYC 1 0, [[0;0;0]])
COOPERATIVITY([ERBB1;ERBB1_2;ERBB1_3;ERBB2_3;IGF1R] -> AKT1 1 0,
    [[0;0;0;0;0]])
COOPERATIVITY([ERBB1;ERBB1_2;ERBB1_3;ERBB2_3;IGF1R] -> MEK1 1 0,
    [[0;0;0;0;0]])
COOPERATIVITY([CycE1;p21;p27] -> CDK2 0 1, [[1;0;0]])
COOPERATIVITY([CycD1;p21;p27] -> CDK4 0 1, [[1;0;0]])
COOPERATIVITY([ERalpha;MYC;AKT1;MEK1] -> CycD1 0 1,
    [[1;1;1;0];[1;1;0;1]])
COOPERATIVITY([AKT1;MEK1] -> CycD1 1 0, [[0;0]])
COOPERATIVITY([ERalpha;AKT1;MYC;CDK4] -> p21 0 1, [[1;0;0;0]])
COOPERATIVITY([ERalpha;CDK4;CDK2;AKT1;MYC] -> p27 0 1, [[1;0;0;0;0]])
COOPERATIVITY([CDK2;CDK4;CDK6] -> pRB 0 1, [[0;1;1];[1;1;1]])
RM({CDK2 0 -> pRB 1 0})
RM({EGF 1 -> EGF 1 0}) (* prevent self-degradation (input) *)

initial_state EGF 1
```

B.5 Le Récepteur du Facteur de Croissance Épidermique - 104 composants

```
process sos1_eps8_e3b1 1 process pp2b 1 process pkc 1 process pp2a 1
    process jnk 1 process gsk3 1 process pi3kr 1 process plcg 1 process
    p90rskerk12d 1 process sos1r 1 process creb 1 process erbb34 1
    process dag 1 process limk1 1 process mekk1 1 process csrc 1 process
    mekk4 1 process bir 1 process cjun 1 process ship2 1 process cfos 1
    process bad 1 process nucerk12 1 process mek12 1 process
    pro_apoptotic 1 process erbb23 1 process erbb24 1 process mlk3 1
    process pdk1 1 process btc 1 process rasgap 1 process erbb14 1
    process eps8r 1 process erbb11 1 process erbb12 1 process erbb13 1
    process endocyt_degrad 1 process shc 1 process mtorr 1 process p38 1
    process grb2 1 process stat5 1 process stat3 1 process stat1 1
    process rntre 1 process akt 1 process elk1 1 process cmyc 1 process
    nrg2a 1 process nrg2b 1 process erbb2 1 process erbb3 1 process erbb1
    1 process erbb4 1 process ca 1 process rac_cdc42 1 process mtor_ric 1
    process pak1 1 process pten 1 process p90rsk 1 process mkp 1 process
    erk12 1 process rheb 1 process p70s6_1 1 process p70s6_2 1 process
```

```
    rab5a 1 process sos1 1 process vav2 1 process rin1 1 process hbegf 1
    process nck 1 process ship2d 1 process tgfa 1 process pi3k 1 process
    tsc1_tsc2 1 process mk2 1 process mkk3 1 process raf1 1 process mkk4
    1 process mkk7 1 process mkk6 1 process mtor_rap 1 process ar 1
    process ras 1 process nrg3 1 process nrg4 1 process aktd 1 process
    pip3 1 process ccbl 1 process gab1 1 process nrg1a 1 process nrg1b 1
    process ptend 1 process ap1 1 process actin_reorg 1 process shp1 1
    process shp2 1 process ip3 1 process shp1d 1 process epr 1 process
    hsp27 1 process pi34p2 1 process erbb44 1 process egf 1

COOPERATIVITY([eps8r;pip3;pi3kr;sos1r] -> sos1_eps8_e3b1 0 1,
    [[1;1;1;1]])
COOPERATIVITY([eps8r;pip3;pi3kr;sos1r] -> sos1_eps8_e3b1 1 0,
    [[0;0;0;0];[1;0;0;0];[0;1;0;0];[1;1;0;0];[0;0;1;0];[1;0;1;0];[0;1;1;0];
    [1;1;1;0];[0;0;0;1];[1;0;0;1];[0;1;0;1];[1;1;0;1];[0;0;1;1];[1;0;1;1];
    [0;1;1;1]])
COOPERATIVITY([pdk1;ca;dag] -> pkc 0 1, [[1;1;1]])
COOPERATIVITY([pdk1;ca;dag] -> pkc 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1]])
COOPERATIVITY([mkk4;mkk7] -> jnk 0 1, [[1;1]])
COOPERATIVITY([mkk4;mkk7] -> jnk 1 0, [[0;0];[1;0];[0;1]])
COOPERATIVITY([p90rsk;akt] -> gsk3 0 1, [[0;0]])
COOPERATIVITY([p90rsk;akt] -> gsk3 1 0, [[1;0];[0;1];[1;1]])
erbb11 1 -> plcg 0 1
erbb11 0 -> egf 1 0
COOPERATIVITY([erk12;p90rsk] -> p90rskerk12d 0 1, [[1;1]])
COOPERATIVITY([erk12;p90rsk] -> p90rskerk12d 1 0, [[0;0];[1;0];[0;1]])
COOPERATIVITY([p90rsk;mk2] -> creb 0 1, [[1;0];[0;1];[1;1]])
COOPERATIVITY([p90rsk;mk2] -> creb 1 0, [[0;0]])

COOPERATIVITY(
        [erbb2;erbb3;erbb4] in [[0;1;1]]
    and not ([nrg1a;nrg1b] in [[0;0]] and [nrg2a;nrg2b] in [[0;0]]),
    erbb34, 1, 0)

plcg 1 -> dag 0 1
plcg 0 -> egf 1 0
pak1 1 -> limk1 0 1
pak1 0 -> egf 1 0
rac_cdc42 1 -> mekk1 0 1
rac_cdc42 0 -> egf 1 0
rac_cdc42 1 -> mekk4 0 1
rac_cdc42 0 -> egf 1 0
jnk 1 -> cjun 0 1
jnk 0 -> egf 1 0
COOPERATIVITY([jnk;erk12;p90rsk;pp2a] -> cfos 0 1,
    [[1;0;0;0];[1;1;0;0];[1;0;1;0];[0;1;1;0];[1;1;1;0]])
COOPERATIVITY([jnk;erk12;p90rsk;pp2a] -> cfos 1 0,
    [[0;0;0;0];[0;1;0;0];[0;0;1;0];[0;0;0;1];[1;0;0;1];[0;1;0;1];
    [1;1;0;1];[0;0;1;1];[1;0;1;1];[0;1;1;1];[1;1;1;1]])
COOPERATIVITY([pak1;akt] -> bad 0 1, [[0;0]])
COOPERATIVITY([pak1;akt] -> bad 1 0, [[1;0];[0;1];[1;1]])
COOPERATIVITY([erk12;mkp] -> nucerk12 0 1, [[1;0]])
COOPERATIVITY([erk12;mkp] -> nucerk12 1 0, [[0;0];[0;1];[1;1]])
```

```
COOPERATIVITY([mekk1;raf1] -> mek12 0 1, [[1;0];[0;1];[1;1]])
COOPERATIVITY([mekk1;raf1] -> mek12 1 0, [[0;0]])
bad 1 -> pro_apoptotic 0 1
bad 0 -> egf 1 0

COOPERATIVITY(
        [erbb2;erbb3] in [[1;1]]
    and not ([epr;nrg1a;nrg1b] in [[0;0;0]] and [btc;nrg2b;bir] in
        [[0;0;0]]),
    erbb23, 1, 0)

COOPERATIVITY(
        [erbb2;erbb4] in [[1;1]]
    and not ([epr;nrg1a;nrg1b] in [[0;0;0]]
                and [btc;nrg2b;bir] in [[0;0;0]]
                and [tgfa;nrg3;nrg4] in [[0;0;0]]
                and [hbegf;nrg2a;egf] in [[0;0;0]]),
        erbb24, 1, 0)

rac_cdc42 1 -> mlk3 0 1
rac_cdc42 0 -> egf 1 0
COOPERATIVITY([shp2;gab1] -> rasgap 0 1, [[0;1]])
COOPERATIVITY([shp2;gab1] -> rasgap 1 0, [[0;0];[1;0];[1;1]])

COOPERATIVITY(
        [erbb1;erbb4;erbb2;shp1d] in [[1;1;0;0]]
    and not ([epr;nrg1a;nrg1b] in [[0;0;0]]
                and [nrg2a;nrg2b;nrg4] in [[0;0;0]]
                and [tgfa;egf] in [[0;0]]),
    erbb14, 1, 0)

COOPERATIVITY(
        [erbb1;shp1d] in [[1;0]]
    and not ([epr;hbegf;ar] in [[0;0;0]]
                and [btc;bir] in [[0;0]]
                and [tgfa;egf] in [[0;0]]),
    erbb11, 1, 0)

COOPERATIVITY(
        [erbb1;erbb2;shp1d] in [[1;1;0]]
    and not ([epr;hbegf] in [[0;0]]
                and [btc;bir] in [[0;0]]
                and [tgfa;egf] in [[0;0]]),
    erbb12, 1, 0)

COOPERATIVITY(
        [erbb1;erbb3;shp1d] in [[1;1;0]]
    and [ar;erbb2] in [[1;1];[1;0];[0;0]]
    and not ([epr;nrg1a;nrg1b] in [[0;0;0]]
                and [tgfa;egf;btc;nrg2a] in [[0;0;0;0]]),
    erbb13, 1, 0)

COOPERATIVITY([rab5a;ccbl] -> endocyt_degrad 0 1, [[1;1]])
COOPERATIVITY([rab5a;ccbl] -> endocyt_degrad 1 0, [[0;0];[1;0];[0;1]])
```

```
COOPERATIVITY(
    not ([erbb14;erbb11;erbb12;erbb13] in [[0;0;0;0]]
        and [erbb23;erbb44;erbb24;erbb34] in [[0;0;0;0]]),
    shc, 0, 1)

COOPERATIVITY([mkk3;mkk4;mkk6] -> p38 0 1,
    [[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1];[1;1;1]])
COOPERATIVITY([mkk3;mkk4;mkk6] -> p38 1 0, [[0;0;0]])

COOPERATIVITY(
    not ([erbb14;erbb11;erbb12;erbb13] in [[0;0;0;0]]
        and [erbb23;erbb44;erbb24;erbb34] in [[0;0;0;0]]
        and [shc] in [[0]]),
    grb2, 0, 1)

COOPERATIVITY([csrc;erbb24;erbb11] -> stat5 0 1,
    [[1;1;0];[1;0;1];[1;1;1]])
COOPERATIVITY([csrc;erbb24;erbb11] -> stat5 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[0;0;1];[0;1;1]])
COOPERATIVITY([csrc;erbb11] -> stat3 0 1, [[1;1]])
COOPERATIVITY([csrc;erbb11] -> stat3 1 0, [[0;0];[1;0];[0;1]])
COOPERATIVITY([csrc;erbb11] -> stat1 0 1, [[1;1]])
COOPERATIVITY([csrc;erbb11] -> stat1 1 0, [[0;0];[1;0];[0;1]])
COOPERATIVITY([eps8r;erbb11] -> rntre 0 1, [[1;1]])
COOPERATIVITY([eps8r;erbb11] -> rntre 1 0, [[0;0];[1;0];[0;1]])

COOPERATIVITY(
        [mtor_ric;pp2a;pdk1] in [[1;0;1]]
    and not [pip3;pi34p2] in [[0;0]],
    akt, 1, 0)

COOPERATIVITY([pp2b;nucerk12] -> elk1 0 1, [[0;1]])
COOPERATIVITY([pp2b;nucerk12] -> elk1 1 0, [[0;0];[1;0];[1;1]])
COOPERATIVITY([nucerk12;gsk3] -> cmyc 0 1, [[1;0]])
COOPERATIVITY([nucerk12;gsk3] -> cmyc 1 0, [[0;0];[0;1];[1;1]])
ip3 1 -> ca 0 1
ip3 0 -> egf 1 0
COOPERATIVITY([sos1_eps8_e3b1;vav2] -> rac_cdc42 0 1,
    [[1;0];[0;1];[1;1]])
COOPERATIVITY([sos1_eps8_e3b1;vav2] -> rac_cdc42 1 0, [[0;0]])
mtorr 1 -> mtor_ric 0 1
mtorr 0 -> egf 1 0
COOPERATIVITY([nck;rac_cdc42;grb2] -> pak1 0 1,
    [[1;1;0];[0;1;1];[1;1;1]])
COOPERATIVITY([nck;rac_cdc42;grb2] -> pak1 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[0;0;1];[1;0;1]])
COOPERATIVITY([erk12;pdk1] -> p90rsk 0 1, [[1;1]])
COOPERATIVITY([erk12;pdk1] -> p90rsk 1 0, [[0;0];[1;0];[0;1]])
mek12 1 -> erk12 0 1
mek12 0 -> egf 1 0
tsc1_tsc2 1 -> rheb 1 0
tsc1_tsc2 0 -> egf 0 1
COOPERATIVITY([jnk;erk12] -> p70s6_1 0 1, [[1;0];[0;1];[1;1]])
```

```
COOPERATIVITY([jnk;erk12] -> p70s6_1 1 0, [[0;0]])
COOPERATIVITY([pdk1;mtor_rap;p70s6_1] -> p70s6_2 0 1, [[1;1;1]])
COOPERATIVITY([pdk1;mtor_rap;p70s6_1] -> p70s6_2 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1]])
COOPERATIVITY([rntre;rin1] -> rab5a 0 1, [[0;1]])
COOPERATIVITY([rntre;rin1] -> rab5a 1 0, [[0;0];[1;0];[1;1]])
COOPERATIVITY([sos1r;p90rskerk12d;grb2] -> sos1 0 1, [[1;0;1]])
COOPERATIVITY([sos1r;p90rskerk12d;grb2] -> sos1 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[0;0;1];[0;1;1];[1;1;1]])
COOPERATIVITY([pi34p2;erbb11;pip3] -> vav2 0 1,
    [[1;1;0];[0;1;1];[1;1;1]])
COOPERATIVITY([pi34p2;erbb11;pip3] -> vav2 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[0;0;1];[1;0;1]])
ras 1 -> rin1 0 1
ras 0 -> egf 1 0
COOPERATIVITY([erbb14;erbb44;erbb11] -> nck 0 1,
    [[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1];[1;1;1]])
COOPERATIVITY([erbb14;erbb44;erbb11] -> nck 1 0, [[0;0;0]])
ship2 1 -> ship2d 0 1
ship2 0 -> egf 1 0

COOPERATIVITY(
        [pi3kr] in [[1]]
    and not ([erbb13;erbb23;erbb34] in [[0;0;0]]
                and [pi3kr;gab1;ras] in [[0;0;0]]),
    pi3k, 1, 0)

akt 1 -> tsc1_tsc2 1 0
akt 0 -> egf 0 1
p38 1 -> mk2 0 1
p38 0 -> egf 1 0
mlk3 1 -> mkk3 0 1
mlk3 0 -> egf 1 0
COOPERATIVITY([csrc;aktd;pak1;ras] -> raf1 0 1,
    [[1;0;0;1];[0;0;1;1];[1;0;1;1]])
COOPERATIVITY([csrc;aktd;pak1;ras] -> raf1 1 0,
    [[0;0;0;0];[1;0;0;0];[0;1;0;0];[1;1;0;0];[0;0;1;0];[1;0;1;0];
    [0;1;1;0];[1;1;1;0];[0;0;0;1];[0;1;0;1];[1;1;0;1];[0;1;1;1];[1;1;1;1]])
COOPERATIVITY([mekk1;mlk3;mekk4] -> mkk4 0 1,
    [[1;0;0];[0;1;0];[1;1;0];[0;0;1];[1;0;1];[0;1;1];[1;1;1]])
COOPERATIVITY([mekk1;mlk3;mekk4] -> mkk4 1 0, [[0;0;0]])
mekk1 1 -> mkk7 0 1
mekk1 0 -> egf 1 0
mlk3 1 -> mkk6 0 1
mlk3 0 -> egf 1 0
COOPERATIVITY([rheb;mtorr] -> mtor_rap 0 1, [[1;1]])
COOPERATIVITY([rheb;mtorr] -> mtor_rap 1 0, [[0;0];[1;0];[0;1]])
COOPERATIVITY([rasgap;sos1] -> ras 0 1, [[0;1]])
COOPERATIVITY([rasgap;sos1] -> ras 1 0, [[0;0];[1;0];[1;1]])
akt 1 -> aktd 0 1
akt 0 -> egf 1 0
COOPERATIVITY([ship2d;ptend;pi3k] -> pip3 0 1, [[0;0;1]])
COOPERATIVITY([ship2d;ptend;pi3k] -> pip3 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[1;0;1];[0;1;1];[1;1;1]])
```

```
erbb11 1 -> ccbl 0 1
erbb11 0 -> egf 1 0
COOPERATIVITY([grb2;erbb11] -> gab1 0 1, [[1;0];[0;1];[1;1]])
COOPERATIVITY([grb2;erbb11] -> gab1 1 0, [[0;0]])
pten 1 -> ptend 0 1
pten 0 -> egf 1 0
COOPERATIVITY([cfos;cjun] -> ap1 0 1, [[1;1]])
COOPERATIVITY([cfos;cjun] -> ap1 1 0, [[0;0];[1;0];[0;1]])
limk1 1 -> actin_reorg 0 1
limk1 0 -> egf 1 0
erbb11 1 -> shp1 0 1
erbb11 0 -> egf 1 0
gab1 1 -> shp2 0 1
gab1 0 -> egf 1 0
plcg 1 -> ip3 0 1
plcg 0 -> egf 1 0
shp1 1 -> shp1d 0 1
shp1 0 -> egf 1 0
mk2 1 -> hsp27 0 1
mk2 0 -> egf 1 0
COOPERATIVITY([ship2d;ptend;pi3k] -> pi34p2 0 1, [[1;0;1]])
COOPERATIVITY([ship2d;ptend;pi3k] -> pi34p2 1 0,
    [[0;0;0];[1;0;0];[0;1;0];[1;1;0];[0;0;1];[0;1;1];[1;1;1]])

COOPERATIVITY(
        [erbb4] in [[1]]
    and not ([btc;nrg2b;bir] in [[0;0;0]]
                and [nrg4;nrg1a;nrg1b;nrg3] in [[0;0;0;0]]),
    erbb44, 1, 0)
```

Bibliographie

Abadi, M. & Gordon, A. D. (1999), 'A calculus for cryptographic protocols : The spi calculus', *Information and Computation* **148**, 36–47.

Ahmad, J., Bernot, G., Comet, J. P., Lime, D. & Roux, O. (2006), 'Hybrid modelling and dynamical analysis of gene regulatory networks with delays', *Complexus* **3**(4), 231–251.

Ahmad, J., Bourdon, J., Eveillard, D., Fromentin, J., Roux, O. & Sinoquet, C. (2009), 'Temporal constraints of a gene regulatory network : Refining a qualitative simulation', *Biosystems* **98**(3), 149 – 159.

Alur, R., Belta, C., Kumar, V., Mintz, M., Pappas, G. J., Rubin, H. & Schug, J. (2002), 'Modeling and analyzing biomolecular networks', *Computing in Science and Engineering* **4**(1), 20–31.

Alur, R., Courcoubetis, C. & Dill, D. L. (1990), Model-checking for real-time systems, *in* 'LICS', IEEE Computer Society, pp. 414–425.

Alur, R. & Dill, D. (1992), The theory of timed automata, *in* J. de Bakker, C. Huizing, W. de Roever & G. Rozenberg, eds, 'Real-Time : Theory in Practice', Vol. 600 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 45–73.

Alur, R. & Henzinger, T. A. (1999), 'Reactive modules', *Formal Methods in System Design* **15**, 7–48.

Amari, S. & Misra, R. (1997), 'Closed-form expressions for distribution of sum of exponential random variables', *Reliability, IEEE Transactions on* **46**(4), 519–522.

Aracena, J. (2008), 'Maximum number of fixed points in regulatory boolean networks', *Bulletin of Mathematical Biology* **70**(5), 1398–1409.

Aracena, J., Demongeot, J. & Goles, E. (2004), 'Positive and negative circuits in discrete neural networks', *IEEE Transactions of Neural Networks* **15**, 77–83.

Aracena, J., Goles, E., Moreira, A. & Salinas, L. (2009), 'On the robustness of update schedules in boolean networks', *Biosystems* **97**(1), 1 – 8.

Baier, C. & Hermanns, H. (2003), 'Model-checking algorithms for continuous-time markov chains', *IEEE Transactions on Software Engineering* **29**(6), 524–541.

Banks, R. (2009), Modelling Genetic Regulatory Networks : Petri Net Techniques and Tools, School of Computing Science, thèse de doctorat, University of Newcastle upon Tyne.

Barrett, C. L., Hunt III, H. B., Marathe, M. V., Ravi, S. S., Rosenkrantz, D. J., Stearns, R. E. & Tosic, P. T. (2001), Gardens of eden and fixed points in sequential dynamical systems, *in* R. Cori, J. Mazoyer, M. Morvan & R. Mosseri, eds, 'DM-CCG', Vol. AA of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pp. 95–110.

Bernot, G., Cassez, F., Comet, J.-P., Delaplace, F., Müller, C. & Roux, O. (2007), 'Semantics of biological regulatory networks', *Electronic Notes in Theoretical Computer Science* **180**(3), 3 – 14.

Bernot, G., Comet, J.-P. & Khalis, Z. (2008), Gene regulatory networks with multiplexes, *in* 'European Simulation and Modelling Conference Proceedings', pp. 423–432.

Best, D. & Roberts, D. (1975), 'Algorithm as 91 : The percentage points of the chi-squared distribution', *Applied Statistics* **24**(3), 385–390.

Blossey, R., Cardelli, L. & Phillips, A. (2008), 'Compositionality, stochasticity and cooperativity in dynamic models of gene regulation', *HFSP Journal* **2**(1), 17–28.

Bobbio, A. & Horváth, A. (2001), 'Petri nets with discrete phase type timing : A bridge between stochastic and functional analysis', *Electr. Notes Theor. Comput. Sci.* **52**(3).

Bouissou, O. (2008), Analyse statique par interprétation abstraite de systèmes hybrides, thèse de doctorat, École Polytechnique.

Brand, D. & Zafiropulo, P. (1983), 'On communicating finite-state machines', *Journal of the ACM* **30**, 323–342.

Bratsun, D., Volfson, D., Tsimring, L. S. & Hasty, J. (2005), 'Delay-induced stochastic oscillations in gene regulation', *Proceedings of the National Academy of Sciences of the United States of America* **102**(41), 14593–14598.

Bryans, J., Bowman, H. & Derrick, J. (2003), 'Model checking stochastic automata', *ACM Trans. Comput. Logic* **4**(4), 452–492.

Bryant, R. E. (1986), 'Graph-based algorithms for boolean function manipulation', *IEEE Transactions on Computers* **35**, 677–691.

Cardelli, L. (2004), Brane calculi, *in* 'Computational Methods in Systems Biology', pp. 257–278.

Cardelli, L., Caron, E., Gardner, P., Kahramanogullari, O. & Phillips, A. (2009), 'A process model of actin polymerisation', *Electronic Notes in Theoretical Computer Science* **229**, 127–144.

Chaouiya, C., Remy, E. & Thieffry, D. (2008), 'Petri net modelling of biological regulatory networks', *Journal of Discrete Algorithms* **6**(2), 165 – 177. Selected papers from CompBioNets 2004, Algorithms and Computational Methods for Biochemical and Evolutionary Networks.

Chatain, T. & Jard, C. (2006), Complete finite prefixes of symbolic unfoldings of safe time petri nets, *in* S. Donatelli & P. Thiagarajan, eds, 'Petri Nets and Other Models of Concurrency - ICATPN 2006', Vol. 4024 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 125–145.

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. & Tacchella, A. (2002), Nusmv 2 : An opensource tool for symbolic model checking, *in* E. Brinksma & K. Larsen, eds, 'Computer Aided Verification', Vol. 2404 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 241–268.

Ciocchetta, F. & Hillston, J. (2009), 'Bio-pepa : A framework for the modelling and analysis of biological systems', *Theoretical Computer Science* **410**(33-34), 3065 – 3084.

Clarke, E. M. & Emerson, E. A. (1981), Design and synthesis of synchronization skeletons using branching-time temporal logic, *in* 'Logic of Programs', Springer-Verlag, London, UK, pp. 52–71.

Cousot, P. & Cousot, R. (1977), Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints, *in* 'Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages', POPL '77, ACM, New York, NY, USA, pp. 238–252.

Cousot, P. & Cousot, R. (1992), 'Abstract interpretation frameworks', *Journal of Logic and Computation* **2**(4), 511–547.

Couvreur, J.-M. & Thierry-Mieg, Y. (2005), *Formal Techniques for Networked and Distributed Systems - FORTE 2005*, chapter Hierarchical Decision Diagrams to Exploit Model Structure, pp. 443–457.

Danos, V., Feret, J., Fontana, W., Harmer, R. & Krivine, J. (2007), *CONCUR 2007 – Concurrency Theory*, chapter Rule-Based Modelling of Cellular Signalling, pp. 17–41.

Danos, V., Feret, J., Fontana, W. & Krivine, J. (2007), Scalable simulation of cellular signaling networks, invited paper, *in* Z. Shao, ed., 'Proceedings of the Fifth Asian Symposium on Programming Systems, APLAS'2007, Singapore', Vol. 4807 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, Singapore, pp. 139–157.

Danos, V., Feret, J., Fontana, W. & Krivine, J. (2008), Abstract interpretation of cellular signalling networks, *in* F. Logozzo, D. Peled & L. Zuck, eds, 'Verification, Model Checking, and Abstract Interpretation', Vol. 4905 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 83–97.

de Jong, H. (2002), 'Modeling and simulation of genetic regulatory systems : A literature review', *Journal of Computational Biology* **9**, 67–103.

Dematté, L., Priami, C. & Romanel, A. (2008*a*), 'Modelling and simulation of biological processes in blenx', *SIGMETRICS Performance Evaluation Review* **35**(4), 32–39.

Dematte, L., Priami, C. & Romanel, A. (2008*b*), 'The Beta Workbench : a computational tool to study the dynamics of biological systems', *Brief Bioinform* p. bbn023.

DiDonato, A. R. & Morris, Jr., A. H. (1986), 'Computation of the incomplete gamma function ratios and their inverse', *ACM Trans. Math. Softw.* **12**(4), 377–393.

Donatelli, S., Haddad, S. & Moreaux, P. (1998), Structured characterization of the markov chain of phase-type SPN, *in* R. Puigjaner, N. Savino & B. Serra, eds, 'Computer Performance Evaluation', Vol. 1469 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 243–254.

Esparza, J. & Heljanko, K. (2008), *Unfoldings : A Partial-Order Approach to Model Checking (Monographs in Theoretical Computer Science. An EATCS Series)*, 1 edn, Springer Publishing Company, Incorporated.

Evans, M., Hastings, N. & Peacock, B. (2000), *Statistical Distributions*, 3rd edn, Wiley-Interscience.

Ewald, R., Himmelspach, J., Jeschke, M., Leye, S. & Uhrmacher, A. M. (2010), 'Flexible experimentation in the modeling and simulation framework JAMES II – implications for computational systems biology.', *Brief Bioinform* .

Fages, F. & Soliman, S. (2008*a*), 'Abstract interpretation and types for systems biology', *Theoretical Computer Science* **403**(1), 52 – 70.

Fages, F. & Soliman, S. (2008*b*), Formal cell biology in biocham, *in* 'Proceedings of the Formal methods for the design of computer, communication, and software systems 8th international conference on Formal methods for computational systems biology', SFM'08, Springer-Verlag, Berlin, Heidelberg, pp. 54–80.

Favaro, S. & Walker, S. (2008), 'On the distribution of sums of independent exponential random variables via wilks' integral representation', *Acta Applicandae Mathematicae* .

Feret, J. (2005), Analysis of mobile systems by abstract interpretation, thèse de doctorat, École Normale Supérieure.

Feret, J., Henzinger, T. A., Koeppl, H. & Petrov, T. (2010), Lumpability abstractions of rule-based systems, *in* G. Ciobanu & M. Koutny, eds, 'Proceedings Fourth Workshop on Membrane Computing and Biologically Inspired Process Calculi 2010', Vol. 40 of *EPTCS*, pp. 142–161.

François, P., Hakim, V. & Siggia, E. D. (2007), 'Deriving structure from evolution : metazoan segmentation', *Mol Syst Biol* **3**.

Fromentin, J., Eveillard, D. & Roux, O. (2010), 'Hybrid modeling of biological networks : mixing temporal and qualitative biological properties', *BMC Systems Biology* **4**(1), 79.

Gay, S., Soliman, S. & Fages, F. (2010), 'A graphical method for reducing and relating models in systems biology', *Bioinformatics* **26**, i575–i581.

Gibson, M. A. & Bruck, J. (2000), 'Efficient exact stochastic simulation of chemical systems with many species and many channels', *The Journal of Physical Chemistry A* **104**(9), 1876–1889.

Gillespie, D. T. (1977), 'Exact stochastic simulation of coupled chemical reactions', *The Journal of Physical Chemistry* **81**(25), 2340–2361.

Gillespie, D. T. (2001), 'Approximate accelerated stochastic simulation of chemically reacting systems', *J. Chem. Phys.* **115**, 1716–1733.

Gonzalez, A. G., Naldi, A., Sánchez, L., Thieffry, D. & Chaouiya, C. (2006), 'Ginsim : A software suite for the qualitative modelling, simulation and analysis of regulatory networks', *Biosystems* **84**(2), 91 – 100. Dynamical Modeling of Biological Regulatory Networks.

Guziolowski, C., Bourde, A., Moreews, F. & Siegel, A. (2009), 'Bioquali cytoscape plugin : analysing the global consistency of regulatory networks', *BMC Genomics* **10**(1), 244.

Hamez, A., Thierry-Mieg, Y. & Kordon, F. (2009), 'Building efficient model checkers using hierarchical set decision diagrams and automatic saturation', *Fundam. Inf.* **94**(3-4), 413–437.

Harel, D. (1987), 'Statecharts : A Visual Formalism for Complex Systems', *Sci. Comput. Prog.* **8**, 231–274.

Heiner, M., Gilbert, D. & Donaldson, R. (2008), *Formal Methods for Computational Systems Biology*, chapter Petri Nets for Systems and Synthetic Biology, pp. 215–264.

Heiner, M., Rohr, C., Schwarick, M. & Streif, S. (2010), A comparative study of stochastic analysis techniques, *in* 'Proceedings of the 8th International Conference on Computational Methods in Systems Biology', CMSB '10, ACM, New York, NY, USA, pp. 96–106.

Hinton, A., Kwiatkowska, M., Norman, G. & Parker, D. (2006), PRISM : A tool for automatic verification of probabilistic systems, *in* '12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems', Vol. 3920 of *LNCS*, Springer.

I3S (SMBioNet), 'Selection of Models of Biological Networks'. `http://www.i3s.unice.fr/~richard/smbionet`.

Katoen, J.-P., Klink, D., Leucker, M. & Wolf, V. (2008), *CONCUR 2008 - Concurrency Theory*, chapter Abstraction for Stochastic Systems by Erlang's Method of Stages, pp. 279–294.

Kauffman, S. A. (1969), 'Metabolic stability and epigenesis in randomly connected nets', *Journal of Theoretical Biology* **22**, 437–467.

Kauffman, S. A. (1993), *Origins of Order Self-Organization and Selection in Evolution*, Oxford University Press.

Klamt, S., Saez-Rodriguez, J. & Gilles, E. (2007), 'Structural and functional analysis of cellular networks with cellnetanalyzer', *BMC Systems Biology* **1**(1), 2.

Klamt, S., Saez-Rodriguez, J., Lindquist, J., Simeoni, L. & Gilles, E. (2006), 'A methodology for the structural and functional analysis of signaling and regulatory networks', *BMC Bioinformatics* **7**(1), 56.

Kohavi, R. & Provost, F. (1998), 'Glossary of terms', *Editorial for the Special Issue on Applications of Machine Learning and the Knowledge Discovery Process* **30**.

Kuttler, C. & Niehren, J. (2006), 'Gene regulation in the pi calculus : Simulating cooperativity at the lambda switch', *Transactions on Computational Systems Biology* **4230**(VII), 24–55.

Kwiatkowska, M., Norman, G. & Parker, D. (2006), *Computer Aided Verification*, chapter Symmetry Reduction for Probabilistic Model Checking, pp. 234–248.

Leloup, J.-C. & Goldbeter, A. (2003), 'Toward a detailed computational model for the mammalian circadian clock', *Proceedings of the National Academy of Sciences* **100**(12), 7051–7056.

LIP6/Move (libDDD), 'the libDDD environment'. `http://ddd.lip6.fr`.

López, G., Hermanns, H. & Katoen, J.-P. (2001), *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*, chapter Beyond Memoryless Distributions : Model Checking Semi-Markov Chains, pp. 57–70.

Mangan, S. & Alon, U. (2003), 'Structure and function of the feed-forward loop network motif.', *PNAS* **100**(21), 11980–11985.

Maurin, M., Magnin, M. & Roux, O. (2009), Modeling of genetic regulatory network in stochastic π-calculus, *in* S. Rajasekaran, ed., 'Bioinformatics and Computational Biology', Vol. 5462 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 282–294.

Mendoza, L., Thieffry, D. & Alvarez-Buylla, E. R. (1999), 'Genetic control of flower morphogenesis in arabidopsis thaliana : a logical analysis.', *Bioinformatics* **15**, 593–606.

Merlin, P. M. (1974), A study of the recoverability of computing systems., thèse de doctorat.

Mi, J. & Naranjo, A. (2003), 'Inferences about the scale parameter of the gamma distribution based on data mixed from censoring and grouping', *Statistics & Probability Letters* **62**(3).

Milner, R. (1989), *Communication and Concurrency*, Prentice Hall International Series in Computer Science, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Milner, R. (1999), *Communicating and mobile systems : the π-calculus*, Cambridge University Press, New York, NY, USA.

Mura, I., Prandi, D., Priami, C. & Romanel, A. (2009), 'Exploiting non-Markovian Bio-Processes', *Electr. Notes Theor. Comput. Sci.* **253**(3), 83–98.

Nadarajah, S. (2008), 'A review of results on sums of random variables', *Acta Applicandae Mathematicae* **103**(2), 131–140.

Naldi, A., Remy, E., Thieffry, D. & Chaouiya, C. (2009), A reduction of logical regulatory graphs preserving essential dynamical properties, *in* P. Degano & R. Gorrieri, eds, 'Computational Methods in Systems Biology', Vol. 5688 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 266–280.

Naldi, A., Thieffry, D. & Chaouiya, C. (2007), Decision diagrams for the representation and analysis of logical models of genetic networks, *in* 'Proceedings of the 2007 international conference on Computational methods in systems biology', CMSB'07, Springer-Verlag, Berlin, Heidelberg, pp. 233–247.

Nikolski, M. (2000), Binary Decision Diagrams and Applications for Reliability Analysis, thèse de doctorat, Université de Bordeaux I.

Norman, G., Palamidessi, C., Parker, D. & Wu, P. (2009), 'Model checking probabilistic and stochastic extensions of the π-calculus', *IEEE Trans. on Software Engineering* **35**(2), 209–223.

Palamidessi, C. (2003), 'Comparing the expressive power of the synchronous and asynchronous *pi*-calculi', *Mathematical Structures in Computer Science* **13**(05), 685–719.

Paulevé, L. (PINT), 'PINT - Process Hitting related tools'. `http://process.hitting.free.fr`.

Paulevé, L., Magnin, M. & Roux, O. (2010), 'Tuning Temporal Features within the Stochastic π-Calculus', *IEEE Transactions on Software Engineering* **99**(PrePrints).

Paulevé, L., Magnin, M. & Roux, O. (2011*a*), 'Abstract Interpretation of Dynamics of Biological Regulatory Networks', *Electronic Notes in Theoretical Computer Science* **272**, 43–56. Proceedings of The First International Workshop on Static Analysis and Systems Biology (SASB 2010).

Paulevé, L., Magnin, M. & Roux, O. (2011*b*), Refining dynamics of gene regulatory networks in a stochastic π-calculus framework, *in* C. Priami, R.-J. Back, I. Petre & E. de Vink, eds, 'Transactions on Computational Systems Biology XIII', Vol. 6575 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 171–191.

Paulevé, L. & Richard, A. (2010), 'Topological Fixed Points in Boolean Networks', *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics* **348**(15-16), 825 – 828.

Paulevé, L., Youssef, S., Lakin, M. R. & Phillips, A. (2010), A generic abstract machine for stochastic process calculi, *in* 'CMSB '10 : Proceedings of the 8th International Conference on Computational Methods in Systems Biology', ACM, New York, NY, USA, pp. 43–54.

Pedersen, M. & Phillips, A. (2009), 'Towards programming languages for genetic engineering of living cells', *Journal of the Royal Society Interface* **6**(S4), 437–450.

Pedersen, M. & Plotkin, G. (2008), A language for biochemical systems, *in* 'Computational Methods in Systems Biology', Vol. 5307 of *LNCS*, Springer, pp. 63–82.

Phillips, A. (2009), 'An abstract machine for the stochastic bioambient calculus', *Electronic Notes in Theoretical Computer Science* **227**, 143–159.

Phillips, A. (SPiM), *SPiM - Stochastic Pi Machine*. `http://research.microsoft.com/en-us/projects/spim`.

Phillips, A. & Cardelli, L. (2007), Efficient, correct simulation of biological processes in the stochastic pi-calculus, *in* 'Computational Methods in Systems Biology', Vol. 4695 of *LNCS*, Springer.

Phillips, A. & Cardelli, L. (2009), 'A programming language for composable DNA circuits', *Journal of the Royal Society Interface* **6**(S4), 419–436.

Phillips, A., Cardelli, L. & Castagna, G. (2006), 'A graphical representation for biological processes in the stochastic pi-calculus', *Trans. in Computational Systems Biology* **4230**, 123–152.

Phillips, A., Lakin, M. & Paulevé, L. (2010), Stochastic simulation of process calculi for biology, *in* G. Ciobanu & M. Koutny, eds, 'Proceedings Fourth Workshop on Membrane Computing and Biologically Inspired Process Calculi 2010', Vol. 40 of *EPTCS*, pp. 1 – 5. invited paper.

Popova-Zeugmann, L., Heiner, M. & Koch, I. (2005), 'Time petri nets for modelling and analysis of biochemical networks', *Fundamenta Informaticae* **67**(1), 149–162.

Prandi, D., Priami, C. & Romanel, A. (2008), Simulation of Non-Markovian Processes in BlenX, Rapport Technique TR-11-2008, The Microsoft Research - University of Trento Centre for Computational and Systems Biology.

Priami, C. (1995), 'Stochastic π-Calculus', *The Computer Journal* **38**(7), 578–589.

Priami, C. (1996), Stochastic π-calculus with general distributions, *in* 'Proc. of the 4th Workshop on Process Algebras and Performance Modelling, CLUT', pp. 41–57.

Priami, C., Regev, A., Shapiro, E. & Silverman, W. (2001), 'Application of a stochastic name-passing calculus to representation and simulation of molecular processes', *Inf. Process. Lett.* **80**(1), 25–31.

R Development Core Team (2009), *R : A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing.
`http://www.R-project.org`

Radulescu, O., Gorban, A., Zinovyev, A. & Lilienbaum, A. (2008), 'Robust simplifications of multiscale biochemical networks', *BMC Systems Biology* **2**(1), 86.

Regev, A., Panina, E. M., Silverman, W., Cardelli, L. & Shapiro, E. Y. (2004), 'Bioambients : an abstraction for biological compartments', *Theor. Comput. Sci.* **325**(1), 141–167.

Regev, A., Silverman, W. & Shapiro, E. (2001), 'Representation and simulation of biochemical processes using the pi-calculus process algebra.', *Pac Symp Biocomput* pp. 459–70.

Remy, É., Ruet, P. & Thieffry, D. (2008), 'Graphic requirements for multistability and attractive cycles in a boolean dynamical framework', *Advances in Applied Mathematics* **41**(3), 335 – 350.

Richard, A. (2006), Modèle formel pour les réseaux de régulation génétique et influence des circuits de rétroaction., thèse de doctorat, Université d'Évry Val d'Essone.

Richard, A. (2009), 'Positive circuits and maximal number of fixed points in discrete dynamical systems', *Discrete Applied Mathematics* **157**(15), 3281 – 3288.

Richard, A. (2010), 'Negative circuits and sustained oscillations in asynchronous automata networks', *Advances in Applied Mathematics* **44**(4), 378 – 392.

Richard, A. & Comet, J.-P. (2007), 'Necessary conditions for multistationarity in discrete dynamical systems', *Discrete Applied Mathematics* **155**(18), 2403 – 2413.

Richard, A., Comet, J.-P. & Bernot, G. (2006), *Modern Formal Methods and Applications*, chapter Formal Methods for Modeling Biological Regulatory Networks, pp. 83–122.

Rizk, A., Batt, G., Fages, F. & Soliman, S. (2008), On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology, *in* M. Heiner & A. Uhrmacher, eds, 'Computational Methods in Systems Biology', Vol. 5307 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 251–268.

Saez-Rodriguez, J., Simeoni, L., Lindquist, J. A., Hemenway, R., Bommhardt, U., Arndt, B., Haus, U.-U., Weismantel, R., Gilles, E. D., Klamt, S. & Schraven, B. (2007), 'A logical model provides insights into t cell receptor signaling', *PLoS Comput Biol* **3**(8), e163.

Sahin, O., Frohlich, H., Lobke, C., Korf, U., Burmester, S., Majety, M., Mattern, J., Schupp, I., Chaouiya, C., Thieffry, D., Poustka, A., Wiemann, S., Beissbarth, T. & Arlt, D. (2009), 'Modeling ERBB receptor-regulated G1/S transition to find novel targets for de novo trastuzumab resistance', *BMC Systems Biology* **3**(1).

Samaga, R., Saez-Rodriguez, J., Alexopoulos, L. G., Sorger, P. K. & Klamt, S. (2009), 'The logic of egfr/erbb signaling : Theoretical properties and analysis of high-throughput data', *PLoS Comput Biol* **5**(8), e1000438.

Schnoebelen, P. (2002), The complexity of temporal logic model checking., *in* 'Advances in Modal Logic'02', pp. 393–436.

Siebert, H. & Bockmayr, A. (2006), Incorporating time delays into the logical analysis of gene regulatory networks, *in* C. Priami, ed., 'Computational Methods in Systems Biology', Vol. 4210 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 169–183.

Thomas, R. (1973), 'Boolean formalization of genetic control circuits', *Journal of Theoretical Biology* **42**(3), 563 – 585.

Thomas, R. (1981), On the relation between the logical structure of systems and their ability to generate multiple steady states or sustained oscillations, *in* 'Series in Synergetics', 9, Springer, pp. 180–193.

Thomas, R. & d'Ari, R. (1990), *Biological Feedback*, CRC Press.

Thomas, R. & Kaufman, M. (2001), 'Multistationarity, the basis of cell differentiation and memory. ii. logical analysis of regulatory networks in terms of feedback circuits', *Chaos : An Interdisciplinary Journal of Nonlinear Science* **11**(1), 180–195.

Tian, T. & Burrage, K. (2004), 'Binomial leap methods for simulating stochastic chemical kinetics', *J. Chem. Phys.* **121**, 10356–10364.

Tyson, J. & Othmer, H. (1978), 'The Dynamics of Feedback Control Circuits in Biochemical Pathways', *Prog. Theor. Biol.* **5**, 1–62.

Vogler, W., Semenov, A. & Yakovlev, A. (1998), Unfolding and finite prefix for nets with read arcs, *in* D. Sangiorgi & R. de Simone, eds, 'CONCUR'98 Concurrency Theory', Vol. 1466 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 501–516.

Wang, D. Y., Cardelli, L., Phillips, A., Piterman, N. & Fisher, J. (2009), 'Computational modeling of the egfr network elucidates control mechanisms regulating signal dynamics', *BMC Systems Biology* **3**(118).

Wilkinson, D. J. (2006), *Stochastic Modelling for Systems Biology (Mathematical and Computational Biology)*, Chapman & Hall/CRC.

Wuensche, A. (1998), 'Genomic regulation modeled as a network with basins of attraction.', *Pac Symp Biocomput* pp. 89–102.

Zaigraev, A. & Podraza-Karakulska, A. (2008), 'On estimation of the shape parameter of the gamma distribution', *Statistics & Probability Letters* **78**(3), 286 – 295.