Thèse de Doctorat

Maxime FOLSCHETTE

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'École centrale de Nantes
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information et mathématiques

Discipline : Informatique et applications
Unité de recherche : Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN)

Soutenue le 8 octobre 2014

Modélisation algébrique de la dynamique multi-échelles des réseaux de régulation biologique

JURY

Présidente :	**M^me^ Mireille RÉGNIER**, Directrice de recherche Inria, École polytechnique (LIX) & Université Paris-Sud 11 (LRI)
Rapporteurs :	**M. Jean-Paul COMET**, Professeur des universités, Université de Nice – Sophia Antipolis (I3S) **M^me^ Anne SIEGEL**, Directrice de recherche CNRS, Inria Rennes (IRISA)
Examinateur :	**M. Denis THIEFFRY**, Professeur des universités, École normale supérieure (Institut de biologie)
Directeur de thèse :	**M. Olivier ROUX**, Professeur des universités, École centrale de Nantes (IRCCyN)
Co-encadrant de thèse :	**M. Morgan MAGNIN**, Maître de conférences, École centrale de Nantes (IRCCyN)

Table des matières

Chapitre 1

Introduction

1.1 Contexte & Motivations

Le principal défi posé par l'étude des systèmes dynamiques réels, qu'ils soient biologiques ou non, repose dans la modélisation qui en est faite. Un modèle permet d'abstraire les comportements du système pour s'intéresser uniquement à ceux qui présentent un intérêt à l'étude, tout en permettant leur analyse à l'aide d'outils préalablement développés.

Un modèle doit donc dans l'idéal :

- être cohérent avec la réalité du système qu'il représente,
- reproduire les comportements présentant un intérêt et abstraire ceux qui surchargent inutilement le modèle,
- faciliter la lecture pour le modélisateur,
- permettre l'analyse par des outils appropriés,
- permettre la traduction depuis ou vers d'autres formalismes et favoriser l'ouverture à d'autres méthodes d'analyse.

Nous nous intéressons dans cette thèse aux propriétés dynamiques d'un modèle. Elles se distinguent des propriétés statiques qui permettent de caractériser le modèle en fonction de sa taille, des liens entre les éléments qui le composent, ou de toute autre propriété portant sur sa structure ou ses attributs, bien que celles-ci apportent parfois aussi des résultats très généraux concernant la dynamique. À l'inverse, les propriétés dynamiques portent sur l'évolution et les comportements possibles d'un modèle, par exemple :

- Étant donné un certain état des entrées du système, le modèle est-il capable d'en reproduire ou d'en prédire les sorties ?
- Y retrouve-t-on des comportements qui s'apparentent à des oscillations ou des états stables ?
- Les comportements recherchés sont-ils accessibles depuis tous les états ? Si non, depuis lesquels ?
- Peut-on modifier le modèle pour voir apparaître un comportement donné, et comment ?

Répondre à ces questions nécessite une analyse détaillée de la dynamique.

Ainsi, l'utilisation d'un modèle pose un double défi : sa **conception** et son **analyse**.

Il est nécessaire de proposer des outils permettant une modélisation cohérente et juste. C'est pourquoi nous proposons dans cette thèse plusieurs formalismes nouveaux permettant des représentations efficaces et complémentaires des systèmes dynamiques étudiés.

L'une des pistes d'enrichissement consiste en l'introduction de contraintes dynamiques afin de filtrer les comportements non désirés, par exemple sous la forme de relations de prévalence entre les différentes évolutions possibles d'un modèle. Cela ajoute de plus la possibilité de contraindre les comportements du modèle en fonction de paramètres temporels issus par exemple de données relatives aux durées de réaction ou de sensibilisation, à des mesures de retards entre phénomènes biologiques, etc. Enfin, l'ajout d'outils de synchronisation à des formalismes purement asynchrones peut s'avérer nécessaire pour représenter certains comportements simultanés.

Un autre problème récurrent est de parvenir à faire le lien entre plusieurs formalismes. En effet, des formalismes différents peuvent permettre des approches variées, et l'application d'outils adaptés à des problèmes précis. Il s'avère donc nécessaire de pouvoir faire le lien entre différentes versions d'un formalisme, ou encore de créer des ponts vers d'autres formalismes répandus dans le cadre des systèmes dynamiques. Aussi, cette thèse offre une part importante à l'**étude des relations entre différentes représentations** complémentaires des modèles dynamiques étudiés, ainsi qu'avec les modèles classiques pour ce type de représentations. L'intérêt d'une telle comparaison est de pouvoir mettre en valeur et d'exploiter les avantages de chacune de ces approches.

Cependant, vérifier de telles propriétés dynamiques présente un coût en termes de temps d'exécution et de taille mémoire, lié à la taille du modèle considéré. Et si, dans les meilleurs cas, ce coût varie de façon polynomiale en fonction de la taille du modèle, la plupart des méthodes formelles permettant de vérifier des propriétés dynamiques font malheureusement face à une explosion combinatoire qui empêche l'analyse de modèles de grande taille. C'est pourquoi des méthodes alternatives peuvent être explorées, comme la vérification par interprétation abstraite, qui consiste à approcher de façon plus ou moins fine la dynamique pour diminuer la complexité et simplifier les calculs. Comme nous le verrons dans la suite de ce manuscrit, il s'agit d'une des facettes de notre travail, et nous soutenons que celle-ci constitue une contribution importante pour l'étude des systèmes considérés.

La section 1.2 propose une rapide vue d'ensemble des types de modèles permettant la représentation et l'étude des systèmes d'interactions, afin de mieux situer le travail proposé dans cette thèse. Les principaux résultats de notre travail sont ensuite présentés à la section 1.3, dans un cadre de collaborations précisées à la section 1.4. La section 1.5 présente la façon dont ce manuscrit est organisé, et la section 1.6 introduit les différentes notations qui y sont utilisées.

1.2 Les réseaux de régulation biologique

L'étude de la machine cellulaire nécessite de s'intéresser aux éléments interagissant qui la composent : gènes, protéines, ARN messager, métabolites, etc. À ce niveau, il est déjà possible d'abstraire dans une certaine mesure une partie des composants. C'est ainsi qu'un gène, la protéine qu'il code et l'ARN messager correspondant sont souvent tous trois modélisés par un unique élément, car la concentration de la protéine et de l'ARN dépendent directement du niveau d'activation du gène. Si cette simplification paraît poussée face à la réalité biologique qu'elle représente, elle permet néanmoins d'abstraire de façon cohérente le mécanisme de création d'une protéine pour s'intéresser aux relations entre la présence d'une protéine et son influence sur la production d'une autre. Comme ce cas de figure est particulièrement répandu dans l'étude des réseaux de régulation, nous assimilerons souvent

dans la suite une protéine avec son gène codant.

Cette simplification permet de mettre en évidence les phénomènes d'interaction entre les différents éléments entrant en jeu. En effet, la présence d'un composant (protéine, catalyseur...) en quantité suffisante peut déclencher l'activation (une hausse de l'activité) ou l'inhibition (une baisse de l'activité) d'un ou plusieurs autres éléments, y compris l'élément déclencheur lui-même. C'est par ce mécanisme que se crée une cascade de réactions entre gènes : en effet, l'un d'eux, selon son degré d'activité, permettra la production d'une concentration plus ou moins importante de la protéine qu'il code ; celle-ci aura alors éventuellement un rôle activateur ou inhibiteur sur un certain nombre d'autres gènes, et ainsi de suite. L'ensemble de ces régulations peut être représenté par un graphe des interactions, où les composants sont modélisés par des nœuds et leurs interactions mutuelles par des arcs.

Afin de représenter la dynamique du modèle, une valeur abstraite est associée à chaque élément afin de modéliser son état courant (niveau d'activité pour un gène, concentration dans le milieu pour une protéine, etc.). Dans le cadre de formalismes utilisant des équations différentielles il s'agit d'une valeur continue, ce qui permet par exemple de lier entre elles les concentrations des différentes protéines et leurs dérivées par rapport au temps, qui représentent alors leurs vitesses d'évolution (Tyson & Othmer, 1978). Cependant, le manque de données expérimentales précises et fiables peut limiter de telles approches. De plus, la résolution analytique ou numérique des équations différentielles est parfois très complexe, quand elle n'est pas impossible.

Une autre approche consiste donc à symboliser l'activité d'un élément par une **valeur discrète** au sein d'un ensemble fini. Elle se justifie par le fait que la courbe d'évolution de la concentration d'une protéine forme généralement une sigmoïde lorsque sa production augmente ou diminue. Cela avait été théorisé notamment par Stuart A. Kauffman (1969) puis par René Thomas (1973) dans le cadre de formalismes booléens, c'est-à-dire restreints à deux niveaux discrets par composant, généralement notés « 0 » et « 1 ». Cette approche a naturellement été étendue par la suite à des formalismes multivalués, où chaque élément peut posséder plus de deux niveaux d'expression, généralement représentés par des entiers consécutifs. Cette approche a l'avantage d'abstraire les valeurs des seuils de concentration, qui sont généralement mal connues mais qui permettent de représenter le niveau de concentration à partir duquel la protéine va influencer un autre composant. Ainsi, à chaque niveau d'expression d'un composant est associé un ensemble de régulations sur d'autres composants.

Le dernier élément permettant de caractériser la dynamique est l'ajout d'une **dimension temporelle**. Dans le cadre des réseaux de régulation discrets, ce temps prend la forme d'une série infinie de pas de temps discrets permettant de représenter les évolutions successives du modèle au cours du temps, sans indication des durées réelles entre ces pas de temps. La question du synchronisme des formalismes développés se pose alors : en effet, s'il existe des systèmes purement synchrones, où entre chaque pas de temps, toutes les réactions sensibilisées ont lieu, l'hypothèse de René Thomas (1973) est au contraire d'abstraire les systèmes biologiques à l'aide de réseaux de régulation purement **asynchrones**. En effet, en l'absence de données temporelles sur le système étudié (vitesse des réactions, durée des dégradations...) il n'est pas possible d'assurer que deux protéines dont la production est activée au même moment seront produites en mêmes concentrations simultanément, ou que leurs concentrations passeront en même temps un seuil d'expression (représenté par un niveau discret).

C'est à partir de ces hypothèses ayant pour but la représentation de réseaux de régu-

lation biologique discrets et asynchrones qu'a été formalisée la version actuelle du modèle de Thomas (Richard, Comet & Bernot, 2006), et c'est en s'inspirant de celle-ci qu'ont été conçues par la suite les Frappes de Processus (Paulevé, Magnin & Roux, 2011*a*). Ces deux formalismes permettent en effet de représenter les interactions entre différents composants sous la forme de l'évolution séquentielle de niveaux d'expression discrets. Cependant, ils se distinguent principalement au niveau de la représentation des interactions entre composants. En effet, le modèle de Thomas considère les interactions entre composants du point de vue d'« influences » globales, c'est-à-dire d'interactions qui vont globalement avoir le rôle d'activer ou inhiber les composants entre eux, tandis que les Frappes de Processus font usage d'« actions », qui décrivent le saut d'un état local d'un composant à un autre. De plus, une restriction particulière porte sur ces actions, qui ne permettent la modification du niveau local d'un composant que par au plus un autre composant. Cette différence de point de vue en implique une autre en ce qui concerne la représentation des coopérations entre composants : là où le modèle de Thomas fait usage de paramètres décrivant les états focaux d'un élément en fonction de l'activité de l'ensemble de ses régulateurs (Snoussi, 1989), les Frappes de Processus introduisent un composant supplémentaire propre à la modélisation et qui joue le rôle de porte logique.

Les Frappes de Processus ont fait précédemment l'objet de plusieurs travaux. Ceux-ci ont notamment permis de montrer qu'elles permettent de représenter une « superposition » de modèles de Thomas, afin de représenter des ensembles de paramètres partiellement connus. De plus, un travail approfondi a permis d'élaborer de puissantes analyses statiques portant sur la recherche d'états stables, mais aussi sur des questions dynamiques comme l'atteignabilité d'un état local (Paulevé, Magnin & Roux, 2012) qui peut être traitée de façon très efficace, et donc **appliquée à de très grands modèles**.

1.3 Contributions

Les apports de cette thèse se déclinent en trois points principaux :

- l'enrichissement des Frappes de Processus par l'introduction de notions de prévalence et de synchronisme entre les différentes évolutions possibles ;
- le développement de méthodes d'analyse de la dynamique efficaces et adaptées aux enrichissements précédemment mentionnés, et leur application concluante à des réseaux de régulation biologique de grande taille ;
- la description des liens formels entre les différentes sémantiques de Frappes de Processus proposées, ainsi qu'avec d'autres formalismes courants pour la représentation des réseaux de régulation biologique.

L'objectif principal de cette thèse est donc de répondre à la double problématique de la conception et de l'analyse d'un modèle, en nous concentrant particulièrement sur l'utilisation et l'amélioration des Frappes de Processus et des outils associés. Les trois aspects précédents sont détaillés dans la suite afin d'insister sur leur rôle dans cette démarche.

Enrichissement des Frappes de Processus

Cette thèse se concentre sur plusieurs **alternatives d'extension du formalisme des Frappes de Processus dans le but d'enrichir celles-ci**. L'un des objectifs de cet enrichissement est *la prise en compte de données temporelles*, qui étaient généralement abstraites

auparavant, comme des durées relatives de réactions biochimiques ou des notions de retard entre les évolutions de certains éléments. afin d'étudier leur influence sur la dynamique. Cependant, plutôt que d'intégrer des données continues dans le modèle, nous proposons de traduire ces données par des formes nouvelles de dynamique. Pour cela, nous proposons trois approches qui gravitent autour des notions de **préemption** et de **synchronisme** entre les différentes évolutions possibles du modèle :

- les **classes de priorités** permettent de définir des règles globales de prévalence entre des ensembles d'actions, afin d'affiner la dynamique et d'obtenir une expressivité équivalente à celle des réseaux booléens,
- les **arcs neutralisants** proposent de raffiner la notion précédente, en définissant les prévalences de façon plus atomique entre les actions individuelles,
- les **actions plurielles**, enfin, permettent d'introduire des comportements synchrones entre les actions, dans le but de modéliser des phénomènes simultanés comme la création des produits d'une réaction.

Nous montrons par ailleurs comment les différentes données temporelles peuvent être intégrées à l'aide de ces nouveaux outils.

Outils efficaces d'analyse de la dynamique

Nous développons par ailleurs des méthodes permettant d'analyser la dynamique des modélisations proposées ci-dessus. Ces méthodes sont basées sur de l'analyse statique par interprétation abstraite précédemment proposée par Paulevé et al. (2012) : leur fonctionnement repose sur l'abstraction de la dynamique globale du système au profit des dynamiques locales de chaque composant. Nous enrichissons cette approche afin d'adapter les méthodes développées aux nouvelles formes de dynamique que nous proposons. Elles ont l'avantage de posséder **une complexité polynomiale en la taille du modèle**, ce qui permet de **traiter efficacement de très grands modèles, de l'ordre de centaines de composants, en quelques dixièmes de seconde**. Néanmoins, étant donné qu'elles sont basées sur une approximation de la dynamique, il est possible qu'elles terminent sans pouvoir conclure, bien que cela ne soit jamais arrivé sur les exemples testés. Nous appliquons notamment ces méthodes à un réseau booléen de quatre-vingt-quatorze (94) composants préalablement traduit en Frappes de Processus, et nous montrons que cela permet de conclure en moins d'une seconde sur plusieurs questions dynamiques.

Équivalences entre les Frappes de Processus

L'analyse statique mentionnée précédemment nécessite l'utilisation de **Frappes de Processus canoniques**, une classe particulière consistant en une restriction de l'un des formalismes mentionnés précédemment. Celle classe présente d'autres intérêts : nous montrons notamment qu'elle est **aussi expressive que tous les formalismes de Frappes de Processus** développés dans cette thèse, et nous donnons les traductions correspondantes. Cela assure notamment qu'il est toujours possible de naviguer entre les formalismes, afin de profiter de leurs avantages respectifs, et notamment de l'utilisation des analyses statiques mentionnées ci-dessus à tous les autres types de Frappes de Processus, moyennant une traduction du modèle.

Équivalences avec d'autres formalismes

Enfin, nous nous intéressons également aux **liens formels entre les différentes sémantiques de Frappes de Processus et les autres modélisations classiques pour la représentation des réseaux de régulation biologique**. Nous nous intéressons notamment au **modèle de Thomas** et aux **réseaux booléens**, deux formalismes proches développés spécifiquement pour la représentation de réseaux de régulations, et nous montrons qu'il est possible de représenter ces deux formalismes de façon exacte grâce à l'ajout de priorités dans les Frappes de Processus. Nous abordons aussi la question de la traduction vers les **réseaux de Petri**, un formalisme plus généraliste mais qui est l'objet d'un intérêt important du fait des capacités d'expressivité et d'analyse qu'il offre, et les intérêts qu'il présente aussi pour la modélisation des réseaux biologiques (Petri, 1962; Chaouiya, 2007). Ces liens permettent d'une part de comprendre la position des Frappes de Processus au sein de l'ensemble des modélisations discrètes asynchrones, et offrent d'autre part des outils de traduction depuis et vers ces autres formalismes, donnant la possibilité d'utiliser les outils d'analyse spécifiquement développés pour ceux-ci.

Les liens d'équivalence entre les différents enrichissements des Frappes de Processus proposés dans ce manuscrit sont résumés au sein de la figure 1.1, où nous faisons aussi apparaître les liens avec d'autres formalismes discrets qui ont été traités dans le cadre de ce travail.

1.4 Collaborations

Une partie du travail de cette thèse a été réalisé dans le cadre d'un stage doctoral dans l'équipe de recherche de Katsumi Inoue, au National Institute of Informatics (Tokyo, Japon). Le sujet de ce stage était : « Raisonnement automatique et recherche d'hypothèses pour la biologie des systèmes. » Ont ainsi participé financièrement à ce travail le National Institute of Informatics, *via* l'Inoue Laboratory, ainsi que la Fondation Centrale Initiatives.

Ce travail s'inscrit par ailleurs dans le projet de recherche ANR blanc BioTempo[1] dont l'intitulé était : « Représentations à l'aide de langage, de temps et de modèles hybrides pour l'analyse de modèles incomplets en biologie moléculaire », et qui s'est étendu de mars 2011 à août 2014. Le présent travail répond notamment à certains objectifs de la tâche 3 du projet, qui concerne l'introduction de synchronisations et de données chronométriques dans les modèles chronologiques.

1.5 Organisation du manuscrit

Le présent manuscrit est organisé de la manière suivante.

Le chapitre 2 offre un état des lieux en matière de modélisation et d'analyse des réseaux de régulation biologique discrets et asynchrones. Nous nous intéresserons notamment aux réseaux booléens et au modèle de Thomas, deux types de réseaux très répandus pour des raisons historiques comme nous l'avons déjà mentionné. Nous définirons aussi les Frappes de Processus standards, c'est-à-dire telles que définies par le travail de Loïc Paulevé (2011), car elles constituent le point de départ de ce travail.

[1] Le site du projet est disponible à `http://biotempo.genouest.org/`

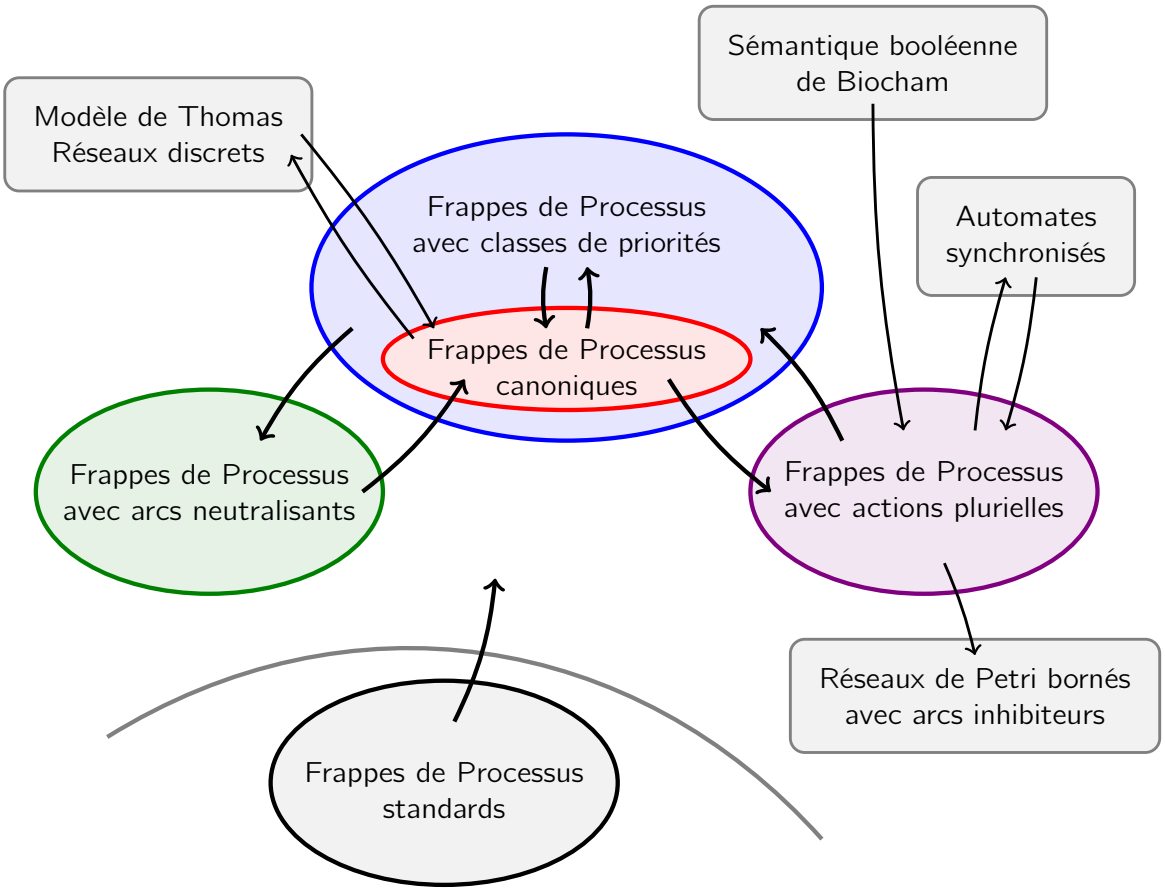


Figure 1.1 – Relations entre les différents enrichissements des Frappes de Processus. Les nœuds rectangulaires représentent des formalismes discrets extérieurs aux Frappes de Processus tandis que les flèches représentent les différentes traductions formalisées dans cette thèse.

Le chapitre 3 propose plusieurs ajouts aux Frappes de Processus afin de l'enrichir et d'en augmenter l'expressivité. Les trois formalismes abordés reposent sur les notions de classes de priorités, d'arcs neutralisants et d'actions plurielles, afin de filtrer les comportements désirés du modèle ou d'ajouter des comportements nécessaires à la modélisation de certains phénomènes. Nous discutons aussi dans ce chapitre des applications possibles de ces formalismes, et traitons en partie la question des équivalences et des traductions entre ces formalismes.

Le chapitre 4 se concentre sur une classe particulière de Frappes de Processus avec classes de priorités, dite « canonique ». Cette classe sert de base au développement de méthodes d'analyse statique permettant de vérifier des propriétés d'atteignabilité locales dans un modèle. Ces méthodes utilisent un mécanisme d'abstraction pour éviter l'explosion combinatoire inhérente à toute étude de la dynamique d'un modèle discret, et ainsi rester efficaces. Nous montrons aussi, traduction à l'appui, que les Frappes de Processus canoniques sont en réalité équivalentes aux trois extensions des Frappes de Processus proposées, ce qui permet d'étendre la portée des résultats précédents à ces formalismes.

Le chapitre 5 dépasse le cadre des Frappes de Processus et s'intéresse aux liens formels entre les extensions proposées plus haut et plusieurs formalismes classiques discrets permettant la modélisation des réseaux de régulation biologique. Nous y prouvons que les extensions des Frappes de Processus permettent d'obtenir la même expressivité que le modèle de Thomas et les réseaux booléens, ce qui étend encore la portée de nos résultats. Nous montrons aussi que les Frappes de Processus sont aussi expressives que le formalisme classique des automates synchronisés. Enfin, nous proposons une traduction vers les réseaux de Petri avec arcs de lecture et arcs inhibiteurs, et, à l'inverse, une traduction depuis le systèmes d'équations biochimiques de la sémantique booléenne de Biocham.

Le chapitre 6 porte sur l'application des modèles et méthodes proposées dans cette thèse à des réseaux de régulation biologique de grande taille. Nous y présentons plusieurs systèmes biologiques classiquement étudiés, la façon dont ils sont représentés en Frappes de Processus, et les résultats de nos méthodes de traduction et d'analyse.

Enfin, le chapitre 7 nous permet de conclure ce manuscrit et d'en discuter les résultats afin d'ouvrir sur des perspectives de travaux futurs.

1.6 Notations

Nombres réels On note $\mathbb{R}$ l'ensemble des nombres réels. Si $i, j \in \mathbb{R}$, on note $[i\,;j] = \{x \in \mathbb{R} \mid i \leq x \leq j\}$ l'ensemble des nombres réels entre i et j compris.

Entiers naturels On note $\mathbb{N}$ l'ensemble des entiers naturels, $\mathbb{N}^* = \mathbb{N} \setminus \{0\}$ l'ensemble des entiers naturels strictement positifs, et $\mathbb{N}^\bullet = \mathbb{N}\setminus\{0,1\}$ l'ensemble des entiers naturels supérieurs ou égaux à 2. Si $i, j \in \mathbb{N}$, $i < j$, on note $[\![i\,;j]\!] = \{i, i+1, \ldots, j-1, j\}$ l'ensemble des entiers naturels entre i et j compris.

Pour tous entiers $i, j, k \in \mathbb{N}$, on note : $k < [\![i\,;j]\!] \overset{def}{\Leftrightarrow} k < i$ et $k > [\![i\,;j]\!] \overset{def}{\Leftrightarrow} k > j$. De plus, si $i_1, i_2, j_1, j_2 \in \mathbb{N}$, on note :

$$[\![i_1\,;j_1]\!] \leq_{[\![]} [\![i_2\,;j_2]\!] \overset{def}{\Leftrightarrow} (i_1 \leq i_2 \wedge j_1 \leq j_2)$$

$$\text{et :} \qquad [\![i_1\,;j_1]\!] <_{[\![]} [\![i_2\,;j_2]\!] \overset{def}{\Leftrightarrow} (i_1 < i_2 \wedge j_1 \leq j_2) \vee (i_1 \leq i_2 \wedge j_1 < j_2)$$

Entiers relatifs On note $\mathbb{Z}$ l'ensemble des entiers relatifs.

Par ailleurs, la fonction signe est définie sur les entiers relatifs comme suit :

$$\begin{aligned} \text{signe} : \mathbb{Z} &\to \{+, -, \varnothing\} \\ n &\mapsto \begin{cases} + & \text{si } x > 0 \\ - & \text{si } x < 0 \\ \varnothing & \text{if } x = 0 \end{cases} \end{aligned}$$

Séquences Si $n \in \mathbb{N}$, on note $e_1 :: \ldots :: e_n$ la séquence finie formée des éléments $e_1, \ldots, e_n$ si $n \geq 1$, et on note ε la séquence vide (si $n = 0$).

Pour toute séquence finie $E = e_1 :: \ldots :: e_n$, on note $|E| = n$ la longueur de cette séquence, et $\mathbb{I}^E = [\![1\,;|E|]\!]$ l'ensemble des indices de cette séquence. Pour tout $i \in \mathbb{I}^E$, on note $A_i = e_i$ le i^e élément de E, et pour tout $i, j \in \mathbb{I}^E$, on note $E_{i..j} = e_i :: \ldots :: e_j$ la sous-séquence formée des éléments i à j de E ; naturellement, $E_{i..j} = \varepsilon$ si $i > j$.

On note de plus : $e \in E \Leftrightarrow \exists i \in \mathbb{I}^E, e = E_i$

Ensembles Le cardinal d'un ensemble A est noté $|A|$ et son ensemble des parties est noté $\wp(A)$.

Si A et B sont deux ensembles, on note $A \cup B$ leur union, $A \cap B$ leur intersection et $A \times B$ leur produit cartésien.

Si $n \in \mathbb{N}$, et $\{A_i\}_{i \in [\![1;n]\!]}$ est un ensemble d'ensembles, on note $\bigcup_{i \in [\![1;n]\!]} A_i = A_1 \cup A_2 \cup \ldots \cup A_n$ leur union, $\bigcap_{i \in [\![1;n]\!]} A_i = A_1 \cap A_2 \cap \ldots \cap A_n$ leur intersection et $\bigotimes_{i \in [\![1;n]\!]} A_i = A_1 \times A_2 \times \ldots \times A_n$ leur produit cartésien. Cette définition peut naturellement être étendue à une séquence d'ensembles. De plus, par convention : $\bigcup_\emptyset = \bigcap_\emptyset = \bigotimes_\emptyset = \bigcup_\varepsilon = \bigcap_\varepsilon = \bigotimes_\varepsilon = \emptyset$.

Si $E = e_1 :: \ldots :: e_n$ est une séquence, on note $\widetilde{E} = \{e_1, \ldots, e_n\}$ l'ensemble correspondant. De même, si $T = (t_1, \ldots, t_n)$ est un n-uplet, on définit : $\widetilde{T} = \{t_1, \ldots, t_n\}$.

Fonctions et plus petit point fixe Si A et B sont deux ensembles, on note $f : A \to B$ si f est une fonction qui associe chaque élément de A à un élément de B.

On note de plus $\mathbf{pppf}\{x_0\}\,(x \mapsto x')$ le plus petit point fixe plus grand que x_0 de la fonction $x \mapsto x'$, s'il existe.

Recouvrement L'opérateur $\Subset$, qui désigne le recouvrement d'un état par un autre, est donné à la définition 2.11 en page 28. Il est ensuite étendu aux contextes à la définition 4.16 en page 82 ainsi qu'aux ensembles de processus à la définition 5.5 en page 111.

Chapitre 2

État de l'art de la modélisation

Nous proposons dans ce chapitre un état de l'art des formalisations discrètes et asynchrones des réseaux de régulation biologique utilisées dans cette thèse. Nous y rappelons le principe et la définition du *modèle de Thomas* et nous revenons brièvement sur les méthodes d'analyse qui existent sur ce modèle. Nous rappelons aussi le formalisme des *Frappes de Processus standards*, qui seront la base des travaux présentés dans cette thèse, et nous faisons une rapide revue des principaux travaux qui ont concerné ce formalisme.

L'état de l'art des Frappes de Processus standards reprend des éléments de (Paulevé, Chancellor, Folschette, Magnin & Roux, 2014).

Les outils de modélisation des réseaux de régulation biologique se déclinent en de nombreuses formes permettant de représenter différents comportement des systèmes étudiés. Au cours de cette thèse, nous nous intéressons tout particulièrement aux modèles discrets et asynchrones.

Les modèles discrets permettent de représenter des systèmes plus complexes, comme des systèmes d'équations différentielles, en abstrayant une partie de la dynamique. Cette abstraction permet de simplifier le modèle pour en faciliter l'analyse, à condition de rester cohérente avec la représentation initiale. Le caractère asynchrone des formalismes vient quant à lui de la constatation suivante : il est biologiquement très improbable que plusieurs entités d'un système évoluent en parfaite simultanéité. Le parallèle avec les systèmes d'équations différentielles est le suivant : il est rare d'observer plusieurs composants passer un seuil simultanément au cours d'une évolution continue de leur état. Ces hypothèses ont notamment été théorisées par René Thomas (1973), qui en a dérivé le modèle qui porte son nom, plus tard enrichi par plusieurs travaux successifs comme l'ajout de paramètres discrets par El Houssine Snoussi (1989) pour représenter les « états focaux » des différents composants en fonction de l'état du modèle.

Dans ce chapitre, nous rappellerons tout d'abord la définition du modèle de Thomas à la section 2.1. Nous dresserons par ailleurs un rapide état de l'art des méthodes d'analyse de la dynamique de ce modèle qui, bien que facilitées par l'abstraction discrète et asynchrone qui en est inhérente, restent l'objet d'une explosion combinatoire non négligeable. Plusieurs

travaux apportent des résultats qui s'appuient uniquement sur les données du modèles et non sur le calcul de sa dynamique, ce qui évite de calculer celle-ci explicitement. Cependant, ces résultats sont très généraux car ils se focalisent le plus souvent sur la présence ou l'absence de comportements dans le modèle (comme des oscillations ou des points fixes). Il est donc rare de pouvoir totalement faire l'impasse sur une analyse plus détaillée de la dynamique, qui nécessite cependant de calculer l'espace des états du modèle, ou une version compressée de celui-ci.

Nous définirons ensuite le formalisme des Frappes de Processus à la section 2.2, tel qu'il a été proposé par Paulevé et al. (2011*a*). Il a été conçu comme une alternative complémentaire au modèle de Thomas avec lequel il partage certaines hypothèses, à savoir la discrétisation des états et l'asynchronisme de la dynamique, bien qu'il soit plus atomique par ailleurs dans sa représentation. Il permet notamment de modéliser tout modèle de Thomas, avec cependant une légère sur-approximation de la dynamique. Bien que cette sur-approximation permette de représenter une classe de modèles de façon abstraite, elle est cependant indésirable lorsque l'on souhaite modéliser fidèlement certains comportements. Des outils ont de surcroît été développés afin de calculer les points fixes d'un modèle de Frappes de Processus ou encore d'y intégrer des paramètres continus sous forme de probabilités. Cependant, l'atout principal des Frappes de Processus réside dans les puissantes méthodes d'analyse statique par interprétation abstraite qui y sont associées, permettant ainsi d'effectuer des calculs d'atteignabilité locale avec une complexité polynomiale dans la taille du modèle considéré. Cela permet notamment d'étudier la dynamique de grands modèles — jusqu'à plusieurs centaines de composants, voire au-delà.

L'état de l'art de la modélisation à l'aide des Frappes de Processus standards a fait l'objet d'un chapitre de livre plus détaillé (Paulevé, Chancellor, Folschette, Magnin & Roux, 2014).

2.1 Le Modèle de Thomas

Nous présentons dans cette section le *modèle de Thomas* (section 2.1.1), historiquement très utilisé dans la représentation des réseaux de régulation biologique. Il s'agit d'un modèle asynchrone basé sur un *graphe des interactions* et une carte de *paramètres* discrets. Les *réseaux discrets asynchrones* sont aussi définis (section 2.1.2) du fait de leurs similitudes avec le modèle de Thomas. Nous proposons enfin un rapide tour d'horizon des différentes méthodes utilisées pour analyser la dynamique de ces modèles (section 2.1.3).

2.1.1 Définition du modèle de Thomas

Le modèle dit *de Thomas* a été théorisé et proposé pour la première fois par René Thomas (1973) dans le but d'abstraire les systèmes d'équations différentielles permettant la représentation de réseaux de régulation biologique. Son intuition était de proposer une simplification discrète cohérente de ces systèmes afin de permettre des analyses plus efficaces. Nous proposons ici une définition inspirée de divers travaux plus récents, comportant notamment des paramètres discrets (Snoussi, 1989) et des arcs non-signés (Folschette, Paulevé, Inoue, Magnin & Roux, 2012*b*).

Un modèle de Thomas représente un ensemble fini de composant se régulant entre eux. Ces composants sont décrits par un niveau d'expression discret qui caractérise leur état

(taux d'activité pour un gène, concentration pour une protéine, etc.). Afin de représenter la structure d'un tel système, on utilise généralement un *graphe des interactions* (définition 2.1) dont les nœuds représentent un ensemble de composants et les arcs (orientés) leurs influences mutuelles. Les nœuds sont étiquetés par un nom (celui du composant : a, b, c, etc.) et un plafond (son niveau d'expression maximum : l_a, l_b, l_c, etc.). Les arcs sont de la forme : $a \xrightarrow{s,t} b$, c'est-à-dire étiquetés par un signe s qui représente le type de régulation ($+$ pour une activation, $-$ pour une inhibition et $\circ$ pour une régulation plus complexe) et un entier t qui représente le seuil de déclenchement de la réaction (c'est-à-dire le niveau d'expression du composant régulateur à partir duquel celui-ci a effectivement une influence sur le composant régulé). Bien que plusieurs variantes aient été proposées, nous nous intéressons ici à une définition où chaque arc est unique, c'est-à-dire qu'il ne peut pas exister deux arcs $a \xrightarrow{s,t} b$ et $a \xrightarrow{s',t'} b$ étiquetés différemment dans le graphe. L'utilité des arcs non-signés ($\circ$) est discutée à la fin de cette section.

Définition 2.1 (Graphe des interactions)**.** Un *graphe des interactions* est un couple $\mathcal{G} = (\Gamma\,; E)$ où Γ est l'ensemble fini des *composants*, étiquetés par un nom et un *plafond*, et E est l'ensemble fini des *régulations* entre deux nœuds, étiquetées par un *signe* et un *seuil* :

$$E \stackrel{def}{=} \{a \xrightarrow{s,t} b, \ldots \mid a, b \in \Gamma \wedge s \in \{+, -, \circ\} \wedge t \in [\![1\,; l_a]\!]\}$$

tel que chaque régulation de a vers b soit unique :

$$\forall a \xrightarrow{s,t} b \in E, \forall a \xrightarrow{s',t'} b \in E, s = s' \wedge t = t' \ .$$

Étant donnée cette définition, on note $E_s \stackrel{def}{=} \{a \xrightarrow{s,t} b \in E\}$ pour $s \in \{+, -, \circ\}$. De plus, pour tout composant $b \in \Gamma$, on note $\mathcal{G}^{-1}(b)$ l'ensemble de ses *régulateurs* :

$$\mathcal{G}^{-1}(b) \stackrel{def}{=} \{a \in \Gamma \mid \exists a \xrightarrow{s,t} b \in E\}$$

Exemple. La figure 2.1(gauche) représente un graphe des interactions $(\Gamma\,; E)$ où $\Gamma = \{a, b, c\}$, avec $l_a = 2$ et $l_b = l_c = 1$, et :

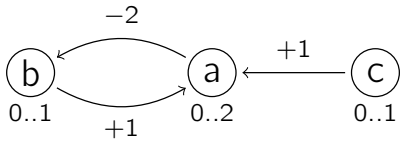
$$E_+ = \{b \xrightarrow{+,1} a, c \xrightarrow{+,1} a\} \qquad E_\circ = \emptyset$$
$$E_- = \{a \xrightarrow{-,2} b\}$$

Ainsi :

$$\mathcal{G}^{-1}(a) = \{b, c\} \qquad \mathcal{G}^{-1}(b) = \{a\}$$
$$\mathcal{G}^{-1}(c) = \emptyset$$

Pour des raisons d'illustration, le composant a ne comporte aucun arc sortant avec le seuil 1, mais possède un arc sortant étiqueté avec le seuil 2. Ce type de configuration ne se rencontre habituellement pas dans les modèles de Thomas pour des raisons discutées en page 20.

Pour tout composant a régulant b, c'est-à-dire si $a \xrightarrow{s,t} b \in E$, on note $\mathsf{niveaux}(a \to b)$ (resp. $\overline{\mathsf{niveaux}}(a \to b)$) l'ensemble des niveaux d'expression de a qui sont au-dessus (resp. en-dessous) du seuil t (définition 2.2). Au niveau de la dynamique, pour tout niveau d'expression de a appartenant à $\mathsf{niveaux}(a \to b)$, a est censé avoir une influence correspondant



$$K_{a,\emptyset} = [\![0\,;0]\!] \qquad K_{b,\emptyset} = [\![0\,;1]\!]$$
$$K_{a,\{b\}} = [\![1\,;1]\!] \qquad K_{b,\{a\}} = [\![0\,;0]\!]$$
$$K_{a,\{c\}} = [\![1\,;1]\!]$$
$$K_{a,\{b,c\}} = [\![2\,;2]\!] \qquad K_{c,\emptyset} = [\![0\,;1]\!]$$

Figure 2.1 – (gauche) Un exemple de graphe des interactions. Les composants sont représentés par les nœuds, comportant un nom et un ensemble de niveaux d'expression, tandis que les régulations sont représentées par des arcs étiquetés par un signe et un seuil. Par exemple, l'arc de b vers a étiqueté $+1$ représente la régulation $b \xrightarrow{+,1} a$. En d'autres termes, b se comporte comme un activateur de a si son niveau d'expression est égal à 1, et se comporte comme un inhibiteur sinon (c'est-à-dire si son niveau d'expression est égal à 0). (droite) Un exemple de paramétrisation du graphe des interactions de gauche.

au signe s sur b, c'est-à-dire être activateur si $s = +$, inhibiteur si $s = -$, ou avoir une influence indéterminée ou multiple si $s = \circ$; en revanche, pour tout niveau d'expression de a appartenant à $\overline{\text{niveaux}}(a \to b)$, l'influence opposée devrait être observée. Cette hypothèse permet de modéliser la dégradation de b en l'absence de l'activation de a si $s = +$, ou l'activation de b en l'absence de l'inhibition de a si $s = -$.

Définition 2.2 (Niveaux effectifs (niveaux))**.** Soit $\mathcal{G} = (\Gamma\,; E)$ un graphe des interactions. Si $a \xrightarrow{s,t} b \in E$, on définit :

$$\text{niveaux}(a \to b) \overset{def}{=} [\![t\,; l_a]\!] \quad \text{et} \quad \overline{\text{niveaux}}(a \to b) \overset{def}{=} [\![0\,; t-1]\!]$$

Exemple. Sur le graphe des interactions de la figure 2.1(gauche) on a notamment :

$$\text{niveaux}(a \to b) = [\![2\,;2]\!] \qquad \overline{\text{niveaux}}(a \to b) = [\![0\,;1]\!]$$

Un *état* d'un graphe des interactions $(\Gamma\,; E)$ est un élément de l'ensemble $\mathcal{S} \overset{def}{=} \prod_{a\in\Gamma}[\![0\,; l_a]\!]$. $s[a]$ se rapporte au niveau d'expression du composant a dans s. Pour tout état, l'ensemble des *ressources* d'un composant donné est l'ensemble de ses régulateurs dont le niveau d'expression est supérieur ou égal au seuil de la régulation (définition 2.3). En d'autres termes, pour chaque état s, tout régulateur b d'un composant a est une ressource si et seulement si $s[b] \in \text{niveaux}(b \to a)$ (et, inversement, n'en est pas une si $s[b] \in \overline{\text{niveaux}}(b \to a)$). Cependant, la dynamique manque de précision dans toute situation où un composant est à la fois activé et inhibé par deux composants différents. C'est pour supprimer ce flou que Snoussi (1989) propose l'ajout d'une *paramétrisation*, c'est-à-dire d'un ensemble de *paramètres* discrets qui tiennent lieu de points focaux : à chaque configuration de ressources d'un composant est associé un paramètre qui détermine le niveau vers lequel le composant va évoluer. Nous proposons à la définition 2.4 une extension de cette notion de paramètre à un intervalle d'entiers afin de gagner en expressivité (Folschette et al., 2012*b*). L'intérêt des paramètres sous forme d'intervalles est discuté à la fin de cette section.

Définition 2.3 (Ressources (Res))**.** Soit $\mathcal{G} = (\Gamma\,;E)$ un graphe des interactions. Pour tout composant $a \in \Gamma$ et tout état $s \in \mathcal{S}$, on appelle *ressources de a dans s* et on note $\mathsf{Res}_a(s)$ l'ensemble des régulateurs de a dont le niveau dans s est supérieur au seuil de la régulation qui les relie à a :

$$\mathsf{Res}_a(s) \stackrel{def}{=} \{b \in \mathcal{G}^{-1}(a) \mid s[b] \in \mathsf{niveaux}(b \to a)\}$$

Définition 2.4 (Paramètre $K_{a,\omega}$ et paramétrisation K)**.** Soit $\mathcal{G} = (\Gamma\,;E)$ un graphe des interactions. Pour un composant $a \in \Gamma$ donné et $\omega \subset \mathcal{G}^{-1}(a)$ un sous-ensemble de ses régulateurs, le *paramètre* $K_{a,\omega} = [\![i\,;j]\!]$ est un intervalle non-vide tel que $0 \leq i \leq j \leq l_a$. La carte complète K des paramètres sur un graphe des interactions $\mathcal{G}$ est appelée *paramétrisation de* $\mathcal{G}$.

Un graphe des interactions et une paramétrisation permettent de représenter un réseau de régulation biologique complet grâce à sa structure et son évolution. En effet, un paramètre $K_{a,\omega}$ représente un ensemble de valeurs vers lesquelles le composant a évolue dans tout état où l'ensemble de ses ressources est égal à ω. Plus précisément, a va évoluer vers la valeur de $K_{a,\omega}$ qui est la plus proche de son niveau d'expression courant. On appellera dans la suite *modèle de Thomas* tout couple $(\mathcal{G}\,;K)$ formé d'un graphe des interactions et d'une paramétrisation.

Pour finir, René Thomas a formulé deux hypothèses concernant la dynamique de ses modèles. Tout d'abord, elle doit être asynchrone, c'est-à-dire qu'un seul composant peut évoluer entre chaque état. Cette hypothèse rend compte du fait qu'il est infiniment peu probable que deux composants passent en même temps un seul d'expression discret. Il a de plus proposé de la rendre unitaire ; autrement dit, chaque composant ne peut évoluer que d'un niveau d'expression discret à la fois. Ces deux hypothèses permettent de conserver un certain nombre de propriétés propres aux systèmes d'équations différentielles dans lesquels l'évolution de chaque composant est continue. Nous définissons donc la dynamique d'un modèle de Thomas avec paramètres discrets comme suit : il existe une transition d'un état s vers un autre état s' si et seulement si il existe un unique un composant a qui évolue entre ces deux états, d'exactement un niveau d'expression et vers le paramètre $K_{a,\mathsf{Res}_a(s)}$ (définition 2.5). Il est à noter cependant que a ne peut pas évoluer si son niveau d'expression dans l'état s appartient déjà à l'intervalle du paramètre $K_{a,\mathsf{Res}_a(s)}$.

Définition 2.5 (Dynamique unitaire d'un modèle de Thomas ($\to$))**.** Pour tout modèle de Thomas $\mathsf{T} = (\mathcal{G}\,;K)$, La dynamique de T est donnée par la relation de transition $\to \in \mathcal{S} \times \mathcal{S}$ définie par :

$$\forall s, s' \in \mathcal{S}, s \to s' \iff \exists a \in \Gamma, s[a] \notin K_{a,\mathsf{Res}_a(s)} \wedge s'[a] = s[a] + \delta^a(s)$$
$$\wedge\, \forall b \in \Gamma, b \neq a \Rightarrow s[b] = s'[b]$$

$$\text{avec : } \delta^a(s) = \begin{cases} +1 & \text{si } s[a] < K_{a,\mathsf{Res}_a(s)} \\ -1 & \text{si } s[a] > K_{a,\mathsf{Res}_a(s)} \end{cases}$$

Les symboles « $<$ » et « $>$ » de cette définition permettant de comparer un entier à un intervalle sont définis à la section 1.6 en page 12.

Exemple. La figure 2.1(droite) donne un exemple de paramétrisation du graphe des interactions de la figure 2.1(gauche), ce qui en fait un modèle de Thomas complet, dont la figure 2.2 donne l'espace des états. On note notamment la présence de trois état stables

pour ce modèle, c'est-à-dire trois états depuis lesquels plus aucune évolution n'est possible : $\langle a_0, b_0, c_0\rangle$, $\langle a_1, b_1, c_0\rangle$ et $\langle a_1, b_0, c_1\rangle$, où x_i représente le niveau d'expression i pour le composant x.

On peut observer l'aspect unitaire de la dynamique d'un modèle de Thomas sur ce graphe. En effet, malgré le paramètre $K_{a,\{b,c\}} = [\![2\,;2]\!]$, le composant a ne peut pas directement passer de l'état a_0 à l'état a_2 en « sautant » l'état a_1. C'est pourquoi on observe les transitions $\langle a_0, b_1, c_1\rangle \to \langle a_1, b_1, c_1\rangle$ et $\langle a_1, b_1, c_1\rangle \to \langle a_2, b_1, c_1\rangle$.

Nous notons enfin que les paramètres sous forme d'intervalles permettent notamment de rendre un composant immobile. C'est par exemple le cas du paramètre $K_{b,\emptyset} = [\![0\,;1]\!]$, qui est pris en compte lorsque a n'est pas au niveau 2. Dans une sémantique ne permettant que des paramètres unitaires, une auto-régulation de b serait nécessaire.

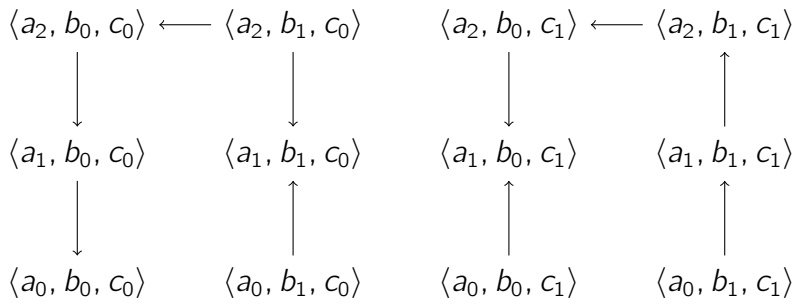


Figure 2.2 – Représentation de la dynamique du modèle de Thomas donné à la figure 2.1. Chaque état est représenté par un triplet $\langle a_i, b_j, c_k\rangle$ où i, j et k représentent respectivement le niveau d'expression de a, b et c, et une transition entre deux états est représentée par une flèche.

Graphe des interactions minimal

Il est possible d'inférer le graphe des interactions *minimal* d'un modèle de Thomas, en ne conservant que les arcs fonctionnels pour la dynamique. Définissons $f^a(s) = s[a]$ si $s[a] \in K_{a,\mathrm{Res}_a(s)}$, et $f^a(s) = s[a] + \delta^a(s)$ sinon. Une régulation positive (resp. négative) de b sur a n'est inférée que s'il existe un état s tel que l'augmentation du niveau de b aurait pour conséquence une augmentation (resp. diminution) de la valeur de f^a ; ou, en d'autres termes, s'il existe un état s tel que : $f^a(s \Cap b_i) <$ (resp. $>$) $f^a(s \Cap b_{i+1})$, où $i < l_b$ et $s \Cap b_i$ est l'état identique à s sauf pour la composante de b qui a été remplacée par b_i. Un tel graphe des interactions peut être utilisé pour inférer des propriétés globales sur la dynamique (cf. par exemple Richard, 2010; Paulevé & Richard, 2011).

Discussion sur la valeur du plafond

Le plafond l_a d'un composant a, qui est le niveau d'expression maximum de a, est généralement choisi comme égal au nombre $|\Gamma^+(a)|$ de régulations sortant du nœud a dans le graphe des interactions, c'est-à-dire le nombre de composants qu'il régule. En effet, chaque niveau discret dans $[\![0\,;l_a]\!]$ représente un ensemble arbitraire de valeurs pour une donnée réelle et généralement continue de a (niveau d'activité, concentration, etc.), et pour laquelle a possède une certaine influence sur plusieurs autres composants qu'il régule. Ainsi,

autoriser des valeurs de l_a plus grandes que $|\Gamma^+(a)|$ n'augmente pas l'expressivité du modèle car plusieurs niveaux d'expression de a auront alors le même rôle au niveau des régulations. À l'inverse, réduire ce plafond à des valeurs plus petites que $|\Gamma^+(a)|$ sous-entendrait que plusieurs seuils de régulations sortant de a sont identiques, ce qui est biologiquement peu plausible. Cependant, ces considérations peuvent être ignorées pour des raisons diverses de modélisation, et nous avons choisi de ne pas contraindre le plafond d'un composant dans ce document pour des raisons de simplicité. Ainsi, dans l'exemple de modèle de Thomas de la figure 2.1, il serait suffisant d'avoir $l_a = 1$ (et $a \xrightarrow{-,1} b \in E$) étant donné que a ne possède qu'une régulation sortante. La valeur $l_a = 2$ permet cependant de rendre l'exemple plus intéressant en montrant notamment l'aspect unitaire de la dynamique du modèle de Thomas (et $a \xrightarrow{-,2} b \in E$ peut être choisi arbitrairement).

Discussion sur les signes

L'ajout d'arcs non-signés ($\circ$) permet de modéliser des régulations dont la tendance générale est plus complexe qu'une simple activation ou inhibition. Il est à noter qu'utiliser $\circ$ comme signe pour étiqueter les régulations en plus des deux signes habituels $+$ et $-$ n'augmente pas l'expressivité du formalisme. En effet, les signes n'ont pas d'impact sur la paramétrisation (définition 2.4) ou sur la dynamique (définition 2.5). Ils ont cependant un but informatif, car ils résument les différentes régulations du modèle. Ils seront d'ailleurs utilisés au chapitre 5 comme base pour énumérer des paramétrisations incomplètes. Les arcs non-signés seront de plus exploités pour modéliser un graphe des interactions avec une connaissance partielle de certaines régulations. Cependant, comme ils ne sont pas courants dans la littérature, l'état de l'art de la section suivante portera principalement sur les modèles de Thomas sans arcs non-signés ($E_\circ = \emptyset$).

Discussion sur les intervalles de paramètres

Tandis que la littérature utilise principalement des paramètres entiers (c'est-à-dire comportant une unique valeur, par exemple : $K_{a,\omega} = i$) nous proposons ici d'utiliser des intervalles comme valeurs des paramètres (par exemple : $K_{a,\omega} = [\![i\,;j]\!]$). Ces intervalles permettent notamment de faciliter la représentation, bien qu'il soit intéressant de noter qu'ils augmentent l'expressivité du modèle de Thomas. En effet, considérons un cas fictif où $K_{a,\omega} = [\![i\,;i+2]\!]$ (c'est-à-dire un intervalle contenant trois valeurs) et s un état tel que $\mathsf{Res}_a(s) = \omega$. D'après la définition 2.5, si $s[a] \in K_{a,\omega}$, alors a ne peut pas évoluer dans s ; ainsi, a possède trois états locaux stables : a_i, a_{i+1} et a_{i+2}. Ce comportement ne peut pas être retrouvé à l'aide de paramètres entiers, car il n'est pas possible de distinguer les trois configurations de ressources correspondant aux trois états locaux. Au plus deux états stables locaux peuvent être créés à l'aide d'une auto-régulation positive sur a pour distinguer deux configurations différentes ($s[a] < t$ et $s[a] \geq t$) ce qui n'est cependant pas suffisant pour obtenir trois états stables.

2.1.2 Définition des réseaux discrets asynchrones

Certaines contraintes propres aux modèles de Thomas peuvent être relâchées pour permettre des comportements supplémentaires. Ainsi, il est courant de représenter des réseaux de régulation biologique sous la forme de *réseaux discrets asynchrones*. Ces réseaux sont

aussi fondés sur un graphe des interactions, mais il utilisent des fonctions d'évolution (définition 2.6) pour plus de permissivité, en lieu et place de paramètres discrets tels que précédemment formalisés à la définition 2.4 en page 19. Par ailleurs, l'hypothèse d'asynchronisme est conservée car un seul composant peut évoluer depuis chaque état, mais leur dynamique n'est pas unitaire dans le cas général car ce composant peut évoluer d'un nombre arbitraire de niveaux d'expression (définition 2.7).

Définition 2.6 (Réseau discret asynchrone (RDA))**.** Si $\mathcal{G} = (\Gamma\,; E)$ est un graphe des interactions, un *réseau discret asynchrone* est un couple $\mathsf{RDA} = (\mathcal{G}\,; F)$ avec $F = (f_x)_{x\in\Gamma}$, tels que $\forall x \in \Gamma, f_x : \mathcal{G}^{-1}(x) \to [\![0\,; l_x]\!]$.

Définition 2.7 (Dynamique d'un réseau discret asynchrone ($\to_{\mathsf{RDA}}$))**.** Pour tout réseau discret asynchrone $\mathsf{RDA} = (\mathcal{G}\,; F)$, La dynamique non-unitaire de RDA est donnée par la relation de transition $\to_{\mathsf{RDA}} \in \mathcal{S} \times \mathcal{S}$ définie par :

$$\forall s, s' \in \mathcal{S}, s \to_{\mathsf{RDA}} s' \iff \exists a \in \Gamma, s[a] \neq f_a(s) \wedge s'[a] = f_a(s) \\ \wedge \forall b \in \Gamma, b \neq a \Rightarrow s[b] = s'[b]$$

Enfin, nous désignons par *réseau booléen asynchrone* tout réseau discret asynchrone dont les composants ne possèdent que deux niveaux discrets, c'est-à-dire tel que : $\forall x \in \Gamma, l_x = 1$.

2.1.3 Analyses formelles du modèle de Thomas

Nous proposons dans cette section un tour d'horizon rapide des différentes méthodes d'analyse de la dynamique des modèles de Thomas. Nous nous concentrons principalement sur les méthodes d'analyse statique, c'est-à-dire permettant d'obtenir des résultats sur la dynamique d'un modèle sans nécessiter d'en calculer la dynamique de façon complète et explicite. Ces approches ont l'avantage de jouir d'une faible complexité, même si les résultats formels qu'elles fournissent sont généralement moins précis ou approximés. L'analyse du graphe des interactions seul peut ainsi permettre d'obtenir certains résultats sur sa dynamique, et la prise en compte de la paramétrisation permet d'affiner cette approche. Cependant, ces résultats sont insuffisants dans beaucoup de cas, nécessitant des approches plus exhaustives. L'analyse complète de la dynamique d'un modèle de Thomas est généralement invoquée en dernier recours, car elle peut être coûteuse en temps et en espace mémoire du fait de l'explosion combinatoire inhérente au calcul de l'espace des états.

Analyse du graphe des interactions

Plusieurs travaux se sont intéressés aux conclusions qui pouvaient être tirées du seul graphe des interactions. La recherche d'*états stables*, aussi appelés *points fixes*, présente un intérêt particulier dans l'étude des réseaux de régulation biologique et a été de fait très étudiée. Les conjectures de René Thomas (1981) tracent notamment un lien entre la présence de circuits dans les régulations et celle d'oscillations ou d'états stables. Elles se formulent comme suit :

– un circuit positif (c.-à-d. comportant un nombre pair de régulations négatives) est une condition nécessaire à la présence de plusieurs points fixes,

– un circuit négatif (c.-à-d. comportant un nombre impair de régulations négatives) est une condition nécessaire à la présence d'oscillations soutenues dans la dynamique, et donc à la présence d'un attracteur constitué d'au moins deux états.

Ces conjectures ont par la suite été démontrées dans le cas booléen, c'est-à-dire lorsque $\forall x \in \Gamma, l_x = 1$ (Remy, Ruet & Thieffry, 2008; Richard, 2006) ainsi que dans le cas multivalué (Richard & Comet, 2007; Richard, 2010).

Plusieurs résultats viennent de surcroît enrichir ce résultat concernant l'existence de points fixes. Des travaux ont par exemple permis d'obtenir une valeur maximale du nombre de points fixes possibles dans un modèle de Thomas tant booléen (Aracena, 2008) que multivalué (Richard, 2009). Cette valeur dépend du nombre de composants à supprimer pour éliminer tout circuit positif dans le modèle. Paulevé & Richard (2010) ont quant à eux proposé une valeur minimale du nombre de *points fixes topologiques* d'un modèle booléen. Ces points fixes ont la particularité de ne pas dépendre de la paramétrisation, ce qui permet de généraliser un tel résultat à un ensemble de modèles de Thomas.

Analyse de la paramétrisation

Les conditions données précédemment sont *nécessaires* mais non *suffisantes* pour observer l'apparition des comportements décrits (oscillations soutenues et multistationnarité). Plusieurs travaux permettent d'analyser la fonctionnalité des circuits, c'est-à-dire leur capacité à effectivement « générer » ces comportements, et donc de rendre ces conditions susnommées nécessaires. Partant d'un modèle de Thomas donné, il est possible de représenter sa paramétrisation à l'aide de diagrammes de décision binaires ou multivalués (Bryant, 1986; Srinivasan, Ham, Malik & Brayton, 1990) afin d'en déduire l'ensemble des états stables mais aussi d'analyser la fonctionnalité des circuits en effectuant des produits de diagrammes de décision (Naldi, Thieffry & Chaouiya, 2007). Les différentes façons dont les oscillations soutenues et la multistationnarité sont « générées » ont aussi été étudiées, ce qui a permis de préciser ces conditions (Comet, Noual, Richard, Aracena, Calzone, Demongeot, Kaufman, Naldi, Snoussi & Thieffry, 2013).

Analyse de la dynamique

Afin d'obtenir des résultats plus précis sur le comportement d'un modèle, comme l'activation d'un composant ou la présence d'oscillations sur une partie donnée du modèle, ce que les méthodes décrites précédemment ne permettent pas d'obtenir, il peut devenir nécessaire de faire appel à des techniques de *model checking* plus poussées (voir p. ex. Richard et al., 2006; Gallet, Manceny, Le Gall & Ballarini, 2014). Ces méthodes permettent généralement une analyse exhaustive de la dynamique, tout en devant faire face à l'explosion combinatoire typique de ce type d'analyse, qui limite grandement la taille des modèles étudiés. De telles analyses ont été théorisées par Bernot, Comet, Richard & Guespin (2004) et se basent sur l'expression de propriétés exprimées en logiques temporelles CTL (Clarke & Emerson, 1982) ou LTL (Pnueli, 1977). Elles possèdent l'avantage d'être très complètes car la dynamique est entièrement explorée et l'expressivité des logiques temporelles permet d'exprimer beaucoup de propriétés intéressantes, mais au prix d'une complexité très importante. Afin de réduire cette complexité, Naldi, Remy, Thieffry & Chaouiya (2009) proposent d'effectuer de la réduction de modèles en éliminant certaine nœuds qui ont un rôle secondaire. Le modèle obtenu possède une taille inférieure et certaines bonnes propriété-

tés sont conservées, comme l'ensemble des points fixes et des attracteurs, au prix d'une dynamique parfois sous-approximée par rapport au modèle d'origine.

2.2 Les Frappes de Processus standards

Cette section définit les Frappes de Processus standards telles qu'elles ont été formalisées par Paulevé et al. (2011*a*). En tant que restrictions du π-calcul (Brand & Zafiropulo, 1983; Milner, 1989), les Frappes de Processus offrent une représentation discrète, asynchrone et indéterministe avec une définition atomique des interactions entre les différents composants du modèle. En cela, elles sont particulièrement adaptées aux représentations des réseaux de régulation biologique bien qu'elles soient assez générales pour potentiellement permettre des applications plus larges en informatique ou dans d'autres domaines.

Nous donnerons tout d'abord à la section 2.2 une définition des Frappes de Processus standards ainsi qu'un certain nombre de définitions formelles supplémentaires nécessaires pour la suite de ce manuscrit. Nous détaillons ensuite à la section 2.2.2 le mécanisme particulier des sortes coopératives tel qu'il avait été formalisé dans les travaux de Loïc Paulevé, et nous montrerons qu'il permet de représenter l'action conjointe de plusieurs composants, mais avec certains inconvénients difficiles à corriger dans cette version du formalisme. Enfin, nous aborderons brièvement les travaux précédents concernant l'analyse statique qui permet d'obtenir des résultats sur l'ensemble des états stables (section 2.2.3) et sur l'atteignabilité d'un niveau local au sein d'un modèle (section 2.2.4), et nous évoquerons les possibilités d'ajout de paramètres stochastiques (section 2.2.5) dans le but d'introduire une composante temporelle continue dans les Frappes de Processus.

2.2.1 Définition des Frappes de Processus standards

Les *Frappes de Processus standards* telles que données à la définition 2.8, aussi appelées *Process Hitting*, ou plus simplement *Frappes de Processus* dans ce chapitre, permettent une modélisation atomique et asynchrone des interactions entre composants. Un modèle de Frappes de Processus standards comporte un nombre fini de *sortes* généralement notées a, b, c... Celles-ci permettent de représenter les différentes entités du modèle, qu'il s'agisse de composants ayant une réalité biologique (gène, protéine...) ou d'entités nécessaires à la modélisation (comme les sortes coopératives qui seront décrites à la section 2.2.2). Chaque sorte contient plusieurs *processus*, qui représentent les différents niveaux d'expression discrets accessibles par la sorte, et qui sont notés a_i où a est le nom de la sorte et i l'indice du processus dans cette sorte. Un processus n'appartient qu'à une unique sorte. Chaque processus est dit *actif* (on écrira aussi : *présent*) s'il représente le niveau d'expression dans lequel doit se trouver sa sorte à un certain moment. Un *état* du modèle est donc décrit par l'ensemble des processus actifs à un instant donné, avec exactement un processus actif par sorte — afin de ne pas sur-représenter ou sous-représenter le niveau d'expression courant d'une sorte.

La dynamique est introduite dans les Frappes de Processus par des *actions* qui permettent de modifier le processus actif d'une sorte, à la condition éventuelle qu'un processus donné d'une autre sorte soit présent. Une action consiste donc en un triplet de processus $a_i \to b_j \not\uparrow b_k$ qui se lit : « a_i frappe b_j pour le faire bondir en b_k », et qui signifie que si, dans un état donné, les processus a_i et b_j sont tous les deux actifs, alors il est possible

d'activer b_k (et de désactiver b_j) dans l'état suivant. Autrement dit, le processus actif de la sorte b peut *bondir* de b_j à b_k à condition que a_i soit actif ; lorsque cela arrive, on dit qu'on a *joué* l'action $a_i \to b_j \nearrow b_k$. Par convention, on contraint de plus que $b_j \neq b_k$ pour assurer que le jeu d'une action provoque bien le changement d'un processus actif. Il est aussi possible de définir une auto-action, où $a_i = b_j$ (et nécessairement $a = b$), qui permet de représenter le cas particulier où le processus b_j peut bondir en b_k sans autre condition.

Les Frappes de Processus sont conçues comme un formalisme à temps discret asynchrone, ce qui signifie que l'évolution d'un tel modèle est modélisée par une succession de pas de temps discrets qui représentent la succession des états du modèle, et exactement une action est jouée entre deux états successifs. Cela implique qu'un seul processus actif à la fois peut bondir entre deux pas de temps successifs, et donc qu'une seule sorte peut évoluer entre deux états. De plus, cela rend la dynamique Frappes de Processus indéterministe, car à tout état du modèle peuvent correspondre plusieurs états successeurs dans le cas où plusieurs actions peuvent y être jouées. Enfin, nous notons que si aucune action n'est jouable dans un état, alors celui-ci ne possède pas de successeur et le modèle ne peut plus évoluer.

Définition 2.8 (Frappes de Processus standards)**.** Les *Frappes de Processus standards* sont définies par un triplet $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$, où :

- $\Sigma \overset{def}{=} \{a, b, \ldots\}$ est l'ensemble fini et dénombrable des *sortes* ;
- $\mathcal{L} \overset{def}{=} \bigotimes_{a\in\Sigma} \mathcal{L}_a$ est l'ensemble fini des *états*, où $\mathcal{L}_a = \{a_0, \ldots, a_{l_a}\}$ est l'ensemble fini et dénombrable des *processus* de la sorte $a \in \Sigma$ et $l_a \in \mathbb{N}^*$, chaque processus appartenant à une unique sorte : $\forall (a_i\,;b_j) \in \mathcal{L}_a \times \mathcal{L}_b, a \neq b \Rightarrow a_i \neq b_j$;
- $\mathcal{H} \overset{def}{=} \{a_i \to b_j \nearrow b_k \mid (a\,;b) \in \Sigma\times\Sigma \wedge (a_i\,;b_j\,;b_k) \in \mathcal{L}_a \times \mathcal{L}_b \times \mathcal{L}_b \wedge b_j \neq b_k \wedge a = b \Rightarrow a_i = b_j\}$ est l'ensemble fini des *actions*.

On note $\mathbf{Proc} \overset{def}{=} \bigcup_{a\in\Sigma} \mathcal{L}_a$ l'ensemble de tous les processus. La sorte d'un processus a_i est donnée par $\mathsf{sorte}(a_i) = a$; on définit aussi l'ensemble des sortes d'une action ou d'un ensemble de processus par :

$$\forall h \in \mathcal{H}, \mathsf{sortes}(h) = \{\mathsf{sorte}(\mathsf{frappeur}(h)), \mathsf{sorte}(\mathsf{cible}(h))\}$$
$$\forall A \subset \mathbf{Proc}, \mathsf{sortes}(A) = \{\mathsf{sorte}(p) \mid p \in A\}$$

Étant donné un état $s \in \mathcal{L}$, le processus de la sorte $a \in \Sigma$ présent dans s est donné par $s[a]$, c'est-à-dire la coordonnée correspondant à a dans l'état s. Si $a_i \in \mathcal{L}_a$, nous définissons la notation : $a_i \in s \overset{def}{\Leftrightarrow} s[a] = a_i$; par extension, si $ps \subset \mathbf{Proc}$, on écrit alors : $ps \subseteq s \overset{def}{\Leftrightarrow} \forall p \in ps, p \in s$. Pour toute action $h = a_i \to b_j \nearrow b_k \in \mathcal{H}$, a_i est appelé le *frappeur*, b_j la *cible* et b_k le *bond* de h, et on note : $\mathsf{frappeur}(h) = a_i$, $\mathsf{cible}(h) = b_j$ et $\mathsf{bond}(h) = b_k$.

Exemple. La figure 2.3 illustre une représentation possible des Frappes de Processus standards. Le modèle $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ représenté comporte trois sortes : $\Sigma = \{a, c, f\}$. Chaque sorte comporte exactement deux processus :

$$\mathcal{L}_a = \{a_0, a_1\} \quad ; \quad \mathcal{L}_b = \{b_0, b_1\} \quad ; \quad \mathcal{L}_f = \{f_0, f_1\} \quad .$$

On peut notamment en déduire le nombre total d'états du système : $|\mathcal{L}| = |\mathcal{L}_a|\cdot|\mathcal{L}_c|\cdot|\mathcal{L}_f| = 2^3 = 8$. Cette grandeur n'est cependant donnée qu'à titre indicatif, car nous évitons de

construire explicitement l'espace des états dont la taille est exponentielle dans le nombre de sortes et de processus du modèle. Enfin, le modèle étudié comporte 7 actions :

$$\begin{aligned}\mathcal{H} = \{ \quad & c_1 \to a_1 \Rsh a_0 \quad , & & c_0 \to a_0 \Rsh a_1 \quad , \\ & c_1 \to c_1 \Rsh c_0 \quad , & & f_1 \to c_0 \Rsh c_1 \quad , \\ & f_1 \to a_0 \Rsh a_1 \quad , & & f_0 \to c_1 \Rsh c_0 \quad , \\ & f_1 \to f_1 \Rsh f_0 & & \}\end{aligned}$$

Dans cet exemple, les Frappes de Processus représentent un modèle simplifié du mécanisme de segmentation des métazoaires qui permet par exemple de décrire la production de rayures chez les drosophiles. Il a été originellement établi *in silico* par François, Hakim & Siggia (2007) à l'aide d'un formalisme à base d'équations différentielles, et le modèle présenté ici est inspiré du modèle proposé par Paulevé et al. (2011*a*).

Les trois sortes a, b et c de ce modèle représentent différents gènes du système, que nous qualifierons d'*actifs* dans le reste de ce document lorsqu'ils seront à l'état 1 (ce qui ne doit pas être confondu avec le caractère *actif* d'un processus, défini en page 24). La production de pigment est déclenchée par le produit du gène a, et une succession d'activations et de désactivations de celui-ci permet donc de produire des rayures. Pour que celles-ci soient régulières, il est donc nécessaire que la durée d'activation de a soit constante, et que la durée entre deux activations le soit aussi. Ce mécanisme est réglé par le gène c qui inhibe à la fois le gène a à intervalles réguliers, et s'inhibe lui-même afin d'avoir le rôle d'une horloge. Enfin, le procédé complet est dirigé par le gène f qui, lorsqu'il est actif, permet la progression d'un front au niveau duquel les pigments sont déposés. Ce gène peut aussi s'auto-inhiber après une certaine période, faisant cesser l'oscillation de l'horloge et ainsi la production de rayures.

Les séquences d'actions ont un rôle particulier pour les Frappes de Processus. Elles permettent notamment d'abstraire une dynamique locale en se concentrant sur la conséquence plutôt que sur la cause, et seront notamment utiles pour la méthode d'analyse statique développée au chapitre 4. Pour toute séquence d'actions A, on note $\mathsf{sortes}(A)$ l'ensemble des sortes dont au moins un processus figure dans A en tant que frappeur, cible ou bond d'une action. De plus, pour toute sorte a, on note $\mathsf{premier}_a(A)$ le premier processus de a référencé dans A, en tant que frappeur ou en tant que cible, et $\mathsf{dernier}_a(A)$ le dernier, en tant que frappeur ou en tant que bond. On note en conséquence $\sup A$ l'ensemble de tous ces premiers processus, et $\mathsf{fin}(A)$ l'ensemble de tous ces derniers processus.

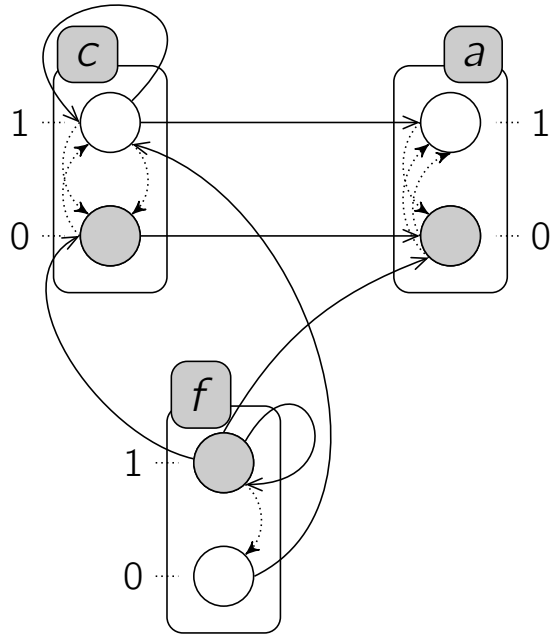


Figure 2.3 – Un exemple de Frappes de Processus standards. Les sortes sont représentées par des rectangles arrondis contenant des cercles représentant les processus. Ainsi, le processus a_1 est représenté par le cercle marqué « 1 » dans le rectangle étiqueté « a », etc. Chaque action est de plus symbolisée par un couple de flèches, l'une en trait plein et l'autre en pointillés ; par exemple, l'action $c_1 \to a_1 \nearrow a_0$ est représentée par une flèche pleine entre les processus c_1 et a_1 suivie d'une flèche en pointillés entre a_1 et a_0. Enfin, les processus grisés représentent un état possible pour ces Frappes de Processus : $\langle a_0, c_0, f_1 \rangle$, qui peut aussi faire office d'état initial pour ce modèle.

Définition 2.9 (premier, dernier, support et fin)**.**

$$\mathsf{premier}_a(A) = \begin{cases} \varnothing & \text{si } a \notin \mathsf{sortes}(A) \\ \mathsf{frappeur}(A_m) & \text{si } m = \min\{n \in \mathbb{I}^A \mid a \in \mathsf{sortes}(A_n)\} \\ & \qquad \wedge\ \mathsf{sorte}(\mathsf{frappeur}(A_m)) = a \\ \mathsf{cible}(A_m) & \text{sinon si } m = \min\{n \in \mathbb{I}^A \mid a \in \mathsf{sortes}(A_n)\} \\ & \qquad \wedge\ \mathsf{sorte}(\mathsf{cible}(A_m)) = a \end{cases}$$

$$\mathsf{dernier}_a(A) = \begin{cases} \varnothing & \text{si } a \notin \mathsf{sortes}(A) \\ \mathsf{bond}(A_m) & \text{si } m = \max\{n \in \mathbb{I}^A \mid a \in \mathsf{sortes}(A_n)\} \\ & \qquad \wedge\ \mathsf{sorte}(\mathsf{bond}(A_m)) = a \\ \mathsf{frappeur}(A_m) & \text{sinon si } m = \max\{n \in \mathbb{I}^A \mid a \in \mathsf{sortes}(A_n)\} \\ & \qquad \wedge\ \mathsf{sorte}(\mathsf{frappeur}(A_m)) = a \end{cases}$$

$$\mathsf{support}(A) = \{p \in \mathbf{Proc} \mid \mathsf{sorte}(p) \in \mathsf{sortes}(A) \wedge p = \mathsf{premier}_{\mathsf{sorte}(p)}(\delta)\}$$
$$\mathsf{fin}(A) = \{p \in \mathbf{Proc} \mid \mathsf{sorte}(p) \in \mathsf{sortes}(A) \wedge p = \mathsf{dernier}_{\mathsf{sorte}(p)}(\delta)\}$$

La définition 2.10 établit la notion de sous-état sur un ensemble de sortes, c'est-à-

dire un ensemble de processus qui sont deux à deux de sortes différentes, ce qui permet de ne considérer qu'une partie d'un état complet. Nous notons $\mathcal{L}^\Diamond$ l'ensemble de tous les sous-états et nous constatons qu'un état est *a fortiori* un sous-état : $\mathcal{L} \subset \mathcal{L}^\Diamond$. Nous notons de plus $\mathbf{Proc}^\Diamond$ l'ensemble des sous-états désordonnés, c'est-à-dire dont l'ordre entre les sortes a été oublié. Le recouvrement d'un état s par un processus a_i est formalisé à la définition 2.11 par un état identique à s, sauf pour le processus de a qui a été remplacé par a_i, ce qui permettra de définir la dynamique des Frappes de Processus à la définition 2.14. La définition de recouvrement est aussi étendue à un sous-état désordonné, autrement dit, à un ensemble de processus contenant au plus un processus par sorte.

Définition 2.10 (Sous-états ($\mathcal{L}^\Diamond$))**.** Si $S \subset \Sigma$ est un ensemble de sortes, un sous-état sur S est un élément de : $\mathcal{L}_S^\Diamond \stackrel{def}{=} \bigotimes_{a \in S} \mathcal{L}_a$. L'ensemble de tous les sous-états est noté : $\mathcal{L}^\Diamond \stackrel{def}{=} \bigcup_{S \in \wp(\Sigma)} \mathcal{L}_S^\Diamond$.
De plus, si $\sigma \in \mathcal{L}^\Diamond$ et $s \in \mathcal{L}$, on note alors :

$$\sigma \subseteq s \stackrel{def}{\Leftrightarrow} \forall a_i \in \mathbf{Proc}, a_i \in \sigma \Rightarrow a_i \in s \ .$$

Enfin, si $S \subset \Sigma$, on note : $\mathbf{Proc}_S^\Diamond = \{\widetilde{ps} \subset \mathbf{Proc} \mid ps \in \mathcal{L}_S^\Diamond\}$ et $\mathbf{Proc}^\Diamond = \{\widetilde{ps} \subset \mathbf{Proc} \mid ps \in \mathcal{L}^\Diamond\}$.

Définition 2.11 (Recouvrement ($\Cap : \mathcal{L} \times \mathbf{Proc} \to \mathcal{L}$))**.** Étant donné un état $s \in \mathcal{L}$ et un processus $a_i \in \mathbf{Proc}$, $(s \Cap a_i)$ est l'état défini par : $(s \Cap a_i)[a] = a_i \wedge \forall b \neq a, (s \Cap a_i)[b] = s[b]$. On étend de plus cette définition à un ensemble de processus par le recouvrement de l'état par chaque processus, à condition que les processus de l'ensemble soient tous de sortes différentes : $\forall ps \in \mathbf{Proc}^\Diamond, s \Cap ps = s \bigCap_{a_i \in ps} a_i$.

Nous définissons dans la suite les outils nécessaires à la dynamique des Frappes de Processus. Pour cela, nous introduisons la notion de propriété de jouabilité à la définition 2.12, qui est semblable à une formule booléenne dont les atomes sont des processus dans **Proc**. Le langage des propriétés de jouabilité permet de décrire la présence d'une configuration de processus actifs dans un état donné, ce qui permet donc notamment de décrire en termes formels la jouabilité d'une action. Cela est mis en pratique pour les Frappes de Processus standards à la définition 2.13 où nous définissons l'*opérateur de jouabilité* de ce formalisme, qui est une formule associant à chaque action sa propriété de jouabilité propre. Pour finir, la dynamique des Frappes de Processus est donnée à la définition 2.14 à partir de celle d'opérateur de jouabilité. Les définitions de propriété de jouabilité (définition 2.12) et de dynamique des Frappes de Processus (définition 2.14) sont volontairement assez générales pour pouvoir être réutilisées au chapitre 3 pour définir la sémantique d'autres formalismes de Frappes de Processus.

Définition 2.12 (Propriété de jouabilité (F))**.** Une *propriété de jouabilité* est un élément du langage F défini inductivement par :

- $\top$ et $\bot$ appartiennent à F ;
- si $a \in \Sigma$ et $a_i \in \mathcal{L}_a$, alors a_i appartient à F et est appelé un *atome* ;
- si $P \in F$ et $Q \in F$, alors $\neg P$, $P \wedge Q$ et $P \vee Q$ appartiennent à F.

Si $P \in F$ est une propriété de jouabilité et $\sigma \in \mathcal{L}^{\Diamond}$ est un sous-état, on note $[P](\sigma)$ l'*évaluation* de P dans σ :

- si $P = a_i \in \mathcal{L}_a$ est un atome, avec $a \in \Sigma$, alors $[a_i](\sigma)$ est vraie si et seulement si $a_i \in \sigma$;
- si P n'est pas un atome, alors $[P](\sigma)$ est vraie si et seulement si on peut l'évaluer récursivement comme vraie en utilisant la sémantique habituelle des opérateurs $\neg$, $\wedge$ et $\vee$ et des constantes $\top$ et $\bot$.

Une fonction $F : \mathcal{H} \to F$ associant à toute action une propriété de jouabilité est appelée un *opérateur de jouabilité*.

Étant donné que ce langage n'utilise que des opérateur logiques classiques, les propriétés de la logique booléenne sont applicables aux propriétés de jouabilité, à savoir celles concernant la distributivité, l'associativité et la commutativité, ainsi que les lois de De Morgan concernant la négation.

Il en résulte notamment la propriété suivante, permettant d'évaluer la négation d'un atome, et qui dérive naturellement du fait que si un processus n'est pas actif dans un état donné, cela signifie alors qu'un autre processus de la même sorte l'est :

$$\forall a \in \Sigma, \forall a_i \in \mathcal{L}_a, \forall \sigma \in \mathcal{L}^{\Diamond}, [\neg a_i](\sigma) \Leftrightarrow \left[\bigvee_{\substack{a_j \in \mathcal{L}_a \\ a_j \neq a_i}} a_j \right](\sigma)$$

L'opérateur de jouabilité F donné à la définition 2.13 est propre aux Frappes de Processus standards. En revanche, la dynamique donnée à la définition 2.14 est générale à toutes les Frappes de Processus, et peut donc à fortiori être utilisée avec l'opérateur F des Frappes de Processus standards.

Définition 2.13 (Opérateur de jouabilité ($F : \mathcal{H} \to F$))**.** L'opérateur de jouabilité des Frappes de Processus est défini par :

$$\forall h \in \mathcal{H}, F(h) \equiv \mathsf{frappeur}(h) \wedge \mathsf{cible}(h) \ .$$

Définition 2.14 (Dynamique des Frappes de Processus ($\to_{\mathcal{PH}}$))**.** Une action $h \in \mathcal{H}$ est dite *jouable* dans l'état $s \in \mathcal{L}$ si et seulement si : $[F(h)](s)$. Dans ce cas, $(s \cdot h)$ est l'état résultant du jeu de l'action h dans s, et on le définit par : $(s \cdot h) = s \Cap \mathsf{bond}(h)$. De plus, on note alors : $s \to_{\mathcal{PH}} (s \cdot h)$.
Si $s \in \mathcal{L}$, un *scénario* δ dans s est une séquence d'actions de $\mathcal{H}$ qui peuvent être jouées successivement depuis s. L'ensemble de tous les scénarios dans s est noté $\mathbf{Sce}(s)$.

Remarque. Les Frappes de Processus standards possèdent une dynamique totalement asynchrone : entre deux états, une unique action peut être jouée. Ce choix de conception est largement inspiré du modèle de Thomas présenté à la section 2.1.1 en page 16, dont la

dynamique est aussi totalement asynchrone. Biologiquement, une telle hypothèse est cohérent avec le fait que la probabilité que deux composants franchissent simultanément un seuil d'expression est très faible. Cependant, cette sémantique asynchrone permet aussi de simplifier la dynamique des Frappes de Processus standards en assurant que deux états successifs ne varient que d'un seul processus (en fait : exactement d'un processus). Cela a notamment permis le développement de l'analyse statique évoquée plus loin, à la section 2.2.4.

Exemple. La dynamique des Frappes de Processus standards de la figure 2.3 est représentée à la figure 2.4. On peut notamment y observer le comportement stationnaire normal du modèle qui consiste en une oscillation alternée des processus actifs des sortes a et c :

$$\langle a_0, c_0, f_1\rangle \to_{\mathcal{PH}} \langle a_1, c_0, f_1\rangle \to_{\mathcal{PH}} \langle a_1, c_1, f_1\rangle \to_{\mathcal{PH}} \langle a_0, c_1, f_1\rangle \to_{\mathcal{PH}} \langle a_0, c_0, f_1\rangle \to_{\mathcal{PH}} \dots$$

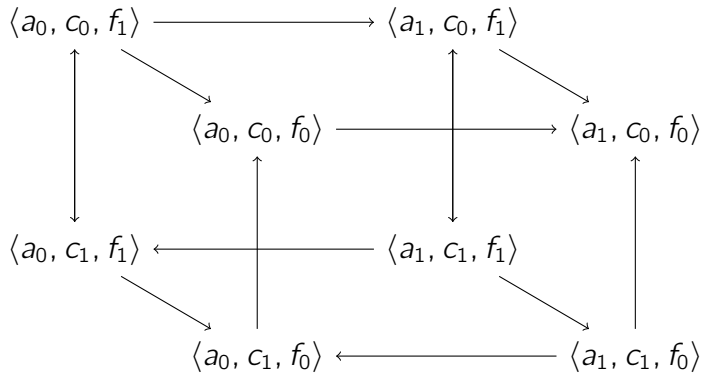


Figure 2.4 – Représentation de la dynamique du modèle de Frappes de Processus standards de la figure 2.3. Chaque état est représenté par un triplet $\langle a_i, c_j, f_k\rangle$ où i, j et k représentent respectivement le niveau d'expression de a, c et f. Une transition entre deux états est représentée par une flèche.

2.2.2 Utilisation des sortes coopératives

L'une des questions qui se posent face à un formalisme totalement asynchrone comme les Frappes de Processus standards est la représentation des coopérations entre les différents composants. En effet, le bond d'un processus dans un modèle de Frappes de Processus standards ne peut se faire que par le jeu d'une action, elle-même déclenchée par la présence d'au plus deux processus : le frappeur et la cible (c'est-à-dire le processus qui va bondir). Il n'est donc pas possible de conditionner le bond d'un processus par la présence de plusieurs processus de sortes différentes de celle de la cible. En effet, si on souhaite par exemple représenter l'activation d'un processus c_1 (d'une sorte c) uniquement lorsque deux autres processus a_1 et b_1 (de deux autres sortes a et b) sont actifs, il n'est pas suffisant d'ajouter deux actions $a_1 \to c_0 \Rsh c_1$ et $b_1 \to c_0 \Rsh c_1$, car celles-ci permettent d'activer c_1 à la condition que a_1 soit actif ou que b_1 soit actif : il s'agit bien de deux interactions distinctes et non d'une coopération.

Exemple. Le modèle de Frappes de Processus de la figure 2.3 représente le mécanisme de segmentation métazoaire évoqué à la page 25. Dans ce modèle, la production de pigment

devrait uniquement être possible à la condition suivante : « f est actif et c n'est pas actif ». Or dans l'état courant du modèle, la désactivation du gène f n'empêche pas la production de pigment, car depuis tout état contenant f_0, il est toujours possible d'activer a à l'aide des actions $f_0 \to c_1 \Rsh c_0$ et $c_0 \to a_0 \Rsh a_1$.

Afin de représenter la coopération entre composants, et donc le raffinement de la dynamique des modèles, Paulevé et al. (2011*a*) ont proposé d'ajouter des sortes particulières appelées *sortes coopératives*, qui servent exclusivement à la modélisation. Une sorte coopérative permet de représenter l'état conjoint de plusieurs sortes dans le modèle. Pour cela, à chaque processus de la sorte coopérative correspond un sous-état des sortes qu'elle *représente*. Ainsi, il est possible de représenter les différents états combinés d'un ensemble de sortes, afin de ne jouer une action que dans une configuration particulière. Ces sortes ont l'avantage d'être des sortes standards, et donc de ne pas nécessiter d'enrichissement particulier de la sémantique. De plus, leur utilisation n'impacte pas les méthodes d'analyse de la dynamique développées : en effet, ces méthodes sont principalement impactées par le nombre de processus dans chaque sorte, et non le nombre total de sortes. Ainsi, à condition de limiter le nombre de processus dans les sortes coopératives, comme expliqué à la fin de cette section, il est possible de les utiliser en maintenant de bonnes performances d'analyse.

Exemple. Les Frappes de Processus standards de la figure 2.5 reprennent les trois sortes f, c et a du modèle de la figure 2.3 et comprennent en plus une sorte coopérative fc permettant de détecter la présence simultanée de f_1 et c_1. Les processus de fc décrivent les combinaisons possibles des états de f et de c : fc_{00} correspond à f_0 et c_0, fc_{01} correspond à f_0 et c_1, etc. Les actions en amont de cette sorte coopérative permettent la mettre à jour, c'est-à-dire de changer son processus actif en fonction des évolutions du processus actif de f et c. Par exemple, si f_1 est actif, les deux actions $f_1 \to fc_{00} \Rsh fc_{10}$ et $f_1 \to fc_{01} \Rsh fc_{11}$ effectuent cette mise à jour en faisant bondir le processus actif de fc depuis un processus représentant la présence de f_0 (fc_{00} ou fc_{01}) vers le processus correspondant représentant la présence de f_1 (respectivement fc_{10} ou fc_{11}). Son action en aval, $fc_{11} \to c_0 \Rsh c_1$, joue alors le rôle d'une coopération entre f_1 et c_0 pour frapper a_0 et le faire bondir en a_1.

Exemple. Afin de pallier partiellement l'absence de coopération entre les sortes f et c dans le modèle de la figure 2.3, il est possible d'intégrer la sorte coopérative décrite à la figure 2.5. La figure 2.6 propose un modèle corrigé de cette manière, avec une sorte coopérative fc permettant de détecter la présence de f_1 et c_0. Les deux actions $c_0 \to a_0 \Rsh a_1$ et $c_0 \to a_0 \Rsh a_1$ sont alors remplacées par une action $fc_{10} \to a_0 \Rsh a_1$ afin d'avoir une véritable coopération entre ces deux processus pour activer a.

Le sortes coopératives possèdent cependant un inconvénient, qui est lié au fait que les actions sont totalement indépendantes car le formalisme des Frappes de Processus standards est totalement asynchrone et indéterministe. En effet, les sortes coopératives ne sont pas nécessairement mises à jour immédiatement après un bond du processus actif d'une des sortes qu'elles représentent. Il peut ainsi exister un « décalage temporel » entre le changement de processus actif d'une sorte et la mise à jour des sortes coopératives. Ce décalage temporel permet alors de jouer des actions modélisant des coopérations dans des états où la coopération n'est plus possible, car même si l'un des processus modélisant la coopération a bondi, la sorte coopérative peut ne pas en avoir fait de même. Mais ce décalage peut aussi aboutir à des comportements indésirables. En effet, le processus actif

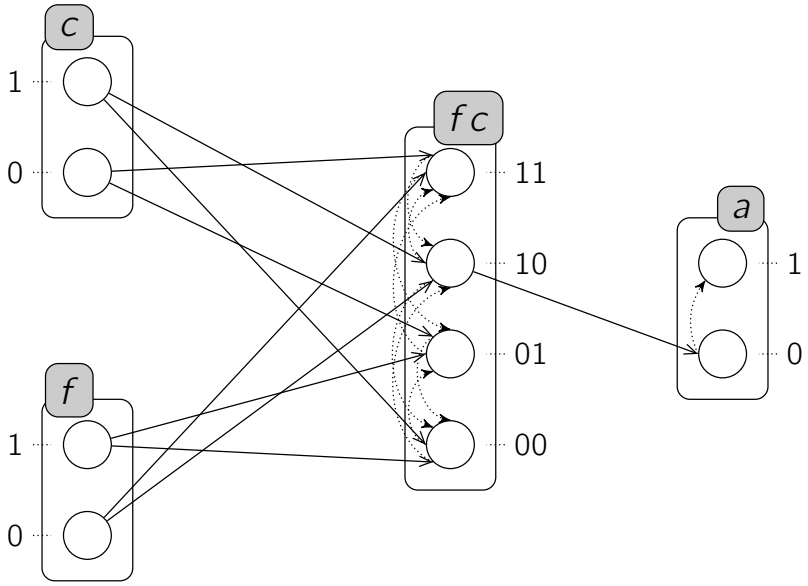


Figure 2.5 – Un exemple de Frappes de Processus standards avec une sorte coopérative $f\,c$. L'action $f\,c_{10} \to a_0 \Rsh a_1$ modélise une coopération entre f_1 et c_0 pour faire bondir le processus actif de a.

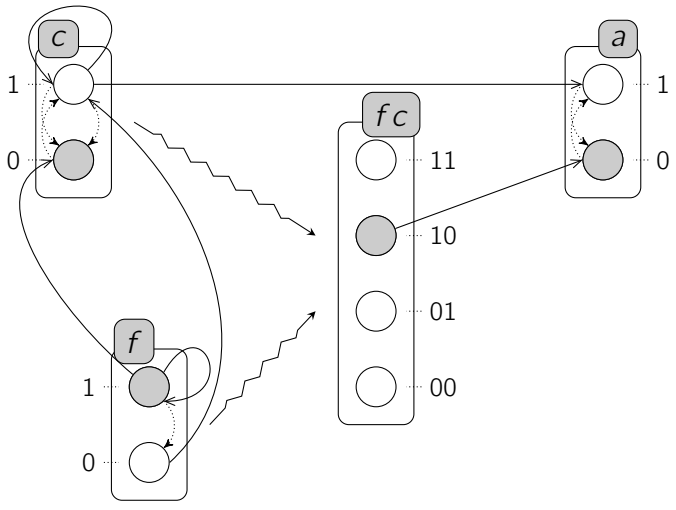


Figure 2.6 – Amélioration du modèle de Frappes de Processus de la figure 2.3 à l'aide de la sorte coopérative $f\,c$. Les processus de cette sorte représentent les différents sous-états formés par les deux sortes f et c. Ainsi, $f\,c_{00}$ représente le fait que f_0 et c_0 sont actifs, etc. Les actions permettant la mise à jour de cette sorte coopérative n'ont pas été représentées explicitement mais sont symbolisées par les deux flèches en zigzag provenant de f et c.

d'une sorte coopérative ne correspond de fait pas à l'état courant des sortes représentées, mais uniquement à une combinaison d'états passés. Il est alors possible d'activer un processus de la sorte coopérative correspondant à un sous-état artificiel, c'est -à-dire non accessible aux sortes représentées. Ce mécanisme sera détaillé notamment au chapitre 4.

Exemple. Malgré l'ajout d'une sorte coopérative $f\,c$ dans le modèle de la figure 2.3, il faut noter que le comportement désiré n'est pas exactement représenté. En effet, l'ajout de cette sorte coopérative devait permettre d'éviter toute activation de a lorsque f devenait inactif, en permettant par exemple de jouer ce type de scénario depuis l'état initial $\langle a_0, c_0, f_1, f\,c_{10}\rangle$:

$$f_1 \to f_1 \Rsh f_0 :: f_0 \to f\,c_{10} \Rsh f\,c_{00}$$

après lequel il n'est plus possible d'atteindre un état où a_1 est actif.

Cependant, il se trouve qu'il existe encore un cas particulier où a peut être activé malgré la présence de f_0. Ce cas particulier relève du comportement mis en valeur ci-dessus, où une action a pour frappeur un processus de sorte coopérative qui ne devrait pas être actif si celle-ci avait été mise à jour. Il s'observe par exemple en jouant le scénario suivant depuis l'état initial $\langle a_0, c_0, f_1, f\,c_{10}\rangle$:

$$f_1 \to f_1 \Rsh f_0 :: f\,c_{10} \to a_0 \Rsh a_1 \ ,$$

ce qui est possible parce que la sorte $f\,c$ n'a pas été mise à jour avant le jeu de l'action $f\,c_{10} \to a_0 \Rsh a_1$.

Nous notons pour terminer qu'il existe deux manières de diminuer le nombre de processus dans une sorte coopérative.

On peut tout d'abord se limiter à deux états par sorte représentée : un état « bon » et un état « mauvais », même si les sortes représentées possèdent plus de deux processus. Cela permet de limiter le nombre de processus de la sorte coopérative à $2^{|A|}$, où A est l'ensemble des sortes à représenter, bien que le nombre de processus dépende toujours exponentiellement de la taille de A.

Il est aussi possible de factoriser les sortes coopératives qui représentent trois sortes ou plus. Pour cela, il suffit par exemple de créer une sorte coopérative intermédiaire entre deux des sortes représentées, puis une deuxième sorte coopérative entre la troisième sorte représentée et cette nouvelle sorte. Ainsi, trois sortes a, b et c peuvent être représentées soit par une unique sorte coopérative abc, soit par deux sortes coopératives ab et abc, la seconde représentant en fait les sortes ab et c (et non a, b et c directement). Le nombre de processus requis devient alors polynomial dans la taille de l'ensemble des sortes à représenter. En conjonction avec la méthode précédente, le nombre de processus requis devient effectivement $4 \cdot (|A| - 1)$.

2.2.3 Recherche de points fixes

Les Frappes de Processus standards bénéficient d'outils puissants permettant la recherche de points fixes, c'est-à-dire d'états dans lesquels le modèle ne peut plus évoluer car aucune action n'est jouable. Nous rappelons ici les méthodes de recherche de points fixes développées par Paulevé et al. (2011*a*) et qui reposent sur la recherche de n-cliques.

La recherche de points fixes peut apporter des informations fondamentales dans l'étude des réseaux de régulation biologique. Un point fixe représente généralement un état stable pour le système — tout du moins du point de vue du modèle. La détection de plusieurs

points fixes, par exemple, met en valeur l'existence d'un embranchement dans la dynamique du système, et le point fixe atteint dépendra du chemin suivi dans le graphe des états.

La recherche de points fixes dans des Frappes de Processus standards $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ peut se ramener à une recherche de $|\Sigma|$-cliques dans un graphe particulier construit à partir de $\mathcal{PH}$ et appelé *graphe sans-frappe*. L'idée principale de cette recherche repose sur la causalité restreinte des actions, qui implique que si les processus frappeur et cible d'une action sont tous deux actifs dans un état, alors cette action est jouable. Ainsi, les points fixes sont les actions dans lesquels il n'existe pas de tel couple de processus actifs, et donc pas d'action jouable. La construction du graphe sans-frappe repose sur cette constatation : ses nœuds sont les processus du modèle $\mathcal{PH}$ d'origine qui ne sont pas frappés par une auto-action, et deux nœuds ne sont reliés par une arête (non orientée) que si les deux processus correspondants ne sont *pas* reliés par une action (définition 2.15). La division du graphe en sortes en fait un graphe n-parti (définition 2.16) avec $n \leq |\Sigma|$ le nombre de sortes ayant au moins un processus non frappé par une auto-action. Le théorème 2.1 stipule alors que les points fixes de $\mathcal{PH}$ sont exactement les $|\Sigma|$-cliques de son graphe sans-frappe, c'est-à-dire les ensemble de nœuds tous connectés entre eux, avec exactement un nœud par partie (définition 2.17).

Définition 2.15. Soient $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ des Frappes de Processus standards. Le *graphe sans-frappe* de $\mathcal{PH}$ est le graphe non orienté (V, E) où :

$$V \stackrel{def}{=} \bigcup_{a\in\Sigma} \{a_i \in \mathcal{L}_a \mid \forall a_k \in \mathcal{L}_a, \nexists a_i \to a_i \Rsh a_k \in \mathcal{H}\}$$

$$E \stackrel{def}{=} \{\{a_i, b_j\} \in V \times V \mid a \neq b, \forall b_k \in \mathcal{L}_b, \nexists a_i \to b_j \Rsh b_k \in \mathcal{H} \wedge \forall a_l \in \mathcal{L}_a, \nexists b_j \to a_i \Rsh a_l \in \mathcal{H}\}$$

Définition 2.16. Si $n \in \mathbb{N}$, un graphe (V, E) est *n-parti* si et seulement si :

- $V = \bigcup_{k\in[\![1;n]\!]} V_k$,
- $\forall k, k' \in [\![1\,;n]\!], k \neq k' \Rightarrow V_k \cap V_{k'} = \emptyset$,
- $\forall \{a_i, b_j\} \in E, \exists k, k' \in [\![1\,;n]\!], k \neq k' \wedge a_i \in V_k \wedge b_j \in V_{k'}$.

Définition 2.17. Soit (V, E) un graphe non orienté. Un sous-ensemble $C \subseteq V$ de ses nœuds est une $|C|$-clique si et seulement si : $\forall (a_i, b_j) \in C \times C, \{a_i, b_j\} \in E$.

Théorème 2.1. *Les points fixes des Frappes de Processus standards $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ sont exactement les $|\Sigma|$-cliques de son graphe sans-frappe.*

Exemple. Les Frappes de Processus standards de la figure 2.3 contiennent uniquement le point fixe suivant : $\{a_1, c_0, f_0\}$, tandis que le modèle de la figure 2.6 contiennent les deux points fixes suivants : $\{a_0, c_0, f_0, f\,c_{00}\}$ et $\{a_1, c_0, f_0, f\,c_{00}\}$.

2.2.4 Analyse statique pour le calcul d'atteignabilité

Nous présentons brièvement et informellement dans cette section l'analyse statique par interprétation abstraite développée par Paulevé et al. (2012) et inspirée des méthodes d'interprétation abstraite de programmes (Cousot & Cousot, 1977). Celle-ci permet de répondre à des questions d'atteignabilité, c'est-à-dire déterminer s'il est possible d'activer

un processus depuis un état initial donné. Sa complexité polynomiale dans la taille du modèle permet une analyse très efficace de réseaux de régulation de très grande taille modélisés en Frappes de Processus.

L'étude de la dynamique des réseaux de régulation biologique se heurte généralement à la taille des modèles, et à la complexité qu'elle engendre. Les *model checkers* traditionnels doivent en effet généralement calculer l'intégralité des comportements possibles de réseau, généralement sous la forme d'un graphe des états, afin de pouvoir produire des conclusions sur la dynamique (Richard et al., 2006). Si une telle approche a le mérite de l'exhaustivité, elle n'est cependant pas applicable à des modèles dépassant quelques dizaines de composants, et donc totalement exclue pour tout modèle de plus d'une centaine de composants.

La forme particulière des actions des Frappes de Processus standards permet cependant de contourner en partie ces problèmes de complexité en approchant la dynamique des modèles. L'analyse statique développée par Paulevé et al. (2012) utilise en effet cette propriété pour résoudre efficacement des problèmes d'atteignabilité, c'est-à-dire des problèmes qui s'énoncent de la forme suivante : « Est-il possible d'activer un processus depuis un état donné en jouant un certain nombre d'actions ? » Ou, autrement dit : « Étant donnés un état s et un processus a_i, existe-t-il un scénario $\delta \in \mathbf{Sce}(s)$ dans cet état tel que $a_i \in s \cdot \delta$? »

La méthode développée pour répondre à ce type de question se base sur deux constatations :

- on peut vérifier localement que le processus actif d'une sorte peut bondir d'un niveau vers un autre en n'observant que les actions frappant cette sorte ;
- le jeu d'une action est conditionnée par au plus un processus d'une autre sorte que la cible de cette action.

De la première constatation découle le fait que l'atteignabilité d'un processus peut être résolue localement, en observant les actions frappant une sorte donnée, ce qui simplifie la recherche. Une fois ce problème résolu localement pour une sorte, la deuxième constatation permet de déplacer ce problème à d'autres sortes afin dans le but d'activer les processus requis pour jouer chaque action nécessaire (c'est-à-dire les frappeurs de ces actions).

Il existe donc un lien de causalité récursif entre l'activation locale d'un processus, les actions requises pour le faire, et de nouveau l'activation des processus nécessaires pour jouer ces actions, etc. Afin de réduire la complexité de la méthode, une approximation de la dynamique est effectuée à chaque étape : pour chaque résolution d'atteignabilité locale au sein d'une sorte, un ensemble de processus requis appartenant à d'autres sortes est produit, mais l'ordre dans lequel ces processus sont nécessaires est abstrait. Cela permet donc de déplacer l'atteignabilité locale d'un processus à plusieurs atteignabilités indépendantes dans d'autres sortes.

Cette approche permet de décliner une approximation supérieure (sur-approximation) et une approximation inférieure (sous-approximation) de l'ensemble de toutes les dynamiques possibles du modèle. La sur-approximation consiste à ne pas s'intéresser à l'ordre dans lequel les processus requis sont activables — et donc à l'ordre dans lequel les actions résolvant l'atteignabilité locale sont jouables. Cela autorise effectivement davantage de comportements, car en pratique un processus peut ne pas être activable après certaines actions. La sous-approximation, à l'inverse, stipule que tous les processus requis doivent être activables dans tous les ordres possibles, bien qu'en pratique, tous les ordres ne soient pas intéressants pour la résolution.

Chacune de ces approximations est représentée à l'aide d'un *graphe de causalité locale*, qui est unique à chaque problème d'atteignabilité, et qui formalise les liens de causalité évoqués précédemment. Enfin, Paulevé et al. (2012) donnent une propriété qui, sous certaines conditions dépendant du graphe de causalité locale correspondant à la sous-approximation, stipule que le processus donné est atteignable depuis l'état donné ; de même, une propriété complémentaire est donnée pour le graphe de causalité locale correspondant à la sur-approximation, qui permet d'obtenir la conclusion inverse. Si aucune des deux propriétés n'est vraie, la méthode est dite non conclusive, et il est nécessaire de raffiner le problème ou le modèle.

Le calcul des deux graphes de causalité locales est polynomial dans la taille des Frappes de Processus standards sur lesquelles la méthode est appliquée, et la vérification des deux propriétés l'est dans la taille des graphes obtenus. Ainsi, cette méthode est plus efficace que les approches par force brute car elle évite l'explosion combinatoire propre à l'analyse de la dynamique. Son implémentation produit des résultats en quelques dixièmes de seconde sur des modèles de plusieurs centaines de composants, et s'avère toujours conclusive sur les exemples étudiés.

Une partie de cette analyse statique a par la suite été exploitée dans le but d'approximer efficacement des ensembles de coupes, c'est-à-dire des ensembles de processus nécessaires à une certaine atteignabilité (Paulevé, Andrieux & Koeppl, 2013). Son utilisation dans ce cadre s'est avérée efficace sur des modèles de plusieurs milliers de composants.

2.2.5 Paramètres et analyse stochastiques

(Paulevé et al., 2011*a*) ont aussi proposé un enrichissement des Frappes de Processus standards à l'aide de paramètres stochastiques, l'objectif étant d'intégrer des données temporelles continues dans les modèles. Cet enrichissement est directement inspiré du *pi*-calcul stochastique (Priami, 1995). Cependant, la loi exponentielle utilisée pour la loi stochastique étant jugée trop grossière, l'approche a été raffinée par l'introduction d'un paramètre supplémentaire permettant de réduire l'intervalle de tir (Paulevé, Magnin & Roux, 2011*b*).

L'introduction de données dans les Frappes de Processus standards consiste à affecter un couple de paramètres stochastiques $(r\,;sa) \in \mathbb{N} \times \mathbb{R}$ à chaque action. La probabilité de tirer une action à un instant donné (sur un axe de temps continu) suit alors une loi d'Erlang en fonction de ces deux paramètres, c'est-à-dire une somme de lois exponentielles. Le premier paramètre, appelé *taux*, indique le nombre de fois qu'une action peut être tirée par unité de temps. Le second paramètre est l'*absorption de stochasticité*, qui détermine le nombre de lois exponentielles sommées pour obtenir la distribution finale.

Tout couple de paramètres stochastiques $(r\,;sa)$ correspond à un intervalle de tir $[d\,;D]$, où $d, D \in \mathbb{R}$, pour un niveau de confiance $\alpha \in [0\,;1]$ donné, et inversement, qui peut être approximé (Paulevé, 2011, p. 72). Cette conversion permet de raisonner sur des intervalles de tirs plutôt qu'en termes de loi d'Erlang, ce qui permet notamment de définir des fenêtres de tir pour chaque action — une action devant être tirée dans sa fenêtre avec un niveau de confiance de α. Au niveau de l'intervalle de tir, les deux paramètres stochastiques ont un rôle particulier :

– Le taux détermine la distance de l'intervalle de tir par rapport à l'origine de l'axe du temps ; autrement dit, une action avec un taux élevé sera tirée plus rapidement après sensibilisation.

– Le facteur d'absorption de stochasticité détermine la taille de l'intervalle de tir. Ainsi, augmenter ce facteur réduit la quantité $D - d$.

Enfin, il est aussi possible d'assigner un taux *infini* à une action ($r = \infty$), ce qui a pour conséquence de rendre le tir de l'action instantané : l'action doit être jouée dès sa sensibilisation. Si plusieurs actions de taux infini sont sensibilisées en même temps, le non-déterminisme est la règle et elles peuvent donc être jouées dans n'importe quel ordre. Une action de taux infini ne possède pas de facteur d'absorption de stochasticité, car ce deuxième paramètre ne changerait rien à sa condition de tir.

Exemple. Définissons les fenêtres de tir donnés à la figure 2.7, inspirées de (Paulevé et al., 2011*a*). Nous pouvons enrichir les Frappes de Processus standards décrites par la figure 2.6 en page 32 à l'aide de ces paramètres, de la façon donnée à la figure 2.8. Cette affectation nous permet notamment d'obtenir un comportement oscillant pour les sortes a et c, et une auto-désactivation de f après plusieurs oscillations. Les paramètres du couple $\mathbf{c}$ ont en effet été conçus de façon à ce que l'arrêt de l'horloge, provoqué par la désactivation de f, survienne après la formation de 7 rayures.

Les modèles de Frappes de Processus standards comportant des paramètres stochastiques peuvent être simulés afin d'observer un exemple d'exécution du modèle étudié. Du fait de la nature indéterministe et probabiliste du modèle, plusieurs dynamiques sont possibles, et chaque simulation donne des résultats différents en termes de chemin d'exécution — ou, pour un même chemin d'exécution, en termes de temps de tir des actions. Il est donc possible d'effectuer un grand nombre de simulations pour obtenir des statistiques sur le comportement d'un modèle. Il est aussi possible de recourir à un *model checker* probabiliste comme PRISM, qui fournit des résultats formels sur la dynamique d'un modèle, et des probabilités d'atteignabilité. Cependant, une telle approche est très coûteuse en termes de temps de calcul et d'espace mémoire, ce qui empêche l'analyse de réseaux de grande taille.

	∞	**a**	**b**	**c**
$[d\,;D]$	$[0\,;0]$	$[0{,}742\,;1{,}29]$	$[1{,}48\,;2{,}59]$	$[23{,}9\,;35{,}4]$
$(r\,;sa)$	$r=\infty$	$(1\,;50)$	$(0{,}5\,;50)$	$(0{,}034\,;100)$

Figure 2.7 – Exemples d'intervalles de tir et de paramètres stochastiques.

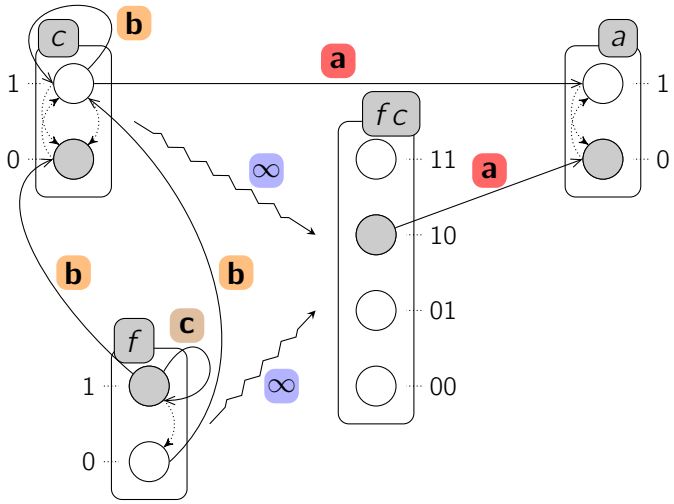


Figure 2.8 – Enrichissement des Frappes de Processus standards de la figure 2.6 à l'aide des paramètres stochastiques proposés à la figure 2.7. Chaque action est affectée à un couple de paramètres stochastiques repéré par une étiquette (∞, **a**, **b**, **c**) placée contre l'action. Les ensembles d'actions mettant à jour la sorte coopérative fc, représentées par des flèches en zigzag, sont toutes affectées, possèdent toutes un taux infini ($r=\infty$).

Chapitre 3

Enrichissement des Frappes de Processus pour l'aide à la modélisation

La sémantique standard des Frappes de Processus de la section 2.2 peut s'avérer insuffisante pour prendre en compte certaines informations connues sur le système étudié, comme des informations en terme de vitesse de réaction. De plus, certains comportements non désirés apparaissent dans les Frappes de Processus standards dès que l'on cherche à synchroniser plusieurs processus. Ce chapitre propose d'étendre sa sémantique afin de pallier ces problèmes en enrichissant les modèles sur deux axes :

- la préemption entre actions, qui permet d'empêcher le jeu d'une action sous certaines conditions,
- la simultanéité d'actions, qui permet de forcer le jeu simultané de plusieurs actions.

Ces deux axes ont pour but de permettre ou de faciliter l'intégration de telles informations au sein des modèles.
Du premier axe découlent deux extensions aux Frappes de Processus prenant la forme d'arcs neutralisants, qui modélisent la préemption d'une action par une autre, et de classes de priorités, qui modélisent la préemption d'un ensemble d'actions par un autre. Le second axe conduit à une troisième extension, reposant sur la notion d'actions plurielles. Après les avoir définies dans ce chapitre, nous montrons que ces trois modélisations sont (faiblement) bisimilaires. De plus, il est à noter que l'analyse statique développée au chapitre 4 peut s'appliquer à toutes ces extensions, moyennant une traduction.

L'extension des Frappes de Processus sous forme de classes de priorités a été publiée dans (Folschette, Paulevé, Magnin & Roux, 2013).

Nous présentons dans ce chapitre les trois sémantiques de Frappes de Processus développées pour enrichir l'expressivité de ce formalisme. Nous les définissons et discutons

de leurs avantages en termes de modélisation pour les réseaux de régulation biologique ou les réseaux de réactions biochimiques. De plus, nous traçons des liens formels entre ces différents formalismes afin de mieux comprendre leur complémentarité et d'offrir une bonne souplesse de représentation et d'analyse.

Les outils que nous proposons dans le présent chapitre permettent d'enrichir un modèle de Frappes de Processus à l'aide de contraintes dérivées d'informations biologiques comme les vitesses de réaction, la connaissance d'inhibitions de réactions en présence de certains composants, ou celle des réactions précises ayant lieu au sein du système, etc. La connaissance de telles informations peut permettre de privilégier un chemin sur un autre à un moment clef de l'évolution du système, empêchant ainsi une certaine évolution du système, en favorisant l'apparition d'une autre. Leur intégration permet donc d'affiner le modèle en réduisant les comportements possibles afin d'obtenir un modèle plus proche du système étudié. Ces connaissances peuvent être intégrées sous la forme de préemptions (la jouabilité d'une action peut empêcher la jouabilité d'une autre action) ou de simultanéité (plusieurs processus peuvent évoluer simultanément).

La sémantique standard des Frappes de Processus développée par Paulevé et al. (2011*a*) et rappelée à la section 2.2 en page 24 ne permet pas de concilier l'introduction de contraintes dérivées d'informations biologiques et une bonne capacité d'analyse des modèles ainsi créés. En effet, s'il est possible d'y intégrer des informations temporelles sous la forme de paramètres stochastiques, tel que mentionné à la section 2.2.5 en page 36, en revanche les analyses puissantes de la dynamique rappelées à la section 2.2.4 en page 34 ne sont alors plus valables. En effet, celles-ci ne prennent pas en compte les fenêtres de tir introduites par les paramètres stochastiques. L'analyse des modèles doit alors être effectuée à l'aide d'outils de *model checking* probabilistes, qui doivent faire face à l'explosion combinatoire provoquée par l'ajout de la dimension temporelle continue. Il n'est alors généralement plus possible d'envisager d'étudier les modèles de plus de cinq composants avec une précision acceptable (Paulevé, 2011, p. 170).

De plus, au niveau de la modélisation, la représentation des coopérations avec des Frappes de Processus standards souffre de certaines lacunes. Ces coopérations sont modélisées à l'aide de sortes coopératives, décrites à la section 2.2.2, et souffrent d'un décalage temporel entre les sortes à représenter et la mise à jour du processus actif de la sorte coopérative, qui peut entraîner l'existence de « faux états » pour la sorte coopérative. Plusieurs solutions sont proposées dans ce chapitre, qu'il s'agisse de rendre les actions de mise à jour des sortes coopératives prioritaires ou plus simplement de les remplacer par une forme plus complexe d'action.

L'une des pistes permettant d'affiner un modèle de Frappes de Processus consiste à y intégrer des informations de préemption entre les actions afin d'affiner la dynamique. Une telle approche permet de modéliser des contraintes temporelles, toute action modélisant une réaction très rapide étant par exemple systématiquement jouée avant les actions modélisant des réactions plus lentes. D'autres contraintes peuvent aussi être prises en compte, comme la concurrence entre réactions, mais aussi la représentation de comportements propres à la modélisation et n'ayant pas nécessairement de sens biologique.

Les formes alternatives de Frappes de Processus présentées dans ce chapitre se concentrent donc sur les notions de préemption et de simultanéité d'une action par rapport à une autre. La préemption permet à une action d'avoir priorité sur une autre ou, du point de vue inverse, permet d'empêcher le jeu d'une action dans une situation où elle pourrait normalement être jouable selon la dynamique des Frappes de Processus standards. Une telle

préemption peut être opérée de façon généralisée comme c'est le cas pour les Frappes de Processus avec classes de priorités (section 3.1), où chaque action peut bloquer l'ensemble des actions de priorité inférieure ; ou de façon ponctuelle, comme au sein des Frappes de Processus avec arcs neutralisants (section 3.2), qui permet de définir des relations plus fines de préemption entre actions individuelles (sans que cette préemption ne soit généralisée à un ensemble d'autres actions). À l'inverse, la simultanéité entre actions permet de s'assurer qu'un ensemble de frappes est joué de façon simultanée, ou plus généralement qu'un ensemble de processus bondit en même temps, comme permettent de le représenter les Frappes de Processus avec actions plurielles (section 3.3). Nous montrons enfin au cours de ce chapitre que ces différentes sémantiques sont deux à deux aussi expressives, ce qui avait été résumé par la figure 1.1 en page 11.

Les apports de ces formes alternatives de Frappes de Processus permettent de restreindre la dynamique d'un modèle par rapport aux Frappes de Processus standards. Elles se posent en alternatives à l'ajout de paramètres stochastiques dans les Frappes de Processus (cf. section 2.2.5 en page 36) qui permettent d'ajouter une dimension probabiliste dans ce formalisme. Leur principal atout est de renforcer la puissance d'expression des Frappes de Processus, ce qui a pour conséquence de simplifier l'écriture et la lecture des modèles, mais aussi d'offrir de nouvelles capacités de modélisation. Par ailleurs, ces différents formalismes sont tous compatibles avec les méthodes d'analyse statique développées au chapitre 4, ce qui assure de pouvoir étudier efficacement la dynamique des modèles créés.

Ces différentes notions font naturellement écho aux problématiques plus générales d'enrichissement dans les modèles discrets. Ainsi, on peut par exemple rapprocher la notion d'arc neutralisant des Frappes de Processus à celle d'arc inhibiteur propre aux réseaux de Petri (Peterson, 1977) à la différence que celle-ci permet de préempter des actions en fonction de l'activité d'une place (correspondant ici à un processus ou à une sorte, selon le point de vue). De même, la notion de priorités a déjà été introduite dans certaines sémantiques de réseaux de Petri (Marsan, Balbo, Chiola & Conte, 1987) et de π-calcul (John, Lhoussaine, Niehren & Uhrmacher, 2010). Enfin, les actions plurielles permettent de se rapprocher de la classe des modèles à dynamique synchrone, dont font partie les automates synchronisés, ou encore les systèmes d'équations biochimiques dans la sémantique booléenne de Biocham, comme cela sera étudié plus tard dans ce manuscrit, au chapitre 5.

La définition du formalisme des Frappes de Processus avec classes de priorités a été publiée dans (Folschette, Paulevé, Magnin & Roux, 2013).

3.1 Frappes de Processus avec classes de priorités

Pour tout entier naturel k non nul, les *Frappes de Processus avec k classes de priorités* sont des Frappes de Processus dont l'ensemble des actions est partitionné en k ensembles, chacun étant associé à une classe de priorité distincte. Cela signifie qu'une action est jouable dans un état si et seulement si, en plus de la condition de la présence du frappeur et de la cible, il n'existe aucune autre action appartenant à une classe de priorité plus grande qui soit aussi jouable dans cet état. Il est à noter que les classes de priorités sont étiquetées de façon décroissante par les entiers de l'ensemble $[\![1\,;k]\!]$ en fonction de l'importance de la priorité ; autrement dit, la classe de priorités 1 contient les actions les plus prioritaires, ne pouvant jamais être préemptées, tandis que la jouabilité d'une action de la classe de priorité k ne peut pas empêcher le jeu d'une autre action. Un exemple de

ce type de modèle est donné par la figure 3.1, où les différentes priorités sont signifiées par des étiquettes numérotées sur les actions.

Cette modélisation permet notamment de distinguer les actions en fonction de différents critères comme leur vitesse d'exécution (les actions les plus rapides étant jouées en priorité), ou tout autre paramètre permettant de déterminer l'existence de la préemption d'une action en fonction de la possibilité d'en jouer une autre. L'application la plus poussée de cette utilisation consisterait à classer les actions d'un modèle en fonction d'un tel critère, et à attribuer à chaque classe de priorité une action unique en fonction de ce classement. De cette manière, les actions seraient arrangées selon un ordre total défini par leurs priorités.

De même, ces classes de priorités permettent de prendre en compte des comportements non biologiques inhérents à la modélisation. Il est par exemple possible de donner une priorité différente aux actions qui n'ont pas de sens biologique propre — mais dont ce sens émerge uniquement dans leur relation avec d'autres actions. L'application la plus immédiate de ce cas est celle des actions de mise à jour des sortes coopératives, comme cela est développé au chapitre 4, où une classe de priorités supérieure offre l'avantage de supprimer les effets d'entrelacement, et ainsi de simuler le comportement d'une véritable porte logique sans le problème de décalage temporel soulevé à la section 2.2.2.

Cette représentation basée sur des classes de priorités permet de modéliser un système dont les actions peuvent être distinguées en plusieurs classes en fonction de leur importance, de leur vitesse d'exécution, ou encore d'autres facteurs leur donnant prévalence sur d'autres. Chaque action peut donc en préempter un ensemble d'autres en fonction de sa classe de priorité. Cela permet une représentation compacte des rapports de priorités entre actions ou, autrement dit, de leur ordonnancement, qui présente néanmoins quelques lacunes. Les phénomènes d'accumulation, notamment, n'y sont pas représentés ; un cycle d'actions prioritaires ne peut jamais être interrompu par une action moins prioritaire, menant à un cycle infini et pouvant contredire la réalité biologique. De plus, les classes de priorités définies pour un modèle sont invariables ; certains modèles pourraient cependant nécessiter l'évolution de certaines classes de priorités en fonction de la présence ou de l'absence d'un composant dans un état donné. Enfin, elles peuvent ne pas permettre la précision nécessaire à une représentation fidèle de certains modèles, notamment lorsqu'il est nécessaire de définir des préemptions ponctuelles comme le permettent les Frappes de Processus avec arcs neutralisants présentées à la section 3.2.

3.1.1 Définition

Définition 3.1 (Frappes de Processus avec k classes de priorités)**.** Si $k \in \mathbb{N}^*$, les *Frappes de Processus avec k classes de priorités* sont définies par un triplet $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H}^{\langle k\rangle})$, où $\mathcal{H}^{\langle k\rangle} = (\mathcal{H}^{(1)}\,;\ldots;\mathcal{H}^{(k)})$ est un k-uplet, et :

- $\Sigma \stackrel{def}{=} \{a, b, \ldots\}$ est l'ensemble fini et dénombrable des *sortes* ;
- $\mathcal{L} \stackrel{def}{=} \bigotimes_{a\in\Sigma} \mathcal{L}_a$ est l'ensemble fini des *états*, où $\mathcal{L}_a = \{a_0, \ldots, a_{l_a}\}$ est l'ensemble fini et dénombrable des *processus* de la sorte $a \in \Sigma$ et $l_a \in \mathbb{N}^*$. Chaque processus appartient à une unique sorte : $\forall(a_i\,;b_j) \in \mathcal{L}_a \times \mathcal{L}_b, a \neq b \Rightarrow a_i \neq b_j$;
- pour tout $n \in [\![1\,;k]\!]$, $\mathcal{H}^{(n)} \stackrel{def}{=} \{a_i \to b_j \nrightarrow b_l \mid (a\,;b) \in \Sigma^2 \wedge (a_i\,;b_j\,;b_l) \in \mathcal{L}_a \times \mathcal{L}_b \times \mathcal{L}_b \wedge b_j \neq b_l \wedge a = b \Rightarrow a_i = b_j\}$ est l'ensemble fini des *actions de priorité n*.

On note $\mathcal{H} \stackrel{def}{=} \bigcup_{n\in[\![1;k]\!]} \mathcal{H}^{(n)}$ l'ensemble de toutes les actions et, pour tout $n \in \mathbb{N}^*$ et $h \in \mathcal{H}^{(n)}$, $\mathsf{prio}(h) \stackrel{def}{=} n$.

On réutilise de surcroît les notations définies à la section 2.2 concernant les états et l'extraction de la sorte d'un processus.

À l'instar de la section 2.2, il faut définir un opérateur de jouabilité pour déterminer la dynamique des Frappes de Processus avec k classes de priorités. Cependant, à l'inverse de celui des Frappes de Processus standards (définition 2.13) il faut ici prendre en compte la possible présence d'actions jouables appartenant à des classes de priorités supérieures. Pour cela, il est suffisant de vérifier que le frappeur et la cible de toute action de priorité plus importante ne sont pas simultanément présents. En effet, prenons deux actions $g, h \in \mathcal{H}$ avec : $\mathsf{prio}(g) < \mathsf{prio}(h)$, et un état $s \in \mathcal{L}$ tel que $\mathsf{frappeur}(g) \in s \wedge \mathsf{cible}(g) \in s$; Deux cas de figures sont alors possibles :

- l'action g est jouable dans s — autrement dit, aucune autre action de priorité plus importante ne la préempte – et elle préempte h en conséquence,
- l'action g n'est pas jouable dans s, ce qui signifie qu'elle est préemptée par une action de priorité plus importante, qui préempte alors aussi l'action h.

Dans les deux cas, h n'est pas jouable, ce qui montre qu'il est suffisant de n'observer que la présence simultanée du frappeur et de la cible de chaque action de priorité supérieure pour déterminer la jouabilité de h. Nous obtenons alors l'opérateur de jouabilité donné à la définition 3.2.

Définition 3.2 (Opérateur de jouabilité ($F_p : \mathcal{H} \to F$))**.** L'opérateur de jouabilité des Frappes de Processus avec k classes de priorités est défini par :

$$\forall h \in \mathcal{H}, F_p(h) \equiv \mathsf{frappeur}(h) \wedge \mathsf{cible}(h) \wedge \left(\bigwedge_{\substack{g\in\mathcal{H}^{(n)} \\ n<\mathsf{prio}(h)}} \neg\left(\mathsf{frappeur}(g) \wedge \mathsf{cible}(g)\right) \right)$$

Exemple. Nous illustrons les possibilités offertes par l'introduction des classes de priorités par la figure 3.1 qui représente un modèle de Frappes de Processus avec 3 classes de priorités $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}^{\langle 3\rangle})$. Celui-ci reprend la structure du modèle de Frappes de Processus standards de la figure 2.6, en y ajoutant trois classes de priorités permettant de distinguer trois types d'actions :

- les actions de $\mathcal{H}^{(1)}$ permettent d'assigner une priorité maximale aux actions mettant à jour la sorte coopérative $f\,c$, et peuvent être considérées comme « instantanées » du point de vue du reste du modèle ;
- les actions de $\mathcal{H}^{(2)}$ assurent que la sorte a est mise à jour immédiatement en fonction de l'évolution de f et c, et peuvent être vues comme « urgentes » par rapport aux actions de $\mathcal{H}^{(3)}$;
- enfin, les actions restantes sont par conséquent considérées comme « lentes » ou « peu urgentes » en regard du reste du modèle ; il s'agit des actions de $\mathcal{H}^{(3)}$, qui représentent des processus biologiques plus lents.

Comme expliqué par la suite au chapitre 4, assigner la priorité maximale aux actions permettant la mise à jour des sortes coopératives permet d'éviter les comportements indésirables décrits à la page 25. De même, accorder aux actions de $\mathcal{H}^{(2)}$ le statut d'« urgentes » permet de s'assurer qu'elles seront jouées avant les actions de $\mathcal{H}^{(3)}$. Dans ce modèle, cela se traduit par le fait que l'activation ou la désactivation de a est forcée lorsque c et f évoluent, ce qui restreint la dynamique aux seuls comportements désirés. En effet, la seule dynamique possible, en partant de l'état initial $\langle a_0, c_0, f_1, f\,c_{10}\rangle$, consiste en un comportement stationnaire oscillant, où c et a sont alternativement activés et désactivés, interrompu par la désactivation de f qui entraîne irrémédiablement celle de c, sans possibilité de le ré-activer par la suite, et provoque cette fois un comportement stationnaire constant (qui se traduit en Frappes de Processus par un point fixe où a reste indéfiniment à sa dernière valeur (actif ou non). Le comportement stationnaire est donné par le scénario suivant, jouable dans l'état initial $\langle a_0, c_0, f_1, f\,c_{10}\rangle$:

$$\begin{aligned} f\,c_{10} \to a_0 \Rsh a_1 :: f_1 \to c_0 \Rsh c_1 :: c_1 \to f\,c_{10} \Rsh f\,c_{11} :: \\ c_1 \to a_1 \Rsh a_0 :: c_1 \to c_1 \Rsh c_0 :: c_0 \to f\,c_{11} \Rsh f\,c_{10} \end{aligned}$$

L'interruption de ce comportement stationnaire se fait grâce à l'auto-action $f_1 \to f_1 \Rsh f_0$, qui est de priorité 3, et donc jouable uniquement dans les deux états suivants : $\langle a_1, c_0, f_1, f\,c_{10}\rangle$ et $\langle a_0, c_1, f_1, f\,c_{11}\rangle$. Depuis le premier état, la désactivation est opérée par le scénario suivant :

$$f_1 \to f_1 \Rsh f_0 :: f_0 \to f\,c_{10} \Rsh f\,c_{00} \ ,$$

qui termine dans l'état $\langle a_1, c_0, f_0, f\,c_{00}\rangle$ et conserve donc le processus a_1, tandis que depuis le deuxième état, la désactivation est opérée par le scénario :

$$f_1 \to f_1 \Rsh f_0 :: f_0 \to f\,c_{11} \Rsh f\,c_{01} :: f_0 \to c_1 \Rsh c_0 :: c_0 \to f\,c_{01} \Rsh f\,c_{00} \ ,$$

qui termine en $\langle a_0, c_0, f_0, f\,c_{00}\rangle$ et conserve cette fois le processus a_0.

3.1.2 Équivalences entre Frappes de Processus avec k classes de priorités

Nous montrons à la section 4.2.1 que les Frappes de Processus avec k classes de priorités sont aussi expressives que les Frappes de Processus avec n classes de priorités, pour tout $k, n \in \mathbb{N}^\bullet$. Nous donnons pour cela un résultat encore plus fort : tout modèle de Frappes de Processus avec k classes de priorités peut être traduit en Frappes de Processus canoniques, comme défini à la section 4.1.1, qui sont des Frappes de Processus avec 2 classes de

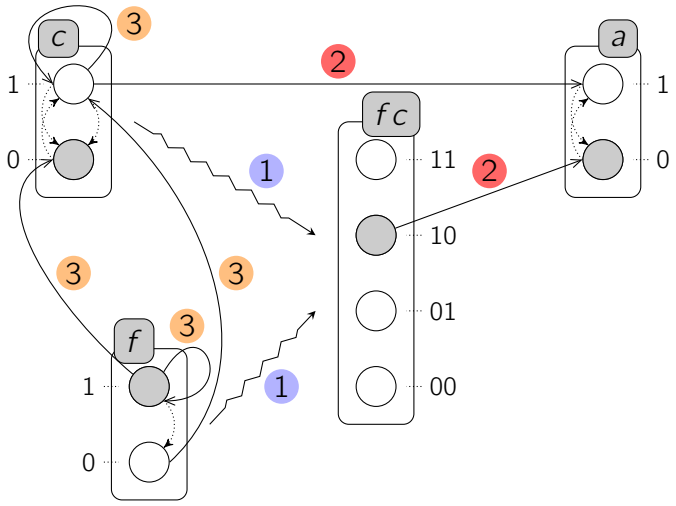


Figure 3.1 – Exemple de Frappes de Processus avec 3 classes de priorités. Ce modèle est issu de celui de la figure 2.6 auquel ont été rajoutées des classes de priorités. Les étiquettes numérotées (de 1 à 3) placées contre les flèches représentant les actions symbolisent leur appartenance à une classe de priorités donnée ; ainsi, on a notamment : $\mathcal{H}^{(2)} = \{fc_{10} \to a_0 \Rsh a_1 \,;\, c_1 \to a_1 \Rsh a_0\}$.

priorités avec une forme particulière. À l'inverse, les Frappes de Processus avec 2 classes de priorités sont naturellement à fortiori des Frappes de Processus avec k classes de priorités, pour tout $k \in \mathbb{N}^\bullet$.

Nous notons cependant que ce résultat n'inclut pas les Frappes de Processus avec 1 classe de priorité (c'est-à-dire les Frappes de Processus standards). En effet, il est intuité que leur expressivité est strictement moindre que les Frappes de Processus possédant plusieurs classes de priorités, mais nous ne démontrons pas ce résultat ici.

Théorème 3.1 (Équivalences entres Frappes de Processus avec classes de priorités)**.** *Pour tout $k, n \in \mathbb{N}^\bullet$, les Frappes de Processus avec k classes de priorités sont aussi expressives que les Frappes de Processus avec n classes de priorités.*

Démonstration. Nous utilisons pour cette démonstration l'opérateur d'aplatissement flat donné à la définition 4.10 en page 74 et les résultats qui le concernent. Soient $k, n \in \mathbb{N}^\bullet$, et soient $\mathcal{PH}$ des Frappes de Processus avec k classes de priorités. D'après le théorème 4.1, l'aplatissement $\mathsf{flat}(\mathcal{PH})$ est faiblement bisimilaire à $\mathcal{PH}$. Posons :

- $\mathsf{flat}(\mathcal{PH}) = (\Sigma\,;\mathcal{L}\,;\mathcal{H}^{\langle 2\rangle})$ avec $\mathcal{H}^{\langle 2\rangle} = (\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)})$,
- $\mathcal{PH}' = (\Sigma\,;\mathcal{L}\,;\mathcal{H}'^{\langle n\rangle})$ avec $\mathcal{H}'^{\langle n\rangle} = (\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}\,;\mathcal{H}'^{(3)}\,;\ldots;\mathcal{H}'^{(n)})$,
  où $\forall i \in [\![3\,;n]\!], \mathcal{H}'^{(i)} = \emptyset$.

Autrement dit, $\mathcal{PH}'$ sont les Frappes de Processus avec n classes de priorités identiques à $\mathsf{flat}(\mathcal{PH})$, où des classes de priorités vides ont été artificiellement ajoutées. Cela est possible car $n \geq 2$. Ainsi, $\mathcal{PH}'$ possède la même dynamique que $\mathsf{flat}(\mathcal{PH})$, et est donc faiblement bisimilaire à $\mathcal{PH}$ (autrement dit, sa dynamique est équivalente au jeu des actions de priorité 1 près). Ainsi, pour tout $k, n \in \mathbb{N}^\bullet$, toutes Frappes de Processus avec k classes de priorités peuvent être représentées en Frappes de Processus avec n classes de priorités. □

3.1.3 Réutilisation des résultats existants

Nous discutons dans cette section de la transposition des différents outils et résultats concernant les Frappes de Processus standards aux Frappes de Processus avec classes de priorités. Nous aborderons la question de la recherche des points fixes d'un modèle (section 3.1.3.1), de l'analyse statique des modèles (section 3.1.3.2) et des possibles (ré)utilisations des paramètres stochastiques (section 3.1.3.3).

3.1.3.1 Points fixes

Nous montrons dans cette sous-section que les points fixes des Frappes de Processus avec classes de priorités peuvent être obtenus de façon similaire à ceux des Frappes de Processus standards. Pour cela, nous définissons la notion de *fusion* d'un modèle de Frappes de Processus qui consiste à fusionner les classes de priorités en une seule, afin de retrouver un modèle de Frappes de Processus standards.

Pour tout modèle de Frappes de Processus avec k classes de priorités, pour $k \in \mathbb{N}^*$, nous notons dans la suite $\mathsf{fusion}(\mathcal{PH})$ la *fusion* de $\mathcal{PH}$, c'est-à-dire le même modèle dont les classes de priorités ont été fusionnées (définition 3.3). En d'autres termes, il s'agit d'un modèle de Frappes de Processus standards dont l'ensemble des actions est l'union de toutes les classes de priorités de $\mathcal{PH}$.

On peut constater que pour un modèle donné, si l'ensemble de toutes les actions reste le même, ajouter (ou retirer) des classes de priorités à des Frappes de Processus ne change pas l'ensemble de ses points fixes. En effet, le théorème 3.2 stipule que l'ensemble des points fixes des Frappes de Processus avec k classes de priorités $\mathcal{PH}$ est identique à l'ensemble des points fixes de sa fusion $\mathsf{fusion}(\mathcal{PH})$. Cela se démontre simplement en constatant qu'il existe une action jouable dans un état donné du modèle $\mathcal{PH}$ si et seulement si il en existe une dans le même état du modèle fusionné. En effet, si une action est jouable dans un état donné de $\mathsf{fusion}(\mathcal{PH})$, alors soit elle est jouable car non préemptée dans le même état de $\mathcal{PH}$, soit elle ne l'est pas car elle est préemptée par une autre action qui, elle, est jouable. L'autre sens de la démonstration est immédiat car l'ajout de priorités restreint la dynamique et n'ajoute aucun comportement supplémentaire possible. Ce résultat permet d'appliquer aux Frappes de Processus avec k classes de priorités les méthodes de recherche de points fixes développée pour les Frappes de Processus standards. L'une d'entre elles repose sur la recherche de n-cliques (section 2.2.3 en page 33) et sa résolution jouit aujourd'hui de méthodes de résolutions performantes. D'autres méthodes de recherche peuvent être envisagées, par exemple par l'utilisation de programmation logique. En effet, la formalisation du problème de recherche de points fixes est très simple, et sa résolution est donc traitée efficacement par des méthodes SAT ou par ASP. Enfin, ce résultat permet aussi de conclure quant aux ensembles de points fixes de deux Frappes de Processus avec un nombre de classes de priorités différent, à condition que leurs modèles fusionnés soient identiques.

Définition 3.3 (Fusion (fusion))**.** Soient $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}^{\langle k \rangle})$ des Frappes de Processus avec k classes de priorités, où $k \in \mathbb{N}^*$. On note $\mathsf{fusion}(\mathcal{PH}) = (\Sigma, \mathcal{L}, \mathcal{H})$ les Frappes de Processus standards appelées *fusion* de $\mathcal{PH}$, dont l'ensemble des actions est l'union de toutes les classes de priorités de $\mathcal{PH}$.

Théorème 3.2 (Points fixes des Frappes de Processus avec classes de priorités)**.** *Soient $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}^{\langle k \rangle})$, où $k \in \mathbb{N}^*$, des Frappes de Processus avec k classes de priorités, et $r \in \mathcal{L}$:*

$$\exists s \in \mathcal{L}, r \to_{\mathcal{PH}} s \iff \exists s' \in \mathcal{L}, r \to_{\mathsf{fusion}(\mathcal{PH})} s'$$

Démonstration. On pose : $\mathcal{PH}' = \mathsf{fusion}(\mathcal{PH})$.

($\Rightarrow$) Supposons qu'il existe $s \in \mathcal{L}$ tel que $r \to_{\mathcal{PH}} s$; cela signifie qu'une action $h \in \mathcal{H}$ est jouable dans $\mathcal{PH}$. Cette action est donc aussi jouable dans $\mathcal{PH}'$ car son frappeur et sa cible sont présents, d'où : $r \to_{\mathcal{PH}'} (r \cdot h)$.

($\Leftarrow$) Supposons qu'il existe $s' \in \mathcal{L}$ tel que $r \to_{\mathcal{PH}'} s'$; cela signifie qu'une action $g \in \mathcal{H}$ est jouable dans $\mathcal{PH}'$. Le frappeur et la cible de g sont donc présents dans r.

- Si cette action n'est pas préemptée par une autre action dans $\mathcal{PH}$, elle est alors jouable et $r \to_{\mathcal{PH}} (r \cdot g)$;
- Si cette action est préemptée par une autre action g' dans $\mathcal{PH}$, cela signifie que cette action g' est jouable, et $r \to_{\mathcal{PH}} (r \cdot g')$. □

3.1.3.2 Analyse statique

Afin de permettre une étude efficace des Frappes de Processus standards de grande taille, une analyse statique par interprétation abstraite avait été développée par Paulevé et al. (2012). Son principe est rappelé à la section 2.2.4 en page 34. L'ajout de classes de priorités au formalisme a pour conséquence d'en restreindre la dynamique, mais n'ajoute aucun comportement supplémentaire. Ainsi, pour tout modèle de Frappes de Processus avec k classes de priorités, il est toujours possible de réutiliser l'analyse statique par sur-approximation en l'appliquant au modèle $\mathsf{fusion}(\mathcal{PH})$. Bien que toujours exacte, cette analyse pourra néanmoins être moins conclusive, n'ayant pas été spécifiquement adaptée aux modèles comportant des classes de priorités.

En revanche, l'analyse statique par sous-approximation n'est plus valable, car elle ne prend pas en compte les possibles préemptions entre actions qui rendent impossibles certaines atteignabilités. C'est pourquoi une nouvelle version de l'analyse statique par sous-approximation sera développée au chapitre 4 sur une classe particulière de Frappes de Processus avec 2 classes de priorités, appelées Frappes de Processus canoniques. Cette classe particulière n'autorise les actions avec une forte priorité que pour la mise à jour des sortes coopératives. Cependant, nous montrons aussi à la section 4.2 en page 72 que toutes Frappes de Processus avec un nombre quelconque de classes de priorités peuvent être traduites en Frappes de Processus canoniques équivalentes, étendant ainsi l'analyse statique mentionnée aux modèles de Frappes de Processus avec k classes de priorités.

3.1.3.3 Paramètres stochastiques

Il est théoriquement toujours possible d'utiliser, dans des Frappes de Processus avec k classes de priorités, des paramètres stochastiques tels que ceux développés par Paulevé et al. (2011*a*) et mentionnés à la section 2.2.5 en page 36. Il suffirait pour cela d'empêcher la sensibilisation de toute action préemptée par une action de priorité plus importante. Cependant, un autre parallèle intéressant peut être tracé entre l'approche par définition de classes de priorités et l'approche par introduction de paramètres stochastiques.

L'ajout de paramètres stochastiques a pour but d'assigner un intervalle de tir temporel à chaque action, afin de s'assurer (avec un certain niveau de confiance) que l'action sera nécessairement tirée dans cet intervalle à partir du moment où elle est devenue jouable. La simulation stochastique développée par Paulevé et al. (2011*a*) ne permet actuellement pas de prendre en compte des classes de priorités entre actions. Il faudrait en effet pour cela raffiner la machine stochastique afin d'y intégrer des contraintes supplémentaires concernant la jouabilité et la sensibilisation de chaque action. Cependant, l'aplatissement proposé à la définition 4.11 en page 75 permet théoriquement d'obtenir un modèle équivalent, utilisable avec la simulation stochastique. Nous détaillons ici le principe de ce procédé sans toutefois donner de preuve de sa fidélité.

L'aplatissement mentionné ci-dessus permet en effet d'obtenir un modèle de Frappes de Processus avec 2 classes de priorités ayant une certaine forme qui permet de distinguer les actions instantanées (de priorité 1) propres à la modélisation des actions possédant une durée (de priorité 2) et permettant de représenter des processus biologiques. On peut ainsi, dans le modèle obtenu, attribuer à chaque action secondaire (c'est-à-dire de priorité 2) des paramètres stochastiques identiques à ceux de l'action originelle dont elle est issue, et à chaque action primaire (c'est-à-dire de priorité 1) une absorption de stochasticité infinie. Le modèle résultant devrait alors posséder une dynamique identique à un potentiel modèle hybride mêlant classes de priorités et paramètres stochastiques.

Plutôt que d'intégrer des données stochastiques connues dans un modèle, il est aussi possible de s'en servir pour l'obtention d'un modèle discret. En effet, pour créer un modèle de Frappes de Processus avec classes de priorités, il est nécessaire de connaître certaines relations entre les phénomènes modélisés afin de répartir correctement les actions entre les différentes classes de priorités. Il peut s'agit de données de préemption (un phénomène en bloque un autre), de durée (un phénomène est beaucoup plus rapide qu'un autre), de vitesse de déclenchement (un phénomène se déclenche toujours avant un autre), etc. La piste que nous présentons dans la suite est l'utilisation d'un modèle de Frappes de Processus standards enrichi à l'aide de paramètres stochastiques. En effet, de tels paramètres stochastiques définis pour chaque action correspondent à une fenêtre de tir avec un intervalle de confiance donné (généralement 95 %), qu'il est possible d'approximer (Paulevé, 2011, p. 72), et donc d'utiliser de façon interchangeable. Ces paramètres stochastiques permettent donc au modélisateur de rendre une action d'autant plus « urgente » que sa fenêtre de tir est proche de sont instant de sensibilisation.

Ainsi, sous la condition que l'on peut distinguer les intervalles de tir en différentes classes entre lesquelles les intervalles ne se recouvrent pas, il est possible d'approximer la modélisation avec des paramètres stochastiques à l'aide de classes de priorités. En associant une classe de priorités à chaque ensemble d'intervalles de tir — la priorité la plus haute étant naturellement associée à la classe dont les intervalles sont les plus proches de zéro, — on retrouve alors un modèle dont les caractéristiques dynamiques sont proches du modèle initial. En effet, le système de classes de priorités permet d'approcher une dynamique où chaque intervalle est joué en priorité avant tous les suivants.

Exemple. Le modèle de Frappes de Processus standards de la figure 2.6 en page 32 peut être enrichi à l'aide des paramètres proposés en page 37, permettant ainsi de contraindre (à l'intervalle de confiance près) le jeu des actions entre elles, et favorisant le jeu des actions « urgentes ». Cela permet notamment de rendre « instantanées » les actions de mise à jour des sortes coopératives, mais aussi de rendre « urgentes » les actions faisant bondir a, et de réguler l'évolution de c afin de lui donner un rôle d'horloge.

Si on observe les fenêtres de tir définies pour ce modèle, on constate qu'elles ne se recouvrent pas ; autrement dit, on peut clairement distinguer des classes d'actions qui seront tirées en priorité par rapport à d'autres classes. Nous proposons donc de traduire ces classes de fenêtres de tir en classes de priorités qui auront le même rôle : empêcher le jeu d'une action tant que d'autres actions plus « urgentes » sont jouables. Nous décidons de distinguer pour cet exemple trois classes de priorités :

- les actions instantanées car de taux infini (∞) feront partie de la classe de plus forte priorité,
- les actions qui influencent a (**a**) peuvent être considérés comme « urgentes », et formeront la classe de priorité intermédiaire,
- les autres actions (**b** et **c**) seront regroupées dans la classe de priorité faible, car considérées comme « peu urgentes ».

Nous obtenons ainsi le modèle donné à la figure 3.1, dont la dynamique a été précédemment discutée en page 43. Nous constatons que la dynamique du modèle en Frappes de Processus avec classes de priorités est « strict » : il n'y a pas d'intervalle de confiance et plus aucune notion de probabilités.

Cependant, il n'est évidemment pas possible d'atteindre avec cette méthode le même niveau de précision, en utilisant un nombre discret de classes de priorités, qu'avec des intervalles de tir définis sur une ligne temporelle continue. Il n'est par exemple pas possible de représenter fidèlement le recouvrement de deux intervalles de tir, qui aurait pour conséquence de favoriser le tir d'une première action sans totalement préempter la seconde action, autrement qu'en mettant ces deux actions au même niveau de priorité. À l'inverse, deux intervalles de tir qui ne se recouvrent pas devraient être associés à la même classe de priorités s'ils sont tous deux recouverts par un troisième intervalle, ce qui aura pour conséquence de mettre les trois actions sur un pied d'égalité, alors que les paramètres stochastiques représentent une situation différente.

Enfin, il faut noter que les phénomènes d'accumulation ou de retard ne sont pas pris en compte dans la modélisation par classes de priorités. En effet, si deux actions n'ont pas de sorte en commun, elles devraient dans l'idéal pouvoir évoluer de façon indépendante, ce qui est notamment permis par la simulation stochastique. En revanche, dans un formalisme avec classes de priorités, si l'une des actions est plus prioritaire que l'autre, elle exercera tout de même sa préemption sur l'autre. L'une des façons de pallier ce défaut est l'utilisation d'arcs neutralisants, comme développé à la šecrefphan.

Exemple. L'exemple de la figure 3.1 en page 45 peut être obtenu à l'aide de paramètres stochastiques, comme expliqué à la page ci-contre. Cependant, cette approche possède certaines limites ; il n'est pas possible par exemple de n'autoriser la désactivation de f qu'après un nombre donné d'oscillations de a ou de c. Pour représenter cela, un modélisateur pourrait être tenté d'introduire une classe de priorité 4 afin d'y intégrer l'action $f_1 \to f_1 \overset{\prime}{\Gamma} f_0$, en effectuant un parallèle avec les paramètres stochastiques proposés en page 37 qui permettent de ne tirer cette action qu'après un certain nombre d'oscillations ; ou encore en constatant que les paramètres de cette action devraient permettre la création d'une telle classe de priorités, car son intervalle de tir n'en recouvre aucun autre. Cependant, un tel choix de conception aurait uniquement pour effet de rendre ladite action injouable, car sans cesse préemptée par des actions de priorité supérieure. En effet, une telle modélisation en Frappes de Processus avec 4 classes de priorités fait abstraction du temps continu, ce qui signifie qu'il n'y a plus de notion d'« accumulation » du temps

de sensibilisation, sur laquelle reposait le fait de pouvoir jouer l'action en question après une certaine durée de sensibilisation.

Nous avons ici montré l'une des limites de la modélisation par Frappes de Processus avec classes de priorités, qui fait abstraction du temps continu sur lequel se basent les paramètres stochastiques proposés par Paulevé et al. (2011*a*). Le modélisateur doit effectivement être prudent pour ne pas rendre impossibles des comportements qui sont seulement retardés par des modélisations chronométriques. Cependant, à condition d'éviter ces écueils, nous avons montré qu'il est possible de représenter un modèle avec des données temporelles à l'aide de ce formalisme, et donc d'en étudier efficacement la dynamique à l'aide de la traduction et des méthodes d'analyse statique proposés au chapitre 4.

3.2 Frappes de Processus avec arcs neutralisants

Nous introduisons ici la notion d'*arc neutralisant* dans les Frappes de Processus afin de représenter la préemption d'une action par une seule autre. Les *Frappes de Processus avec arcs neutralisants* (définition 3.1) permettent notamment une modélisation plus atomique par rapport aux classes de priorités présentées à la section 3.1.

Un arc neutralisant est un couple d'actions noté $h_1 \rtimes h_2$, où h_1 est appelée *action bloquante*, et peut préempter h_2, appelée *action bloquée*, dans certaines situations. Avec la présence d'arcs neutralisants, une action est dite *activée* dans un état donné si son frappeur et sa cible y sont présents ; une action est donc activée pour les Frappes de Processus avec arcs neutralisants là où elle était immédiatement jouable pour les Frappes de Processus standards (définition 2.13). Une action est *jouable* pour les Frappes de Processus avec arcs neutralisants si et seulement si elle est activée, et que pour tout arc neutralisant la bloquant, son action bloquante n'est pas activée. Une action activée mais qui n'est pas jouable est dite *neutralisée*. La figure 3.2 propose un exemple de Frappes de Processus avec deux arcs neutralisants.

Il est à noter que la neutralisation d'une action par une autre ne dépend donc pas de la jouabilité de l'action bloquante, mais uniquement de son activation. Cela permet d'avoir un modèle cohérent, sans quoi certaines situations pourraient ne pas être définies, notamment dans le cas d'un interblocage. Ainsi, faire reposer la neutralisation d'une action bloquée sur la jouabilité de l'action bloquante devient inextricable dans un cas comme le suivant : $h_1 \rtimes h_2$, $h_2 \rtimes h_3$ et $h_3 \rtimes h_1$, car si les trois actions h_1, h_2 et h_3 sont actives, leur jouabilité reste indéterminée. En revanche, si cette neutralisation repose sur l'état activé d'une action, la situation précédente se résout immédiatement car aucune des trois actions n'est jouable. On constate par ailleurs qu'une action peut en neutraliser une autre même si elle-même est neutralisée. Nous ne nous avancerons cependant pas sur la signification biologique de ce fait.

Enfin, il semble nécessaire de faire un parallèle entre les arcs inhibiteurs développés ici et les arcs inhibiteurs utilisés dans les réseaux de Petri (Peterson, 1977). Leur rôle est effectivement proche, leur but étant la préemption d'une action en fonction d'une condition extérieure à son déclenchement. Malgré tout, les arcs inhibiteurs des réseaux de Petri se différencient car ils reposent sur la présence d'un certain nombre de jetons dans une place — ce qui se traduirait, en Frappes de Processus, par la présence d'un certain processus actif d'une sorte donnée — là où les arcs inhibiteurs des Frappes de Processus ne permettent l'inhibition qu'en fonction de l'activité d'une autre action. Cependant, un tel choix de conception peut aisément se pallier dans un sens comme dans l'autre. En

effet, représenter l'activité d'une action revient à créer deux arcs inhibiteurs en réseaux de Petri : l'un pour le frappeur et l'autre pour la cible. À l'inverse, il est possible qu'une action h se bloque elle-même, à l'aide d'un « auto-arc neutralisant » $h \rtimes h$, permettant ainsi de bloquer une autre action g en fonction de la présence du processus $\mathsf{frappeur}(h)$ à l'aide d'un deuxième arc neutralisant $h \rtimes g$.

3.2.1 Définition

Définition 3.4 (Frappes de Processus avec arcs neutralisants)**.** Les *Frappes de Processus avec arcs neutralisants* sont définies par un quadruplet $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H}\,;\mathcal{N})$, où :

- $\Sigma \stackrel{def}{=} \{a, b, \ldots\}$ est l'ensemble fini et dénombrable des *sortes* ;
- $\mathcal{L} \stackrel{def}{=} \bigotimes_{a\in\Sigma} \mathcal{L}_a$ est l'ensemble fini des *états*, où $\mathcal{L}_a = \{a_0, \ldots, a_{l_a}\}$ est l'ensemble fini et dénombrable des *processus* de la sorte $a \in \Sigma$ et $l_a \in \mathbb{N}^*$. Chaque processus appartient à une unique sorte : $\forall (a_i\,;b_j) \in \mathcal{L}_a \times \mathcal{L}_b, a \neq b \Rightarrow a_i \neq b_j$;
- $\mathcal{H} \stackrel{def}{=} \{a_i \to b_j \Rsh b_l \mid (a\,;b) \in \Sigma^2 \wedge (a_i\,;b_j\,;b_l) \in \mathcal{L}_a \times \mathcal{L}_b \times \mathcal{L}_b \wedge b_j \neq b_l \wedge a = b \Rightarrow a_i = b_j\}$ est l'ensemble fini des actions ;
- $\mathcal{N} = \{h_1 \rtimes h_2 \mid (h_1\,;h_2) \in \mathcal{H} \times \mathcal{H}\}$ est l'ensemble fini des arcs neutralisants.

Un arc neutralisant $u = h_1 \rtimes h_2 \in \mathcal{N}$ est donc formellement un couple d'actions. On note $\mathsf{bloquante}(u) = h_1$ la première action du couple u et $\mathsf{bloquée}(u) = h_2$ sa seconde action. On réutilise par ailleurs les autres notations définies à la section 2.2.

L'opérateur de jouabilité des frappes de Processus avec arcs neutralisants (définition 3.5) se concentre sur la présence du frappeur et de la cible de l'action considérée, et sur celle de toutes ses actions bloquantes. En ce sens, il est semblable à celui des Frappes de Processus avec k classes de priorités (définition 3.2).

Définition 3.5 (Opérateur de jouabilité ($F_{an} : \mathcal{H} \to F$))**.** L'opérateur de jouabilité des Frappes de Processus avec arcs neutralisants est défini par :

$$\forall h \in \mathcal{H}, F_{an}(h) \equiv \mathsf{frappeur}(h) \wedge \mathsf{cible}(h) \wedge \left(\bigwedge_{\substack{g\in\mathcal{H} \\ g\rtimes h\in\mathcal{N}}} \neg\left(\mathsf{frappeur}(g) \wedge \mathsf{cible}(g)\right) \right)$$

Exemple. La figure figure 3.2 présente un exemple de Frappes de Processus avec arcs neutralisants. Il s'agit du modèle de segmentation métazoaire $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H}\,;\mathcal{N})$ précédemment proposé à la figure 2.6 en page 32 et enrichi à l'aide de l'ensemble d'arcs neutralisants suivant :

$$\mathcal{N} = \{ \quad c_1 \to a_1 \Rsh a_0 \rtimes c_1 \to c_1 \Rsh c_0 \quad , \quad f\,c_{10} \to a_0 \Rsh a_1 \rtimes f_1 \to c_0 \Rsh c_1 \quad \}$$

Ces arcs neutralisants permettent de se rapprocher du comportement observé biologiquement : ils assurent que l'évolution de la sorte a (son activation ou sa désactivation) primera toujours sur celle de la sorte c. Ainsi, c possède bien un rôle d'horloge, et son rôle inhibiteur envers a doit toujours aboutir avant son prochain cycle d'horloge. Le bon scénario suivant est alors jouable dans le modèle $\mathcal{PH}$ depuis l'état $\langle a_0, c_0, f_1, f\,c_{10}\rangle$:

$$f\,c_{10} \to a_0 \Rsh a_1 :: f_1 \to c_0 \Rsh c_1 :: c_1 \to f\,c_{10} \Rsh f\,c_{11} :: c_1 \to a_1 \Rsh a_0 :: c_1 \to c_1 \Rsh c_0 :: c_0 \to f\,c_{11} \Rsh f\,c_{10}$$

En revanche, il est impossible de jouer l'action $f_1 \to c_0 \Rsh c_1$ depuis ce même état car celle-ci est neutralisée. Cela permet d'assurer que a sera activé avant le prochain cycle de l'horloge modélisée par c. L'ajout de ces deux arcs neutralisants permet en effet d'introduire des contraintes locales de ce type dans le modèle. Ces contraintes sont plus fines que celles offertes par les Frappes de Processus avec classes de priorités, car la préemption d'une action n'empêche notamment pas le parallélisme à d'autres endroits du modèles, comme par exemple l'auto-désactivation de f. Cette finesse fait parfois défaut aux Frappes de Processus avec classes de priorités, comme cela avait été discuté en page 49.

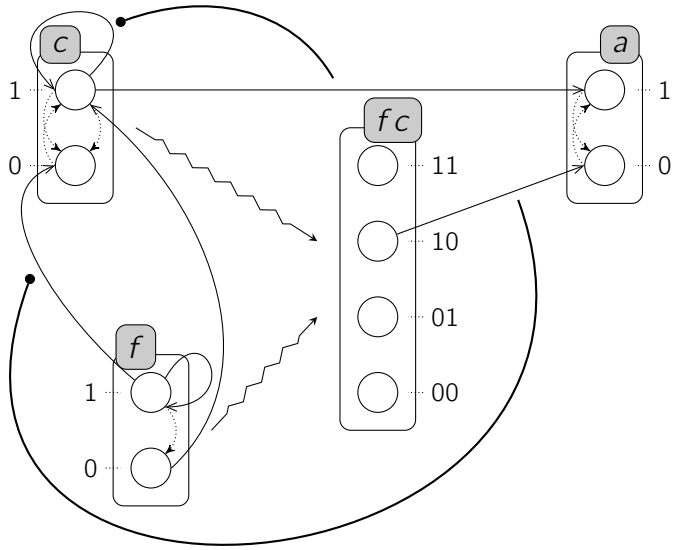


Figure 3.2 – Exemple de Frappes de Processus avec arcs neutralisants. Les arcs neutralisants sont représentés par un trait gras entre deux actions terminant par un point du côté de l'action bloquée.

On constate que les arcs neutralisants introduits dans le modèle de la figure 3.2 ne suffisent pas à forcer la mise à jour de la sorte coopératives. De façon générale, la seule possibilité s'offrant au modélisateur pour assurer cette mise à jour avec des arcs neutralisants est d'en ajouter pour chaque couple d'actions $h_1 \rtimes h_2$ où h_1 est une action de mise à jour d'une sorte coopérative, et h_2 est une action standard (qui ne met pas à jour de sorte coopérative). Cette solution rend cependant le modèle anormalement complexe, car elle nécessite un grand nombre d'arcs neutralisants. Une solution alternative consiste à étendre les Frappes de Processus avec arcs neutralisants afin d'y inclure des classes de priorités telles que vues à la section 3.1. Ainsi, il est possible d'assigner une priorité forte aux actions mettant à jour les sortes coopératives et une priorité faible actions standards, tout en définissant des préemptions fines à l'aide d'arcs neutralisants entre les actions standards. Cette approche a l'avantage de tirer le meilleur parti des deux types de modélisations : les classes de priorités sont utilisées pour définir des relations de préemptions globales entre des ensemble (potentiellement larges) d'actions, alors que les arcs neutralisants permettent de définir des préemptions individuelles, utiles pour rendre compte de contraintes locales connues.

Exemple. On note que le scénario suivant est jouable dans le modèle de la figure 3.2 depuis l'état $\langle a_0, c_0, f_1, f\,c_{10}\rangle$:

$$f\,c_{10} \to a_0 \Rsh a_1 :: f_1 \to c_0 \Rsh c_1 :: c_1 \to a_1 \Rsh a_0 :: f\,c_{10} \to a_0 \Rsh a_1$$

Ce comportement n'est pourtant pas désirable car il « court-circuite » l'état de c qui a été modifié, et permet donc d'activer a à tout moment. Il est dû au fait que la sorte coopérative fc n'est pas mise à jour, permettant donc de jouer l'action $fc_{10} \rightarrow a_0 \nearrow a_1$ même lorsque c_1 est actif.

Pour pallier cela, il est possible, comme expliqué ci-dessus, d'utiliser un modèle hybride mêlant deux classes de priorités (pour prioriser les actions de mise à jour des sortes coopératives) à des arcs neutralisants (pour définir des préemptions locales entre les autres actions). Ainsi, en assignant une priorité haute aux actions mettant la sorte fc à jour, il est possible de retrouver un bon comportement comme celui décrit en page 43.

3.2.2 Équivalence avec les Frappes de Processus avec k classes de priorités

Les Frappes de Processus avec arcs neutralisants ont une expressivité équivalente aux Frappes de Processus avec k classes de priorités, où $k \in \mathbb{N}^*$. En effet, la théorème 3.3 propose une traduction des Frappes de Processus avec k classes de priorités en Frappes de Processus avec arcs neutralisants. À l'inverse, les Frappes de Processus avec arcs neutralisants peuvent être représentées en Frappes de Processus canoniques, qui sont une sous-classe des Frappes de Processus avec 2 classes de priorités, comme démontré à la section 4.2.2.

Théorème 3.3 (Équivalence avec les classes de priorités)**.** *Soient $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}^{\langle k \rangle})$ des Frappes de Processus avec k classes de priorités, où $k \in \mathbb{N}^*$. Il existe un modèle $\overline{\mathcal{PH}}$ de Frappes de Processus avec arcs neutralisants tel que :*

$$\forall s, s' \in \mathcal{L}, s \rightarrow_{\mathcal{PH}} s' \iff s \rightarrow_{\overline{\mathcal{PH}}} s'$$

Démonstration. On pose : $\overline{\mathcal{PH}} = (\Sigma, \mathcal{L}, \mathcal{H}, \mathcal{N})$ les Frappes de Processus avec arcs neutralisants dont l'ensemble des actions est l'union de toutes les classes de priorités de $\mathcal{PH}$, et : $\mathcal{N} = \{g \rtimes h \mid g \in \mathcal{H}^{(n)}, h \in \mathcal{H}^{(k)}, n < k\}$. Soit $h \in \mathcal{H}$. D'après la définition de $\overline{\mathcal{PH}}$, on constate que : $\{g \in \mathcal{H} \mid g \rtimes h \in \mathcal{N}\} = \{g \in \mathcal{H} \mid n < \mathsf{prio}(h)\}$. Ainsi, $\forall h \in \mathcal{H}, F_p(h) = F_{an}(h)$, ce qui implique que les actions jouables dans l'état s du modèle $\mathcal{PH}$ sont exactement les actions jouables dans l'état s du modèle $\overline{\mathcal{PH}}$. Naturellement, leur jeu amène aux mêmes états car la sémantique des Frappes de Processus avec arcs neutralisants et celle des Frappes de Processus avec k classes de priorités sont basées sur la même définition de la sémantique (définition 2.14 en page 29). D'où : $s \rightarrow_{\mathcal{PH}} s' \iff s \rightarrow_{\overline{\mathcal{PH}}} s'$. □

3.2.3 Réutilisation des résultats existants

À l'instar de la section 3.1.3 en page 46, nous transposons dans la suite une partie des résultats des Frappes de Processus standards aux Frappes de Processus avec arcs neutralisants.

3.2.3.1 Points fixes

Nous proposons ici deux méthodes pour obtenir les points fixes d'un modèle de Frappes de Processus avec arcs neutralisants. La première est directe mais partielle, car elle ne

prend pas en compte certaines préemptions, tandis que la seconde est indirecte (car elle nécessite une traduction du modèle) mais permet d'obtenir tous les points fixes.

De façon analogue à la section 3.1.3.1, nous définissons ici la *fusion* d'un modèle de Frappes de Processus avec arcs neutralisants comme étant le même modèle en Frappes de Processus standards, dont les arcs neutralisants ont été ignorés (définition 3.6). Cette définition nous permet d'avancer le résultat suivant : les points fixes de $\mathsf{fusion}_{an}(\mathcal{PH})$ sont des points fixes de $\mathcal{PH}$, d'après la contraposée du théorème 3.2 en page 47.

Définition 3.6 (Fusion (fusion_{an}))**.** Soient $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}, \mathcal{N})$ des Frappes de Processus avec arcs neutralisants. On note $\mathsf{fusion}_{an}(\mathcal{PH}) = (\Sigma, \mathcal{L}, \mathcal{H})$ les Frappes de Processus standards appelées *fusion* de $\mathcal{PH}$, dont on a retiré les arcs neutralisants.

Théorème 3.4 (Points fixes des Frappes de Processus avec arcs neutralisants)**.** *Soient $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}, \mathcal{N})$ des Frappes de Processus avec arcs neutralisants, et $r \in \mathcal{L}$:*

$$\exists s \in \mathcal{L}, r \rightarrow_{\mathcal{PH}} s \Longrightarrow \exists s' \in \mathcal{L}, r \rightarrow_{\mathsf{fusion}_{an}(\mathcal{PH})} s'$$

Démonstration. Si une action est jouable dans r, cela signifie notamment que son frappeur et sa cible sont présents dans r. Cette condition est suffisante pour que cette même action soit jouable dans $\mathsf{fusion}_{an}(\mathcal{PH})$. □

Pour obtenir l'ensemble exact des points fixes d'un modèle de Frappes de Processus avec arcs neutralisants, il est nécessaire de prendre en compte les cycles d'actions qui se neutralisent entre elles, car ces cas de figure rajoutent des points fixes. En effet, si plusieurs actions sont actives mais se neutralisent de façon circulaire (par exemple, $h_1 \rtimes h_2$, $h_2 \rtimes h_3$ et $h_3 \rtimes h_1$) alors aucune des trois ne sera jouable dans le modèle avec arcs neutralisants, mais elles le seront dans le modèle fusionné. Nous intuitons que l'ensemble des points fixes d'un modèle de Trappes de Processus avec arcs neutralisants est l'ensemble des points fixes de son modèle fusionné entremêlés avec les cycles d'arcs neutralisants existants ; autrement dit, ces cycles d'arcs neutralisants ajoutent des points fixes partiels qui peuvent être combinés à d'autres points fixes partiels sur le reste du modèle, trouvés à l'aide du graphe sans-frappes. Cependant, nous ne démontrons pas ce résultat.

Pour obtenir l'ensemble des points fixes d'un modèle de Frappes de Processus avec arcs neutralisants de façon certaine, il est possible d'aplatir celui-ci comme décrit à la section 4.2.2, afin de produire un modèle équivalent de Frappes de Processus avec 2 classes de priorités. La recherche de points fixes est alors rendu possible à l'aide de la méthode proposée à la section 3.1.3.1, et le résultat est transposable au modèle original. En effet, le modèle aplati possédant la même dynamique, il est possible de retrouver les points fixes du modèle d'origine en supprimant simplement les sortes coopératives des résultats.

3.2.3.2 Analyse statique

De même qu'à la section 3.1.3.2, étant donné que les Frappes de Processus avec arcs neutralisants permettent de restreindre la dynamique par rapport aux Frappes de Processus standards, il est toujours possible d'utiliser l'analyse statique par sur-approximation, mais l'analyse statique par sous-approximation n'est plus valable.

Cependant, nous proposons au chapitre 4 une traduction des Frappes de Processus avec arcs neutralisants en Frappes de Processus canoniques, qui sont des Frappes de

Processus avec 2 classes de priorités avec des contraintes précises, ainsi qu'une nouvelle approche d'analyse statique par sous-approximation qui s'applique à cette classe particulière de modèles. L'analyse statique par sous-approximation est donc possible sur les Frappes de Processus avec arcs neutralisants, au prix de cette traduction.

3.2.3.3 Paramètres stochastiques

Enfin, à l'instar de la section 3.1.3.3, il n'est pas possible d'utiliser directement les outils développés pour la simulation stochastique car ceux-ci ne prennent pas en compte la présence d'arcs neutralisants. Malgré cela, une telle utilisation reste théoriquement possible à condition de prendre en compte les neutralisations entre actions.

Cependant, à nouveau, un parallèle intéressant peut être tracé entre les arcs neutralisants et l'introduction de paramètres stochastiques dans le modèle. En effet, étant donné un ensemble de paramètres stochastiques, si les intervalles de tir de deux actions sont disjoints, alors il est possible de modéliser cela par un arc neutralisant dont l'action bloquante est l'action la plus « rapide » (c'est-à-dire dont l'intervalle est le plus proche de zéro) et l'action bloquée est la plus « lente » (c'est-à-dire dont l'intervalle est le plus éloigné de zéro).

Cette représentation est avantageuse par rapport à celle proposée à la section 3.1.3.3. En effet, il n'est pas nécessaire pour créer le modèle de distinguer des classes globales d'actions selon leurs intervalles de tir, mais seulement de déterminer si les intervalles de tir de toutes les actions sont, deux à deux, disjoints (voire assez éloignés selon des critères qui peuvent être fixés arbitrairement). Cela permet de plus d'obtenir des relations plus fines entre actions, là où les Frappes de Processus ne permettent que la distinction de classes de priorités globales, qui sont parfois trop grossières pour certaines modélisations. Cependant, le problème de la représentation de l'accumulation persiste : une partie du modèle évoluant « plus rapidement » peut totalement préempter une action « plus lente » si celle-ci est sans cesse neutralisée. Or en pratique, un tel cas devrait en définitive autoriser l'action « lente » à s'exécuter après un certain temps. Cela peut être corrigé en supprimant quelques arcs neutralisants bien choisis, ou encore en ne permettant pas la création d'arcs neutralisants entre les parties indépendantes du modèle, mais cela nécessite une analyse préalable assez poussée de la dynamique du système.

3.3 Frappes de Processus avec actions plurielles

Il peut être intéressant de vouloir représenter un système au niveau de ses réactions biochimiques, c'est-à-dire des réactions entre les différents composants présents. De telles réactions peuvent avoir différentes formes (transformation, complexation, dissociation...), et il est fréquent qu'elles fassent intervenir plusieurs réactifs et plusieurs produits. Biocham (Fages, Soliman & Chabrier-Rivier, 2004) propose par exemple de modéliser un tel système de réactions biochimiques à l'aide d'un ensemble de règles de réaction de la forme : $X \xrightarrow{Y} Z$, ou encore : $X + Y \rightarrow Y + Z$, où X est un ensemble de réactifs, Y un ensemble de catalyseurs et Z un ensemble de produits.

Les *Frappes de Processus avec actions plurielles* permettent de représenter de telles réactions mettant en jeu un nombre arbitraire de réactifs, de produits et de catalyseurs. Ainsi, une réaction de la forme : $X \xrightarrow{Y} Z$ peut être représentée à l'aide de l'action $A \rightarrowtail B$ où A et B sont deux ensembles des processus, A regroupant tous les processus représentant

les composants nécessaires à initier la réaction, et B tous les processus qui ont évolué pendant la réaction. Une telle action peut donc être jouée dans un état contenant tous les processus de A et fait évoluer celui-ci vers un état contenant tous les processus de B, les autres processus restant inchangés. Cela implique toutefois que pour tout processus de B, il existe un autre processus de la même sorte dans A. Les Frappes de Processus avec actions plurielles permettent donc de représenter un nombre arbitraire de bonds simultanés — autrement dit, de changements simultanés de processus actifs — déclenchés par un nombre arbitraire de prérequis — sous la forme de processus actifs dans l'état courant. Les figures 3.3 et 3.4, plus loin dans cette section, représentent des exemples de Frappes de Processus avec actions plurielles.

Un parallèle peut être tracé d'une part entre A et l'ensemble des réactifs et catalyseurs, et d'autre part entre B et l'ensemble des produits. Cependant, la modélisation par Frappes de Processus avec actions plurielles nécessite aussi de donner explicitement les composants qui sont absents. Par exemple, une réaction de complexation du type : $x + y \to c$ sera représentée en Frappes de Processus avec actions plurielles à l'aide de trois sortes x, y et c contenant chacune deux processus et représentant respectivement les deux réactifs et le complexe produit, et par l'action $\{x_1, y_1, c_0\} \rightarrowtail \{x_0, y_0, c_1\}$. Autrement dit, il est nécessaire de décomposer chaque élément en fonction de sa présence (x_1 et y_1) ou de son absence (c_0) au début comme à la fin de la réaction, et pas uniquement d'indiquer les composants présents en tant que réactifs ou produits.

On note ainsi qu'une réaction de la forme $\{a_0, b_0, c_0\} \rightarrowtail \{a_1, b_1\}$ ne peut pas être jouée si a ou b est déjà au niveau 1, comme c'est le cas par exemple dans l'état $\langle a_1, b_0, c_0 \rangle$. Un tel comportement a du sens lorsque les différents processus d'une sorte (a_0 et a_1, par exemple) représentent différents états d'une même molécule : la réaction ne peut alors pas être jouée pour des raisons de stœchiométrie. Cependant, si ces différents processus représentent plutôt des niveaux de concentration (a_1 représentant par exemple un niveau de concentration de la molécule a plus élevé que a_0), cette restriction n'a plus de sens car une plus forte concentration d'une des entités ne devrait pas empêcher la réaction d'avoir lieu et de produire la seconde entité. Cela peut néanmoins être corrigé en ajoutant les actions $\{a_1, b_0, c_0\} \rightarrowtail \{a_1, b_1\}$ et $\{a_0, b_1, c_0\} \rightarrowtail \{a_1, b_1\}$, ou encore en séparant la production de a_1 et de b_1 en deux actions (ou ensemble d'actions) distinctes.

Cette forme des Frappes de Processus peut être aisément représentée à l'aide d'un réseau d'automates synchronisés, car chaque sorte possède un rôle similaire à celui d'un automate et chaque action pouvant être remplacée par un ensemble de transitions étiquetées avec le même libellé. La section 5.3 en page 110 formalise cette équivalence. On peut aussi la représenter à l'aide de Frappes de Processus avec 4 classes de priorités, comme détaillé à la section 3.3.2 ; cependant, cette représentation a l'inconvénient d'être moins claire car faisant intervenir un nombre important d'actions et de sortes supplémentaires.

3.3.1 Définition

La définition 3.7 formalise la notion de Frappes de Processus avec actions plurielles, en accord avec la discussion informelle ci-dessus : une action plurielle est constituée de deux ensembles de processus de sortes distinctes, qui représentent l'ensemble des frappeurs et celui des bonds. Cela permet de formaliser le déclenchement d'une action par une synchronisation exacte entre un nombre arbitraire de frappeurs, et une synchronisation exacte entre plusieurs bonds lorsque l'action est jouée. Nous définissons de plus l'opérateur

de jouabilité des Frappes de Processus avec actions plurielles à la définition 3.8, afin de formaliser la dynamique de ce type de modèles à l'aide de la sémantique donnée à la définition 2.14 en page 29.

Définition 3.7 (Frappes de Processus avec actions plurielles)**.** Les *Frappes de Processus avec actions plurielles* sont définies par un triplet $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$, où :

- $\Sigma \stackrel{def}{=} \{a, b, \ldots\}$ est l'ensemble fini et dénombrable des *sortes* ;
- $\mathcal{L} \stackrel{def}{=} \bigotimes_{a\in\Sigma} \mathcal{L}_a$ est l'ensemble fini des *états*, où $\mathcal{L}_a = \{a_0, \ldots, a_{l_a}\}$ est l'ensemble fini et dénombrable des *processus* de la sorte $a \in \Sigma$ et $l_a \in \mathbb{N}^*$, chaque processus appartenant à une unique sorte : $\forall(a_i\,;b_j) \in \mathcal{L}_a \times \mathcal{L}_b, a \neq b \Rightarrow a_i \neq b_j$;
- $\mathcal{H} \stackrel{def}{=} \{A \rightarrowtail B \mid A, B \in \mathbf{Proc}^{\Diamond} \setminus \emptyset \wedge \forall q \in B, \exists p \in A, (p \neq q \wedge \mathsf{sorte}(p) = \mathsf{sorte}(q))\}$ est l'ensemble fini des *actions*.

Pour toute action $h = A \rightarrowtail B \in \mathcal{H}$, A est appelé le *frappeur* et B le *bond* de h, et on note : $\mathsf{frappeur}(h) = A$, $\mathsf{bond}(h) = B$. On appelle par ailleurs *frappeurs invariants* les processus de A qui ne sont pas désactivés par l'action plurielle (autrement dit, tels qu'il n'existe pas d'autre processus de la même sorte dans B) et on appelle *cibles* les autres processus de A (pour lesquels il existe un autre processus de la même sorte dans B). On note en conséquence : $\mathsf{cible}(h) = \{p \in A \mid \exists q \in B, \mathsf{sorte}(p) = \mathsf{sorte}(q)\}$, et $\mathsf{invariant}(h) = \{p \in A \mid \mathsf{sorte}(p) \notin \mathsf{sortes}(B)\}$.

Définition 3.8 (Opérateur de jouabilité ($F_{plur} : \mathcal{H} \to F$))**.** L'opérateur de jouabilité des Frappes de Processus avec actions plurielles est défini par :

$$\forall h \in \mathcal{H}, F_{plur}(h) \equiv \bigwedge_{p\in\mathsf{frappeur}(h)} p \ .$$

Modélisation des coopérations

Nous notons que les Frappes de Processus avec actions plurielles proposent une alternative très efficace à la représentation des coopérations entre actions. Une coopération est représentée par une sorte coopérative en Frappes de Processus standards, qui peut présenter un défaut de modélisation provoquant des décalages temporels, voire l'apparition de « faux états », comme expliqué à la section 2.2.2 en page 30. De plus, cela nécessite l'utilisation d'un grand nombre d'actions pour assurer la mise à jour du processus actif de la sorte coopérative.

Les Frappes de Processus avec classes de priorités permettent de pallier ce défaut, comme cela sera expliqué à la section 4.1 en page 67. Cependant, cette solution ne règle pas le problème du grand nombre d'actions utilisée pour la mise à jour de la sorte coopérative, qui peuvent nuire à la lecture du modèle. Les Frappes de Processus avec actions plurielles peuvent alors se révéler utiles pour la représentation des processus de coopération. En effet, il suffit d'une action plurielle pour définir une coopération entre processus, à condition de lui attribuer les frappeurs adéquats. Ainsi, on peut obtenir une coopération depuis un modèle de Frappes de Processus standards en remplaçant :

- toute action de la forme $a_i \to b_j \,\vec{\cap}\, b_k$, où a n'est pas une sorte coopérative, par une action plurielle $\{a_i, b_j\} \rightarrowtail \{b_k\}$;
- toute action de la forme $f_\sigma \to b_j \,\vec{\cap}\, b_k$, où f est une sorte coopérative, par une action plurielle $(\widetilde{\sigma} \cup \{b_j\}) \rightarrowtail b_k$.

Le modèle obtenu corrige ainsi le problème de décalage temporel propre aux sortes coopératives, tout en permettant une meilleur lisibilité car chaque chaque sorte coopérative et toutes ses actions sont remplacées par une seule action plurielle.

Exemple. On peut prendre en exemple la sorte coopérative $f\,c$ des Frappes de Processus standards de la figure 2.6. Le modèle corrigé en Frappes de Processus avec actions plurielles est donné à la figure 3.3, et contient notamment l'ensemble d'actions plurielles suivant :

$$\begin{aligned}\mathcal{H} = \{ \quad & \{c_1, a_1\} \rightarrowtail \{a_0\} \quad , \qquad \{c_0, f_1, a_0\} \rightarrowtail \{a_1\} \quad , \\ & \{c_1\} \rightarrowtail \{c_0\} \quad , \qquad \{f_1, c_0\} \rightarrowtail \{c_1\} \quad , \\ & \{f_0, c_1\} \rightarrowtail \{c_0\} \quad , \qquad \{f_1\} \rightarrowtail \{f_0\} \qquad \}\end{aligned}$$

Il a été obtenu en reprenant toutes les actions dont le frappeur et la cible n'étaient pas des processus de sortes coopératives, et en remplaçant la sorte coopérative $f\,c$ et ses actions par l'action plurielle : $\{c_0, f_1, a_0\} \rightarrowtail \{a_1\}$, selon le procédé décrit à la page précédente.

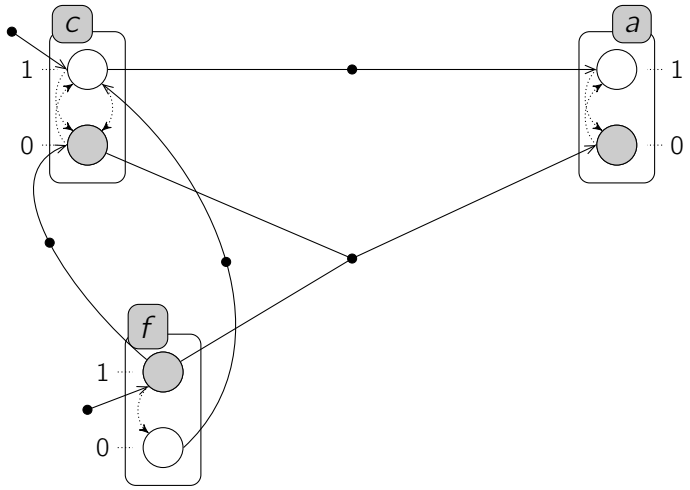


Figure 3.3 – Exemple de Frappes de Processus avec actions plurielles. Chaque action plurielle est représentée par plusieurs arcs et plusieurs couples de flèches, partant tous d'un même point. Les arcs indiquent les frappeurs invariants tandis que les flèches pleines indiquent les cibles, et sont suivies par une flèche en pointillés pour indiquer le bond correspondant. Par exemple, l'action plurielle $\{c_1, a_1\} \rightarrowtail \{a_0\}$ est représentée par un point d'où partent un arc vers c_1 et un couple de flèches vers a_1 puis vers a_0. De même, l'action plurielle $\{c_1\} \rightarrowtail \{c_0\}$ n'est représentée que par un couple de flèches vers c_1 puis vers c_0, car elle ne possède pas de frappeur invariant.

3.3.2 Traduction vers les Frappes de Processus avec 4 classes de priorités

Nous proposons dans cette section une traduction des Frappes de Processus avec actions plurielles en Frappes de Processus avec 4 classes de priorités (définition 3.9) et nous montrons que les modèles obtenus de cette façon sont faiblement bisimilaires (théorème 3.5). Les figures 3.4 et 3.5 illustrent cette traduction.

La traduction proposée permet d'obtenir un modèle de Frappes de Processus pseudo-canoniques avec 4 classes de priorités, telles que définies à la section 4.1 en page 67. De

façon informelle, ce modèle comporte 4 classes de priorités et repose sur l'utilisation de deux types de sortes particulières :

- des sortes coopératives pour vérifier la présence de tous les processus du frappeur,
- des *sortes de réaction* permettant de modéliser le déclenchement d'une *réaction*, ou son arrêt.

Une réaction modélise le fait qu'un ensemble d'actions (standards) est en train de simuler le jeu d'une action plurielle. À toute action plurielle h du modèle d'origine correspond une sorte coopérative f^h entre les sortes des processus de A et une sorte de réaction r^h dans le modèle résultant de cette traduction. La sorte coopérative comporte notamment un processus f^h_π qui représente le sous-état où tous les processus de A sont présents ; une action de priorité 4 de la forme $f^h_\pi \to r^h_0 \Rsh r^h_1$ permet d'activer la sorte de réaction. Une auto-action de priorité 3 de la forme $r^h_1 \to r^h_1 \Rsh r^h_0$ permet de plus la désactivation de la sorte de réaction une fois que toutes les actions de priorité 2 auront été jouées. Les actions de priorité 2 ont la forme $r^h_1 \to b_j \Rsh b_k$ avec $b_j \in A$ et $b_k \in B$, ce qui permet d'effectuer tous les bonds nécessaires à l'activation des processus de B. Enfin, les sortes coopératives sont toutes mises à jour par des actions de priorité 1, afin d'éviter les problèmes d'entrelacement et de correspondre à la définition de Frappes de Processus pseudo-canoniques telle que donnée par la définition 4.7 en page 70. L'agencement de ces classes de priorités permet ainsi de simuler des actions plurielles tout en empêchant l'entrelacement entre réactions — car deux réactions ayant lieu en même temps pourraient potentiellement amener le système dans un état normalement inaccessible.

Définition 3.9. Soient $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ des Frappes de Processus avec actions plurielles, et $(\!|\mathcal{PH}|\!) = (\Sigma'\,;\mathcal{L}'\,;\mathcal{H}'^{\langle 4\rangle})$ leur traduction en Frappes de Processus avec 4 classes de priorités, où :

- $\Sigma' = \Sigma \cup \{r^h \mid h \in \mathcal{H}\} \cup \{f^h \mid h \in \mathcal{H}\}$;
- $\mathcal{L}' = \mathcal{L} \times \left(\bigotimes_{h\in\mathcal{H}} \mathcal{L}_{r^h}\right) \times \left(\bigotimes_{h\in\mathcal{H}} \mathcal{L}_{f^h}\right)$, où :
  - $\forall h \in \mathcal{H}, \mathcal{L}_{r^h} = \{r^h_0, r^h_1\}$,
  - $\forall h \in \mathcal{H}, \mathcal{L}_{f^h} = \{f^h_\sigma \mid \sigma \in \mathcal{L}^{\Diamond}_{\mathsf{sortes}(\mathsf{frappeur}(h))}\}$;
- $\mathcal{H}'^{\langle 4\rangle} = (\mathcal{H}'^{(1)}\,;\mathcal{H}'^{(2)}\,;\mathcal{H}'^{(3)}\,;\mathcal{H}'^{(4)})$, où :
  - $\mathcal{H}'^{(1)} = \{a_i \to f^h_\sigma \Rsh f^h_{\sigma'} \mid h \in \mathcal{H} \wedge a \in \mathsf{sortes}(\mathsf{frappeur}(h)) \wedge a_i \in \mathcal{L}_a \wedge f^h_\sigma, f^h_{\sigma'} \in \mathcal{L}_{f^h} \wedge \sigma[a] \neq a_i \wedge \sigma' = \sigma \Cap a_i\}$,
  - $\mathcal{H}'^{(2)} = \{r^h_1 \to b_j \Rsh b_k \mid h \in \mathcal{H} \wedge b \in \mathsf{sortes}(\mathsf{cible}(h)) \wedge b_j, b_k \in \mathcal{L}_b \wedge b_j \in \mathsf{cible}(h) \wedge b_k \in \mathsf{bond}(h)\}$,
  - $\mathcal{H}'^{(3)} = \{r^h_1 \to r^h_1 \Rsh r^h_0 \mid h \in \mathcal{H}\}$,
  - $\mathcal{H}'^{(4)} = \{f^h_\pi \to r^h_0 \Rsh r^h_1 \mid h \in \mathcal{H} \wedge f^h_\pi \in \mathcal{L}_{f^h} \wedge \mathsf{frappeur}(h) \subseteq \pi\}$.

Pour tout état $s \in \mathcal{L}$ de $\mathcal{PH}$, on note $(\!|s|\!)$ l'état correspondant dans $(\!|\mathcal{PH}|\!)$:

- $\forall a \in \Sigma, (\!|s|\!)[a] = s[a]$,
- $\forall h \in \mathcal{H}, (\!|s|\!)[r^h] = r^h_0$,
- $\forall h \in \mathcal{H}, (\!|s|\!)[f^h] = f^h_\sigma$, tel que $\forall a \in \mathsf{sortes}(\mathsf{frappeur}(h)), \sigma[a] = (\!|s|\!)[a]$.

À l'inverse, pour tout état $s' \in \mathcal{L}'$ de $(\!|\mathcal{PH}|\!)$, on note $((s'))$ l'état correspondant dans $\mathcal{PH}$: $\forall a \in \Sigma, ((s'))[a] = s'[a]$.

Théorème 3.5 ($\mathcal{PH} \approx (\!|\mathcal{PH}|\!)$)**.** *Soient $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ des Frappes de Processus avec actions plurielles, et posons : $(\!|\mathcal{PH}|\!) = (\Sigma'\,;\mathcal{L}'\,;\mathcal{H}'^{\langle 4\rangle})$. On a :*

$$\forall s, s' \in \mathcal{L}, \exists h \in \mathcal{H}, s' = s \cdot h \iff \exists \delta \in \mathbf{Sce}((\!|s|\!)), \Big((\!|s'|\!) = (\!|s|\!) \cdot \delta \wedge |\widetilde{\delta} \cap \mathcal{H}'^{(4)}| = 1\Big) \quad .$$

Démonstration. Posons $(\!|\mathcal{PH}|\!) = (\Sigma'\,;\mathcal{L}'\,;\mathcal{H}'^{\langle 4\rangle})$. Soient $s, s' \in \mathcal{L}$.

($\Rightarrow$) On suppose qu'il existe une action $h \in \mathcal{H}$ telle que $s' = s \cdot h$. D'après la définition 3.9, cela implique notamment l'existence de sortes r^h et f^h dans Σ', et des actions suivantes :

- $\mathcal{H}'^{(1)}_h = \{a_i \to f^h_\sigma \Rsh f^h_{\sigma'} \mid a \in \mathsf{sortes}(\mathsf{frappeur}(h)) \wedge a_i \in \mathcal{L}_a \wedge f^h_\sigma, f^h_{\sigma'} \in \mathcal{L}_{f^h} \wedge \sigma[a] \neq a_i \wedge \sigma' = \sigma \Cap a_i\} \subset \mathcal{H}'^{(1)}$,
- $\mathcal{H}'^{(2)}_h = \{r^h_1 \to b_j \Rsh b_k \mid b \in \mathsf{sortes}(\mathsf{bond}(h)) \wedge b_j, b_k \in \mathcal{L}_b \wedge b_j \in \mathsf{frappeur}(h) \wedge b_k \in \mathsf{cible}(h)\} \subset \mathcal{H}'^{(2)}$,
- $h_3 = r^h_1 \to r^h_1 \Rsh r^h_0 \in \mathcal{H}'^{(3)}$,
- $h_4 = f^h_\pi \to r^h_0 \Rsh r^h_1 \in \mathcal{H}'^{(4)}$ avec π tel que $[F_{plur}(h)]\,(\pi)$.

Comme h est jouable dans s, alors $\mathsf{frappeur}(h) \subseteq s$, d'où $f^h_\pi \in (\!|s|\!)$, avec $\widetilde{\pi} = \mathsf{frappeur}(h)$. Ainsi, $f^h_\pi \to r^h_0 \Rsh r^h_1$ est jouable dans $(\!|s|\!)$. Toutes les actions de $\mathcal{H}'^{(2)}_h$ sont jouables dans $(\!|s|\!) \cdot h_4$. Elles peuvent être jouées successivement et alternativement avec des actions de $\mathcal{H}'^{(1)}$ car celles-ci modifient uniquement l'état de sortes coopératives, ce qui n'influe donc pas sur la jouabilité des actions de $\mathcal{H}'^{(2)}_h$. Soit $\mathcal{H}'^{(2)}_h = \{h^i_2\}_{i\in[\![1;n]\!]}$ un étiquetage des actions de $\mathcal{H}'^{(2)}_h$ avec $n = |\mathcal{H}'^{(2)}_h|$. On peut jouer depuis l'état $(\!|s|\!) \cdot h_4$ une séquence d'actions de la forme : $\delta_h = h^1_2 :: \delta^1 :: h^2_2 :: \delta^2 :: \ldots :: h^n_2 :: \delta^n$ où toutes les séquences δ^i avec $i \in [\![1\,;n]\!]$ sont des séquences d'actions de $\mathcal{H}^{(1)}$ mettant à jour des sortes coopératives. Après avoir joué cette séquence, le modèle se trouve dans un état $(\!|s|\!) \cdot h_4 \cdot \delta_h$ tel que : $\forall h_i \in \mathcal{H}'^{(2)}_h, \mathsf{bond}(h_i) \in ((\!|s|\!) \cdot h_4 \cdot \delta_h)$, c'est-à-dire : $\mathsf{bond}(h) \subseteq ((\!|s|\!) \cdot h_4 \cdot \delta_h)$. De plus, toutes les sortes coopératives sont mises à jour, ce qui signifie qu'aucune action de $\mathcal{H}'^{(1)}_h \cup \mathcal{H}'^{(2)}_h$ n'est plus jouable. Il est donc possible de jouer pour finir l'auto-action h_3, et on a : $(\!|s|\!) \cdot h_4 \cdot \delta_h \cdot h_3 = (\!|s'|\!)$, avec : $\widetilde{\delta} \cap \mathcal{H}'^{(4)} = \{h_4\}$.

($\Leftarrow$) On suppose qu'il existe un scénario $\delta \in \mathbf{Sce}((\!|s|\!))$ tel que $(\!|s'|\!) = (\!|s|\!) \cdot \delta$ et $|\widetilde{\delta} \cap \mathcal{H}'^{(4)}| = 1$. D'après la forme de $(\!|\mathcal{PH}|\!)$ (définition 3.9), et en s'inspirant du raisonnement précédent, on constate que la seule action jouable dans $(\!|s|\!)$ est une action de $\mathcal{H}'^{(4)}$, puis que dans l'état résultat, on ne peut jouer qu'une alternance entre une action de $\mathcal{H}'^{(2)}$ et des actions de $\mathcal{H}'^{(1)}$, et que l'état résultant une fois toutes les actions de $\mathcal{H}'^{(1)} \cup \mathcal{H}'^{(2)}$ jouées ne permet que le jeu d'une action de $\mathcal{H}'^{(3)}$. On en déduit que ce scénario a nécessairement la forme suivante : $\delta = h_4 :: \delta_{12} :: h_3$, avec $h_4 \in \mathcal{H}'^{(4)}$, $h_3 \in \mathcal{H}'^{(3)}$, et $\delta_{12} = h^1_2 :: \delta^1 :: h^2_2 :: \delta^2 :: \ldots :: h^n_2 :: \delta^n$ où pour tout $i \in [\![1\,;n]\!]$, $h^i_2 \in \mathcal{H}'^{(2)}$ et $\delta^i \in \mathbf{Sce}$ est un scénario composé uniquement d'actions de $\mathcal{H}'^{(1)}$. De plus, les seules actions jouables dans $(\!|s|\!) \cdot \delta$ sont à nouveau des actions de $\mathcal{H}'^{(4)}$, donc δ ne peut pas être plus grand car $|\widetilde{\delta} \cap \mathcal{H}'^{(4)}| = 1$. Posons :

- $\mathcal{H}'^{(1)}_\delta = \widetilde{\delta} \cap \mathcal{H}'^{(1)}$,
- $\mathcal{H}'^{(2)}_\delta = \widetilde{\delta} \cap \mathcal{H}'^{(2)} = \{h^i_2 \mid i \in [\![1\,;n]\!]\}$,
- $h_3 = r_1 \to r_1 \Rsh r_0$ et
- $h_4 = f_\pi \to r_0 \Rsh r_1$.

D'après la construction de $(\!|\mathcal{PH}|\!)$ donnée à la définition 3.9, il existe donc nécessairement une action $h \in \mathcal{H}$ dont r est la sorte de réaction et f est la sorte coopérative correspondante. De plus, toujours grâce à cette définition, on a : $\pi \in (\!|\mathcal{PH}|\!)^{\Diamond}$ et $\mathsf{frappeur}(h) \subseteq \pi$. Autrement dit : $\mathsf{frappeur}(h) = \widetilde{\pi}$, donc que h est jouable dans s, par définition de $(\!|s|\!)$. De plus, toujours d'après la définition 3.9, on a : $\forall i \in [\![1\,;n]\!], h_2^i = r_1 \to b_j^i \Rsh b_k^i \wedge b_j^i \in \mathsf{frappeur}(h) \wedge b_k \in \mathsf{bond}(h)$. Ainsi, $\mathsf{bond}(h) = \{b_k = \mathsf{bond}(h_2^i) \mid i \in [\![1\,;n]\!]\}$. Or on constate immédiatement que toutes les sortes coopératives sont nécessairement mises à jour dans $s \cdot \delta$ car la dernière action jouée est une action de $\mathcal{H}'^{(3)}$ et que son jeu ne rend aucune action de $\mathcal{H}'^{(1)}$ jouable. De plus, $s \cdot \delta[r] = r_0$ et $(\!|s|\!) \cdot \delta = (\!|s|\!) \Cap \{\mathsf{bond}(h_2^i) \mid i \in [\![1\,;n]\!]\}$. Ainsi, $(\!|s'|\!) = (\!|s|\!) \cdot \delta = (\!|s|\!) \Cap \mathsf{bond}(h) = (\!|s \Cap \mathsf{bond}(h)|\!) = (\!|s \cdot h|\!)$. □

Exemple. Afin d'illustrer la traduction définie dans cette section, nous nous intéressons aux Frappes de Processus avec actions plurielles $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ suivantes, représentées à la figure 3.4 :

- $\Sigma = a, b, c$,
- $\mathcal{L}_a = \{a_0, a_0\} \quad , \quad \mathcal{L}_b = \{b_0, b_0\} \quad , \quad \mathcal{L}_c = \{c_0, c_0\}$,
- $\mathcal{H} = \{\{a_1, b_1, c_1\} \rightarrowtail \{b_0, c_0\}\}$.

On appelle $h = \{a_1, b_1, c_1\} \rightarrowtail \{b_0, c_0\}$ l'unique action que comporte ce modèle. Celle-ci représente un cas intéressant d'action plurielle, car elle comporte plusieurs cibles (et donc plusieurs bonds). En effet : $\mathsf{cible}(h) = \{b_1, c_1\}$; par ailleurs : $\mathsf{invariant}(h) = \{a_1\}$.

La définition 3.9 nous permet de traduire ces Frappes de Processus avec actions plurielles en un modèle de Frappes de Processus avec 4 classes de priorités $(\!|\mathcal{PH}|\!) = (\Sigma'\,;\mathcal{L}'\,;\mathcal{H}'^{\langle 4\rangle})$ qui est présenté à la figure 3.5. Ce modèles comporte cinq sortes : $\Sigma' = \{a, b, c, r^h, f^h\}$, et notamment les classes d'actions suivantes :

$$\mathcal{H}'^{(2)} = \{ \quad r_1^h \to b_1 \Rsh b_0 \quad , \qquad r_1^h \to c_1 \Rsh c_0 \quad \}$$
$$\mathcal{H}'^{(3)} = \{ \quad r_1^h \to r_1^h \Rsh r_0^h \quad \} \qquad \mathcal{H}'^{(4)} = \{ \quad f_7^h \to r_0^h \Rsh r_1^h \quad \}$$

3.3.3 Équivalence avec les Frappes de Processus avec k classes de priorités

Cette section permet de formaliser l'équivalence d'expressivité entre les Frappes de Processus avec actions plurielles d'une part, et les Frappes de Processus avec k classes de priorités d'autres part, si $k \in \mathbb{N}^{\bullet}$. Le théorème 3.6 formalise cette conclusion, bien que sa démonstration s'appuie sur des résultats qui seront présentés à la section 4.2.4 en page 78. En effet, elle s'appuie sur plusieurs résultats plus forts qui stipulent que des Frappes de Processus avec actions plurielles et les Frappes de Processus avec k classes de priorités peuvent toujours être représentées par une classe particulière de Frappes de Processus avec 2 classes de priorités, qui elle-même peut aussi être représentée par des Frappes de Processus avec actions plurielles.

Théorème 3.6 (Équivalence)**.** *Pour tout $k \in \mathbb{N}^{\bullet}$, les Frappes de Processus avec actions plurielles et les Frappes de Processus avec k classes de priorités ont une expressivité équivalente.*

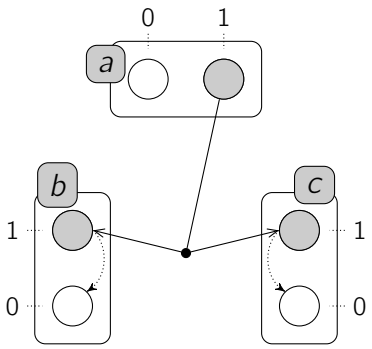


Figure 3.4 – Exemple de Frappes de Processus avec actions plurielles. L'unique action de ce modèle est $\{a_1, b_1, c_1\} \rightarrowtail \{b_0, c_0\}$ et comporte plusieurs cibles (b_1 et c_1).

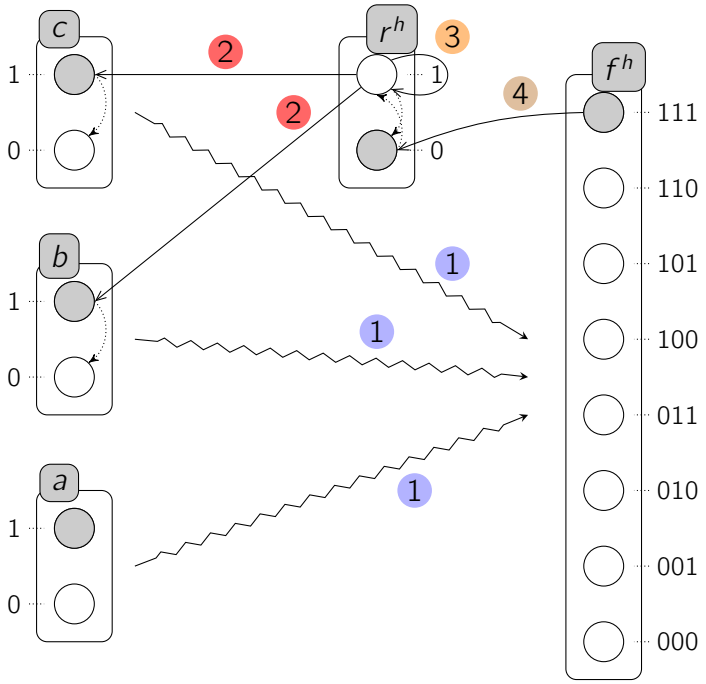


Figure 3.5 – Traduction des Frappes de Processus avec actions plurielles de la figure 3.4 en Frappes de Processus avec 4 classes de priorités. Le sortes a, b et c sont identiques à celles du modèle d'origine. Les deux autres sortes ont été ajoutées pour les besoins de la traduction de l'action h. r^h est une sorte de réaction (la présence de r_1^h modélise le fait que les actions représentant h sont en train d'être jouées) et f^h est une sorte coopérative (permettant de vérifier la présence de tous les frappeurs).

Démonstration. Soit $k \in \mathbb{N}^\bullet$. D'après le théorème 3.5 en page 60, il est toujours possible de traduire des Frappes de Processus avec actions plurielles en Frappes de Processus avec 4 classes de priorités, et celles-ci peuvent être à leur tour traduites en Frappes de Processus avec 2 classes de priorités d'après le théorème 4.1 en page 75. Or des Frappes de Processus avec 2 classes de priorités sont à fortiori des Frappes de Processus avec k classes de priorités. Inversement, toutes Frappes de Processus avec k classes de priorités peuvent être aplaties en Frappes de Processus canoniques toujours d'après le théorème 4.1, qui elles-mêmes peuvent être traduites en Frappes de Processus avec action plurielles d'après le théorème 4.2 en page 79. □

3.3.4 Réutilisation des résultats existants

Nous passons rapidement en revue dans la suite différentes méthodes permettant de réutiliser certains résultats des Frappes de Processus standards sur les Frappes de Processus avec actions plurielles. Nous ne détaillons pas ces méthodes qui reprennent principalement des résultats traités à d'autres endroits de cette thèse.

3.3.4.1 Points fixes

Du fait de la complexité des actions plurielles, il n'existe pas à l'heure actuelle de méthode générique efficace permettant de déduire les points fixes de Frappes de Processus avec actions plurielles. Cependant, à l'instar des Frappes de Processus avec arcs neutralisants (voir section 3.2.3.1 en page 53) il est possible d'obtenir ces points fixes en les cherchant sur le modèle aplati (tel que défini à la section 4.2.3 en page 78). En effet, sa dynamique étant équivalente, aux sortes coopératives près, le résultat peut être transposé au modèle d'origine.

3.3.4.2 Analyse statique

Du fait de la forme particulière des actions, les méthodes d'analyse statique ne s'appliquent pas directement aux Frappes de Processus avec actions plurielles. Cependant, moyennant l'utilisation de la traduction vers des Frappes de Processus avec 4 classes de priorités donnée à la définition 3.9, il est possible de les appliquer sur un modèle équivalent.

3.3.4.3 Paramètres stochastiques

Il est théoriquement possible d'associer des paramètres stochastiques à chaque action plurielle d'un modèle de Frappes de Processus avec actions plurielles. Un tel ajout aurait notamment l'avantage d'éviter l'utilisation de la valeur « infinie » d'absorption de stochasticité, qui avait principalement pour but de simuler des actions successives instantanées, afin notamment de pallier le fait qu'un seul processus ne peut évoluer pour chaque jeu d'action des Frappes de Processus standards.

Une autre alternative consisterait à utiliser l'aplatissement de la section 4.2.3, et d'attribuer aux actions correspondant à des activations de réaction dans le modèle obtenu les paramètres stochastiques voulus.

3.4 Bilan

Nous avons proposé dans ce chapitre trois modélisations alternatives des Frappes de Processus. L'ajout de classes de priorités (section 3.1) permet de définir des contraintes de préemption globales entre les ensembles d'action d'un modèle tandis que les arcs neutralisants (section 3.2) permettent plus de finesse en définissant des relations locale de préemption entre des couples d'actions. À l'inverse, l'introduction d'actions plurielles (section 3.3) permet de représenter des relations de synchronisme entre actions.

Nous traçons par ailleurs des liens entre ces formalismes, en montrant notamment que le synchronisme des actions plurielles et la granularité des arcs neutralisants peuvent tous deux être décrits à l'aide de classes de priorités, bien que cela rende le modèle plus complexe. Ces traductions auront néanmoins un intérêt particulier dans le chapitre suivant, qui se base sur le formalisme des Frappes de Processus avec 2 classes de priorités pour définir une classe de modèles particulière, dont la dynamique peut être analysée efficacement.

Chapitre 4

Représentation canonique pour l'analyse des Frappes de Processus

Une analyse statique efficace des questions d'atteignabilité avait précédemment été développée sur les Frappes de Processus standards par Paulevé et al. (2012). Elle ne peut cependant pas être appliquée aux formalismes alternatifs de Frappes de Processus présentés au chapitre 3, qui possèdent une expressivité supérieure. L'objectif de ce chapitre est de présenter les travaux ayant permis de développer une nouvelle forme de sous-approximation pour l'analyse statique. Celle-ci s'applique à une classe restreinte de Frappes de Processus avec 2 classes de priorités, appelées *Frappes de Processus canoniques*. Nous montrons par ailleurs que toutes les sémantiques alternatives de Frappes de Processus introduites au chapitre 3 peuvent être traduites en Frappes de Processus canoniques, prouvant d'une part la large expressivité de cette classe de modèles, et permettant d'autre part d'appliquer à tous ces formalismes la sous-approximation que nous avons développée dans le cadre de cette thèse et qui est présentée dans la suite.

Ces résultats ont été publiés dans (Folschette, Paulevé, Magnin & Roux, 2013), dont une version étendue fait l'objet d'une soumission en journal en cours de *review*.

Les Frappes de Processus standards (définition 2.8 en page 25) sont très atomiques, et leurs actions ont une forme particulière : un processus peut déclencher le bond d'un autre processus dans une autre sorte. Autrement dit, le changement du processus actif d'une sorte est conditionné par la présence d'au plus un autre processus actif. Dans le cas d'une auto-action $b_j \to b_j \not\Rightarrow b_k$, par exemple, le frappeur et la cible b_j sont confondus ; il n'y a donc formellement aucun prérequis pour que b_j bondisse vers b_k. Dans le cas plus général d'une action $a_i \to b_j \not\Rightarrow b_k$, où $a_i \neq b_j$, le seul prérequis pour que b_j bondisse en b_k est la présence de a_i.

Cette restriction sur la forme des actions permet une analyse efficace des questions d'atteignabilité au sein des Frappes de Processus standards, qui s'expriment de la façon suivante :

« Partant d'un état donné, est-il possible, en jouant un nombre quelconque d'actions, d'atteindre un état dans lequel un processus donné est actif ? »

En termes plus formels, l'atteignabilité d'un processus b_k depuis un état s se définit comme l'existence d'un scénario $\delta \in \mathbf{Sce}(s)$, c'est-à-dire une séquence d'actions jouables dans s (cf. définition 2.14 en page 29) tel que $(s \cdot \delta)[b] = b_k$,

En effet, comme expliqué plus haut, la forme particulière des actions impose au plus un prérequis pour pouvoir faire bondir le processus actif d'une sorte vers un autre niveau. Partant de ce constat, une analyse statique permettant de répondre efficacement aux questions d'atteignabilité avait été précédemment développée par Paulevé et al. (2012). Si on considère l'exemple de l'atteignabilité d'un processus b_k depuis un état s tel que $s[b] = b_j$, avec $b_j \neq b_k$, la méthode utilisée repose sur l'analyse de la dynamique locale de b, ou, autrement dit, sur l'analyse des bonds entre processus de b produits par les actions frappant cette sorte. En particulier, si on reprend les deux exemples d'actions donnés ci-dessus,

- s'il existe une auto-action $b_j \to b_j \Rsh b_k$, alors la réponse est immédiate car b_k est accessible en jouant cette action qui n'a pas de prérequis ;
- s'il existe une action $a_i \to b_j \Rsh b_k$, alors l'atteignabilité de b_k est conditionnée par l'atteignabilité de a_i — à la condition près que cette seconde atteignabilité laisse b_j intact.

Dans le cas général, faire bondir un processus de b_j à b_k peut en fait nécessiter plusieurs actions, ou aucune si le processus est déjà présent dans l'état initial. Ainsi, l'atteignabilité d'un processus peut nécessiter l'atteignabilité de plusieurs autres processus, ou d'aucun autre processus. Cette approche permet donc de résoudre un problème d'atteignabilité de façon récursive, les cas terminaux étant ceux pour lesquels la résolution ne nécessite aucune action, ou ne nécessite que des auto-actions.

Cependant, cette méthode d'analyse statique ne s'applique pas totalement aux nouveau formalismes de Frappes de Processus présentés au chapitre 3. En effet, ces formalismes augmentent l'expressivité des Frappes de Processus, ce qui invalide, au moins en partie, la méthode développée. Nous proposons dans ce chapitre une nouvelle approche par analyse statique qui permet de répondre à des questions d'atteignabilité sur une classe particulière des Frappes de Processus avec 2 classes de priorités, appelées *Frappes de Processus canoniques*. La méthode que nous proposons possède une efficacité proche de la méthode d'origine, le calcul du résultat étant de complexité polynomiale selon le nombre de sortes dans le modèle. Cela permet notamment de traiter de grands modèles efficacement ; en effet, l'implémentation qui en a été fait répond toujours en moins d'une seconde sur des modèles comportant une centaine de composants, comme nous le montrerons par la suite au chapitre 6.

Les Frappes de Processus canoniques ne comportent que 2 classes de priorités ainsi qu'une restriction dans l'utilisation des actions de priorité 1 (c'est-à-dire les plus prioritaires) : celles-ci ne doivent servir qu'à mettre à jour des sortes coopératives, et tous les autres types d'actions doivent donc être de priorité 2. Ce formalisme permet ainsi de représenter de façon simple les modèles de Thomas (avec paramètres) ou les modèles booléens (avec portes logiques), à l'aide de traductions de complexité polynomiale. Nous montrons de plus qu'elles forment une classe de modèles qui est aussi expressive que toutes les Frappes de Processus avec k classes de priorités, les Frappes de Processus avec arcs neutralisants et les Frappes de Processus avec actions plurielles. Nous donnons par ailleurs

les traductions appropriées pour pouvoir aussi traiter ces modèles, qui peuvent cependant s'avérer avoir une complexité exponentielle selon certaines caractéristiques des modèles. Le positionnement des Frappes de Processus canoniques par rapport aux différents autres enrichissements de Frappes de Processus est résumé à la figure 1.1 en page 11.

Nous définissons dans un premier temps les Frappes de Processus canoniques de façon formelle dans la section 4.1 et donne un certain nombre de résultats à leur propos. La section 4.2 aborde ensuite la question du lien entre les Frappes de Processus canoniques et les autres formalismes présentés au chapitre 3, et propose les traductions nécessaires. Pour terminer, nous présentons formellement les méthodes d'analyse statiques développées spécifiquement pour les Frappes de Processus canoniques à la section 4.3.

Les travaux présentés dans ce chapitre ont été publiés dans (Folschette, Paulevé, Magnin & Roux, 2013), qui a été sélectionné pour la soumission d'un article étendu en journal.

4.1 Les Frappes de Processus canoniques

Nous donnons dans cette section la définition des Frappes de Processus canoniques, qui sont un sous-ensemble des Frappes de Processus avec classes de priorités (section 4.1.1). Un modèle de Frappes de Processus est dit *canonique* s'il comporte 2 classes de priorités et qu'il respecte un certain nombre de critères concernant l'utilisation des actions prioritaires. Il faut notamment que celles-ci soient uniquement utilisées pour la mise à jour des sortes coopératives, et que cette mise à jour soit correctement formée, c'est-à-dire qu'elle ne comporte pas d'actions en trop ou manquantes. Un certain nombre de résultats peuvent être déduits de la forme particulière de ces Frappes de Processus (section 4.1.2). La figure 4.1 en page 71 propose un exemple simple de Frappes de Processus canoniques.

4.1.1 Critères et définitions

Nous donnons dans la suite les définitions et critères permettant de caractériser entièrement les Frappes de Processus canoniques (définition 4.8). Cependant, afin de proposer un cadre assez général et permettre une réutilisation des éléments de cette section, nous considérerons dans un premier temps le cas de Frappes de Processus avec k classes de priorités dans les définitions et critères qui suivent, pour $k \in \mathbb{N}^\bullet$. Cela permettra aussi de définir les Frappes de Processus pseudo-canoniques (définition 4.7), qui forment un ensemble plus général ayant des propriétés proches.

Nous définissons pour commencer la *réduction* d'un modèle donné de Frappes de Processus comme le modèle équivalent dont on a retiré toutes les actions qui ne sont pas de priorité 1 (définition 4.1). Cette définition permet dans la suite de considérer le comportement des actions les plus prioritaires d'un modèle afin d'en contraindre la forme. Il est à noter cependant que cette réduction provoque une perte d'informations, et n'est donc pas comparable aux réductions de modèles comme (Naldi et al., 2009).

Définition 4.1 (Réduction ($\mathcal{PH}^1$))**.** Si $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H}^{\langle k\rangle})$ sont des Frappes de Processus avec k classes de priorités, on note $\mathcal{PH}^1 = (\Sigma\,;\mathcal{L}\,;\mathcal{H}^{(1)})$ la *réduction* de $\mathcal{PH}$. $\mathcal{PH}^1$ est un modèle de Frappes de Processus standard (ou encore : avec 1 classe de priorité). Si $s \in \mathcal{L}$, on note de plus : $\mathbf{Sce}^1(s)$ l'ensemble des scénarios depuis l'état s dans les Frappes de Processus $\mathcal{PH}^1$.

Dans la suite, nous appelons *actions primaires* les actions de l'ensemble $\mathcal{H}^{(1)}$, c'est-à-dire les actions les plus prioritaires. Ces actions auront pour unique but de mettre à jour des sortes coopératives, permettant à celles-ci de modéliser des portes logiques sans décalage temporel, comme expliqué en page 70. De fait, on peut dans certains cas considérer ces actions comme « non biologiques », ou « instantanées », car elles sont présentes principalement pour des raisons de modélisation. À l'inverse, les actions de l'ensemble $\mathcal{H} \setminus \mathcal{H}^{(1)}$, qui ne sont pas de priorité 1, seront appelées *actions secondaires*. Comme elles permettent de représenter les différentes réactions et régulations intervenan au sein du système, elles peuvent par conséquent être qualifiées d'actions « biologiques » ou « lentes ».

Exemple. Nous prendrons comme exemple, tout au long de cette section, les Frappes de Processus avec 2 classes de priorités représentées à la figure 4.1, qui sera étudié plus en détail en page 70 mais permettra entre-temps d'illustrer certaines définitions. Nous constatons de prime abord que les actions primaires de ce modèle permettent uniquement de mettre à jour la sorte coopérative *ab*, tandis que les actions secondaires frappent les autres composants.

La définition de la forme canonique des Frappes de Processus débute avec le critère 4.1 qui stipule que la dynamique ne doit pas contenir de séquence infinie d'actions primaires. En effet, pour que ces actions effectuent une mise à jour des sortes coopératives sans perturber le reste du système, il est nécessaire qu'elles ne puissent pas préempter les actions secondaires indéfiniment. Sans cela, le modèle pourrait exhiber un comportement dit de Zénon (Zhang, Johansson, Lygeros & Sastry, 2001), où une suite infinie d'actions peut être jouée en un temps nul et ainsi bloquer l'évolution du système. Il est donc nécessaire de postuler que tous les scénarios d'actions primaires sont finis, ce qui a par ailleurs pour conséquence que $\mathbf{Sce}^1$ est un ensemble fini.

Critère 4.1 (Terminaison finie)**.** La dynamique de $\mathcal{PH}^1$ ne contient aucun cycle : $\exists N \in \mathbb{N}, \forall s \in \mathcal{L}, \forall \delta \in \mathbf{Sce}^1(s), |\delta| \leq N$.

Étant donnée une sorte $a \in \Sigma$ et un état $s \in \mathcal{L}$, on note $\mathsf{focals}^1_s(a)$ (définition 4.2) l'ensemble des processus de la sorte a qui peuvent être présents après avoir joué tous les scénarios d'actions de priorité 1 depuis l'état s. Cet ensemble est toujours défini du fait du critère 4.1.

Définition 4.2 ($\mathsf{focals}^1 : \mathcal{L} \times \Sigma \to \wp(\mathbf{Proc})$)**.** Pour tout état $s \in \mathcal{L}$ et pour toute sorte $a \in \Sigma$,

$$\mathsf{focals}^1_s(a) = \{(s \cdot \delta)[a] \in \mathcal{L}_a \mid \delta \in \mathbf{Sce}^1(s) \wedge \nexists h \in \mathcal{H}^{(1)}, (\delta :: h) \in \mathbf{Sce}^1(s)\}$$

Nous définissons dans la suite la notion de *composant bien formé* (définition 4.3) et de *sorte coopérative bien formée* (définition 4.5). Un composant bien formé n'est frappé que par des actions secondaires, car il ne subit que des influences « biologiques ». Une sorte coopérative bien formée n'est frappée que par des actions primaires « non biologiques » qui convergent toujours vers le même processus en fonction de l'état des sortes qui l'influencent (définition 4.4), afin qu'elle représente chaque configuration de ces sortes par un unique processus. Le critère 4.2 stipule alors que l'ensemble des sortes des Frappes de Processus canoniques (Σ) est une partition entre un ensemble de composants bien formés (noté Γ) et un ensemble de sortes coopératives bien formées (noté Δ).

Définition 4.3 (Composant bien formé (Γ))**.** Une sorte $a \in \Sigma$ est un *composant bien formé* si et seulement si :

$$\forall h \in \mathcal{H}, \mathsf{sorte}(\mathsf{cible}(h)) = a \Rightarrow \mathsf{prio}(h) \geq 2 \ .$$

On note Γ l'ensemble des composants bien formés du modèle.

Définition 4.4 (Influence ($\mathsf{comp}^1 : \Sigma \to \wp(\Gamma)$))**.** Pour toute sorte $a \in \Sigma$, on définit : $\mathsf{comp}^1(a) \stackrel{def}{=} (\mathsf{conn}^1(a) \cap \Gamma) \setminus \{a\}$ où $\mathsf{conn}^1(a) \subset \Sigma$ est le plus petit ensemble de sortes satisfaisant :

$$a \in \mathsf{conn}^1(a)$$
$$\forall h \in \mathcal{H}^{(1)}, \mathsf{sorte}(\mathsf{cible}(h)) \in \mathsf{conn}^1(a) \Rightarrow \mathsf{sorte}(\mathsf{frappeur}(h)) \in \mathsf{conn}^1(a)$$

Définition 4.5 (Sorte coopérative bien formée (Δ))**.** Une sorte $a \in \Sigma$ est une *sorte coopérative bien formée* si et seulement si :

(i) $\forall h \in \mathcal{H}, \mathsf{sorte}(\mathsf{cible}(h)) = a \Rightarrow \mathsf{prio}(h) = 1$,

(ii) $\forall s \in \mathcal{L}, |\mathsf{focals}^1_s(a)| = 1$,

(iii) $\forall a_i \in \mathcal{L}_a, \exists s \in \mathcal{L}, a_i \in \mathsf{focals}^1_s(a)$,

(iv) $\forall \sigma \in \mathcal{L}^{\Diamond}_{\mathsf{comp}^1(a)}, \forall s, s' \in \mathcal{L}, (\sigma \subseteq s \wedge \sigma \subseteq s') \Rightarrow \mathsf{focals}^1_s(a) = \mathsf{focals}^1_{s'}(a)$.

On note Δ l'ensemble des sortes coopératives bien formées du modèle.

Exemple. Le modèle de Frappes de Processus avec 2 classes de priorités de la figure 4.1 possède trois composants bien formés : $\Gamma = \{a, b, c\}$ et une sorte coopérative bien formée : $\Delta = \{ab\}$. On en déduit notamment :

$$\mathsf{conn}^1(c) = \{c\} \quad \text{et} \quad \mathsf{conn}^1(ab) = \{a, b, ab\} \ ,$$
$$\text{d'où :} \quad \mathsf{comp}^1(c) = \{c\} \quad \text{et} \quad \mathsf{comp}^1(ab) = \{a, b\} \ .$$

D'après $\mathsf{comp}^1(ab) = \{a, b\}$, on peut donc considérer que la sorte coopérative ab modélise une coopération entre a et b.

Critère 4.2 (Partition des composants et des sortes coopératives)**.** Γ et Δ forment une partition de Σ :

$$\Sigma = \Gamma \cup \Delta \wedge \Gamma \cap \Delta = \emptyset$$

Exemple. Le modèle de la figure 4.1 vérifie le critère 4.2.

Il est à noter qu'une sorte a qui n'est frappée par aucune action, c'est-à-dire telle que $\forall h \in \mathcal{H}, \mathsf{sorte}(\mathsf{cible}(h)) \neq a$, est en accord à la fois avec la définition de composant bien formé et avec celle de sorte coopérative bien formée. Cette sorte peut être arbitrairement et indifféremment affectée à Γ ou à Δ. Un tel cas peut se produire pour des composants ayant un rôle d'« entrée », c'est-à-dire dont le niveau d'expression est uniquement déterminé par l'état initial, ou par certaines constructions particulières de sortes coopératives créées par la traduction depuis les réseaux discrets asynchrones proposée à la section 5.1 en page 92.

Pour toute sorte $a \in \Sigma$ et tout état $s \in \mathcal{L}$, étant donné le point (ii) de la définition 4.5, on a toujours : $\exists a_i \in \mathcal{L}_a, \mathsf{focals}^1_s(a) = \{a_i\}$. On notera donc dans la suite : $\mathsf{focals}^1_s(a) = a_i$. De plus, grâce au point (iii) de la définition 4.5, on peut introduire la notation $\mathsf{procState}(a_i)$ qui permet de caractériser l'ensemble des sous-états représentés par le processus a_i de toute sorte coopérative bien formée a (définition 4.6).

Définition 4.6 ($\mathsf{procState} : \mathbf{Proc} \to \wp(\mathbf{Proc})$)**.** Pour tout $a \in \Delta$ et $a_i \in \mathcal{L}_a$,

$$\mathsf{procState}(a_i) \stackrel{def}{=} \{\widetilde{ps} \mid ps \in \mathcal{L}^{\Diamond}_{\mathsf{comp}^1(a)} \wedge \exists s \in \mathcal{L}, (ps \subseteq s \wedge \mathsf{focals}^1_s(a) = a_i)\}$$

La notation $\widetilde{ps}$, qui permet de considérer un tuple comme un ensemble, a été définie à la section 1.6 en page 12. Dans la suite, on écrira simplement « composant » (resp. « sorte coopérative ») en lieu et place de « composant bien formé » (resp. « sorte coopérative bien formée »).

Enfin, la définition 4.8 définit la notion de *Frappes de Processus canoniques*. La forme particulière de ces Frappes de Processus permettra de développer une méthode efficace d'analyse statique pour le problème d'atteignabilité défini à la section 4.3, grâce notamment aux résultats développés dans la section suivante. La définition 4.7 offre de plus une définition plus étendue du concept de *Frappes de Processus pseudo-canoniques avec k classes de priorités*, dont le nombre de classes de priorités n'est pas limité à 2, mais dont la classe de priorité la plus forte possède les mêmes restrictions. Cette définition étendue sera notamment utile pour des questions de traduction, à la section 4.2.1.

Définition 4.7 (Frappes de Processus pseudo-canoniques)**.** Si $k \in \mathbb{N}^{\bullet}$, les *Frappes de Processus pseudo-canoniques avec k classes de priorités* sont les Frappes de Processus avec k classes de priorités respectant les critères 4.1 et 4.2.

Définition 4.8 (Frappes de Processus canoniques)**.** Les *Frappes de Processus canoniques* sont les Frappes de Processus pseudo-canoniques avec 2 classes de priorités (c'est-à-dire respectant les critères 4.1 et 4.2).

Exemple. Les Frappes de Processus de la figure 4.1 sont des Frappes de Processus canoniques, car elles respectent les critères 4.1 et 4.2. Ce modèle décrit un comportement de coopération entre les deux processus a_1 et b_1. En effet, l'action $ab_{11} \to c_0 \Rsh c_1$ permet de représenter fidèlement la coopération entre a_1 et b_1 pour frapper c_0 et le faire bondir en c_1. De plus, il est possible de bondir en a_1 ou en b_1 si ces deux processus sont absents grâce aux actions $b_0 \to a_0 \Rsh a_1$ et $a_0 \to b_0 \Rsh b_1$; autrement dit, a_1 (resp. b_1) est accessible uniquement lorsque b_0 (resp. a_0) est actif. Enfin, a et b s'auto-inhibent grâce aux actions $a_1 \to a_1 \Rsh a_0$ et $b_1 \to b_1 \Rsh b_0$.

Comparons les Frappes de Processus canoniques $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$ de la figure 4.1 avec leur version fusionnée $\mathcal{PH}' = (\Sigma\,;\mathcal{L}\,;\mathcal{H}')$, c'est-à-dire $\mathcal{PH}' = \mathsf{fusion}(\mathcal{PH})$, et donc $\mathcal{H}' = \mathcal{H}^{(1)} \cup \mathcal{H}^{(2)}$, selon la définition 3.3 en page 46. En d'autres termes, $\mathcal{PH}'$ est la version sans priorités de $\mathcal{PH}$, c'est-à-dire où toutes les actions sont présentes mais où la notion de classes de priorités a été supprimée.

Intéressons-nous à la dynamique de $\mathcal{PH}'$. Comme expliqué à la section 2.2.2 en page 30, ce modèle comporte une sorte coopérative ab qui est définie de la façon dont le sont toutes les sortes coopératives en Frappes de Processus standards, faute de priorités. Cette version sans priorités introduit des comportements supplémentaires qui ne sont pas désirables pour représenter une porte logique exacte. En effet, on pourrait s'attendre à ce que ab_{11}

représente la présence simultanée de a et b dans un état passé, ou, en d'autres termes, que seule la présence simultanée de a_1 et b_1 puisse permettre l'activation de ab_{11}. Or on constate que ce n'est pas le cas, car le processus c_1 peut être activé depuis l'état initial $\langle a_0, b_0, c_0, ab_{00}\rangle$. Le scénario suivant permet par exemple d'activer ce processus, sans pour autant que a_1 et b_1 ne soient simultanément présents, et bien que le processus initial de ab soit cohérent avec ceux de a et b : $\langle a_1, b_0, c_0, ab_{10}\rangle \rightarrow_{\mathcal{PH}'} \langle a_0, b_0, c_0, ab_{10}\rangle \rightarrow_{\mathcal{PH}'} \langle a_0, b_1, c_0, ab_{10}\rangle \rightarrow_{\mathcal{PH}'} \langle a_0, b_1, c_0, ab_{11}\rangle$. En effet, il se trouve que l'activation de ab_{11} ne représente pas la présence simultanée de a_1 et b_1 dans un même état, mais leur présence consécutive dans différents états. En d'autres termes, la présence de ab_{11} dans un état n'implique pas $a_1 \wedge b_1$ dans le même état, ni même $\mathsf{P}(a_1 \wedge b_1)$, mais plutôt : $\mathsf{P}(a_1) \wedge \mathsf{P}(b_1)$ où P signifie : « précédemment ». On constate d'ailleurs que l'analyse statique développée dans (Paulevé et al., 2012) répond positivement quand à l'accessibilité de c_1 depuis l'état $\langle a_1, b_0, c_0, ab_{10}\rangle$.

Un tel comportement est indésirable si on souhaite utiliser les sortes coopératives en tant que portes logiques exactes, c'est-à-dire sans ce décalage temporel menant à une sur-approximation de la dynamique. Les Frappes de Processus canoniques permettent alors de modéliser un comportement (faiblement) bisimilaire au comportement d'un réseau discret asynchrone, comme montré à la section 5.1 en page 92, les actions primaires permettant la mise à jour sans délai des sortes coopératives, à l'instar du modèle $\mathcal{PH}$ de la figure 4.1. En effet, le processus ab_{11} est impossible à activer si a_1 et b_1 ne sont pas simultanément actifs, et ce cas ne peut se présenter que dans l'état initial. Il s'agit d'un jardin d'Eden dans le sens développé par Loïc (Paulevé, 2011, p. 123).

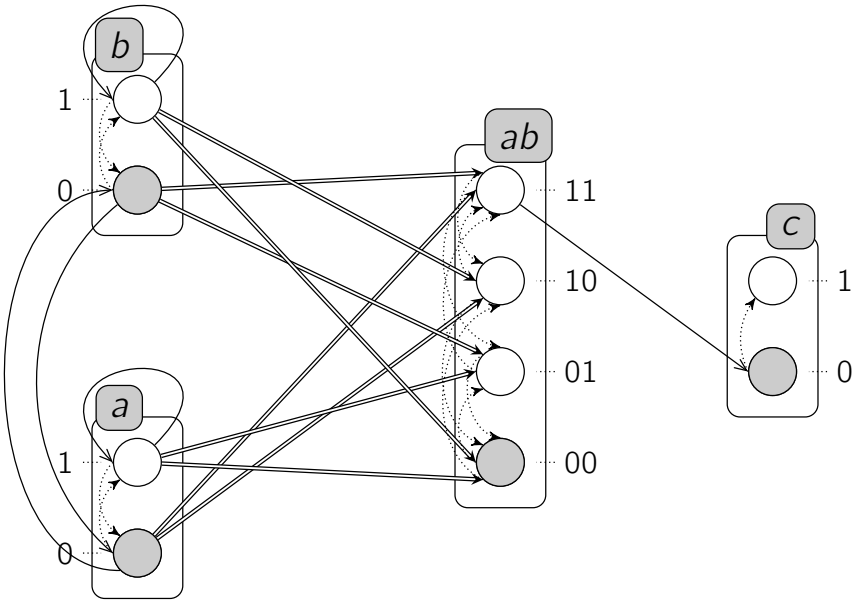


Figure 4.1 – Un exemple de Frappes de Processus canoniques. Les actions de priorité 1 sont dessinées en traits doubles tandis que les actions de priorité 2 sont représentées avec des traits simples. Les processus grisés présentent un exemple d'état de départ : $\langle a_0, b_0, c_0, ab_{00}\rangle$.

4.1.2 Résultats préliminaires sur les Frappes de Processus canoniques

Dans cette section, nous donnons plusieurs résultats généraux qui peuvent être dérivés des restrictions de la section 4.1.1. Nous considérons donc dans la suite des Frappes de Processus canoniques $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$. Ces résultats permettront de construire la méthode d'analyse statique de la section 4.3.

Pour tout état $s \in \mathcal{L}$, on appelle $\mathsf{update}(s)$ l'état dans lequel tous les composants ont le même processus actif que dans s mais où les sortes coopératives ont été mises à jour en fonction de l'état des sortes qui les influencent (définition 4.9). Cet état est unique du fait du point (ii) de la définition 4.5 en page 69. La notation « $\Cap$ » représente le recouvrement d'un état par ensemble de processus, qui avait été donné à la définition 2.11 en page 28. Le lemme 4.1 stipule ensuite que depuis tout état s, il existe un scénario d'actions primaires mettant à jour toutes les sortes coopératives de façon à arriver dans $\mathsf{update}(s)$.

Définition 4.9 ($\mathsf{update} : \mathcal{L} \to \mathcal{L}$)**.** Pour tout $s \in \mathcal{L}$, on définit :

$$\mathsf{update}(s) = s \Cap \{\mathsf{focals}^1_s(a) \mid a \in \Delta\} \ .$$

Lemme 4.1. $\forall s \in \mathcal{L}, \exists \delta \in \mathbf{Sce}^1(s), s \cdot \delta = \mathsf{update}(s)$

Démonstration. Soit $s \in \mathcal{L}$ un état. Soit $a \in \Delta$ une sorte coopérative telle que $s[a] \neq \mathsf{focals}^1_s(a)$. Étant donnée la définition 4.5, il existe un scénario δ mettant à jour a dans s tel que : $\forall \delta' \in \mathbf{Sce}^1(s \cdot \delta), (s \cdot \delta \cdot \delta')[a] = \mathsf{focals}^1_s(a)$. Ce raisonnement est applicable à chacune des sortes coopératives du modèle indifféremment de leur ordre. □

Le lemme 4.2 stipule que pour un état donné $s \in \mathcal{L}$, et pour toute action secondaire $h = a_i \to b_j \Rsh b_k \in \mathcal{H}$ où a et b sont des composants, si $s[a] = a_i$ et $s[b] = b_j$, alors h peut toujours être jouée après une série d'actions primaires, et ces actions ne perturbent pas cette jouabilité (autrement dit, elles ne modifient pas les processus a_i et b_j).

De façon complémentaire, le lemme 4.3 énonce le même résultat si a est une sorte coopérative, sous la condition que a soit déjà à jour dans s.

Lemme 4.2. $\forall s \in \mathcal{L}, \forall a, b \in \Gamma, \forall h = a_i \to b_j \Rsh b_k \in \mathcal{H}$,
$(s[a] = a_i \wedge s[b] = b_j) \Rightarrow (\exists \delta \in \mathbf{Sce}^1(s), [F_p(h)]\,(s \cdot \delta))$

Démonstration. Soient $s \in \mathcal{L}$ un état, $a, b \in \Gamma$ deux composants et $h = a_i \to b_j \Rsh b_k \in \mathcal{H}$ une action. D'après le lemme 4.1, il existe un scénario δ tel que : $(s \cdot \delta) = \mathsf{update}(s)$. Étant donné que a et b sont des composants, $a_i \in (s \cdot \delta)$ et $b_j \in (s \cdot \delta)$. De plus, par définition de $\mathsf{update}(s)$, toutes les sortes coopératives ont été mises à jour dans $(s \cdot \delta)$ et aucune action primaire n'y est donc jouable. Ainsi, h est jouable dans $(s \cdot \delta)$. □

Lemme 4.3. $\forall s \in \mathcal{L}, \forall a \in \Delta, \forall b \in \Gamma, \forall h = a_i \to b_j \Rsh b_k \in \mathcal{H}$,
$(s[a] = a_i \wedge s[b] = b_j \wedge \mathsf{focals}^1_s(a) = a_i) \Rightarrow (\exists \delta \in \mathbf{Sce}^1(s), [F_p(h)]\,(s \cdot \delta))$

Démonstration. Similaire à la preuve du lemme 4.2 ; étant donné que $a_i \in \mathsf{focals}^1_s(a)$, on a : $a_i \in (s \cdot \delta)$. □

4.2 Équivalence avec les autres formalismes de Frappes de Processus

Cette section vise à tracer des liens entre les différentes sémantiques des Frappes de Processus présentées au chapitre 3. Son principal apport est l'*aplatissement* des Frappes

de Processus avec k classes de priorités, donné à la section 4.2.1, et qui permet de les traduire en Frappes de Processus canoniques. Cette traduction permet donc, à partir d'un modèle de Frappes de Processus comprenant un nombre arbitraire de priorités, d'obtenir un modèle canonique respectant la même dynamique. Par la suite, le cas des Frappes de Processus avec arcs neutralisants est aussi traité, de façon analogue, dans la section 4.2.2. Enfin, la section 4.2.3 réutilise ces résultats pour proposer aussi une traduction depuis les Frappes de Processus avec actions plurielles.

Ces différentes traductions et propriétés de bisimulation (faible) qu'elles proposent permettent d'établir que les différentes sémantiques de Frappes de Processus sont aussi expressives que les Frappes de Processus canoniques. Notamment, cela nous permet d'assurer que les Frappes de Processus avec plusieurs classes de priorités sont équivalentes ; autrement dit, considérer plus de deux classes de priorités n'augmente pas l'expressivité — bien que cela puisse faciliter la modélisation.

4.2.1 Aplatissement des Frappes de Processus avec k classes de priorités

Le but de cette section est de montrer qu'un modèle de Frappes de Processus avec k classes de priorités peut être *aplati*, c'est-à-dire traduit en un autre modèle ne comportant que 2 classes de priorités. Ce dernier modèle est assuré de posséder la même dynamique, car il lui est faiblement bisimilaire, comme établi par le théorème 4.1 en page 75. De plus, les actions de priorité 1 (les plus prioritaires) ne sont utilisées que pour mettre à jour les sortes coopératives ; il s'agit en fait d'un modèle de Frappes de Processus canoniques telles que définies à la section 4.1.1. La forme particulière de ces modèles permet d'y appliquer les méthodes d'analyse statique développées à la section 4.3. Cela nous permet de plus de montrer que les Frappes de Processus avec k classes de priorités sont aussi expressives entre elles pour tout $k \in \mathbb{N}^*$, car elles sont toutes aussi expressives que les Frappes de Processus canoniques.

Étant donné que les propriétés de jouabilité n'utilisent que des opérateurs de logique booléenne standards, il est possible de calculer la forme normale disjonctive (FND) de toute propriété de jouabilité. Pour toute action $h \in \mathcal{H}$, cette FND est de la forme :

$$F_p(h) \equiv \bigvee_{i \in [\![1;n^h]\!]} \left(\bigwedge_{j \in [\![1;m^{h,i}]\!]} p_{i,j} \right)$$

où $n^h \in \mathbb{N}$ et $\forall i \in [\![1\,;n^h]\!]$, $m^{h,i} \in \mathbb{N}^*$. Si $n^h = 0$, alors $F_p(h) \equiv \bot$, ce qui signifie que l'action h ne peut jamais être jouée car elle est préemptée dans dans tous les états où son frappeur et sa cible sont présents. Une telle action peut être retirée du modèle sans en changer le comportement. En revanche, si $n^h > 0$, alors $F_p(h) \not\equiv \bot$; dans ce cas, $F_p(h)$ est une disjonction de n^h conjonctions d'atomes, et peut donc être vue comme une disjonction de n^h propriétés de jouabilité plus petites. Ces n^h conjonctions d'atomes peuvent être traduites en autant de sortes coopératives priorisées, afin d'obtenir une dynamique équivalente avec un nombre réduit de classes de priorités utilisées. Dans ce second cas, on note, pour tout $i \in [\![1\,;n^h]\!]$: $\mathsf{dep}^i(h) = \{\mathsf{sorte}(p_{i,j}) \mid j \in [\![1\,;m^{h,i}]\!]\}$.

L'opérateur d'aplatissement F_{plat} donné à la définition 4.10 permet de caractériser la jouabilité d'une action sans prendre en compte les actions primaires ; en d'autres termes, une action h est jouable dans un état s si et seulement si $[F_{plat}(h)]\,(s)$ et aucune action de

priorité 1 n'est jouable. Le lemme 4.4 permet alors de caractériser la jouabilité d'une action dans un état à l'aide d'un sous-état correspondant à l'une des conjonctions de sa propriété d'aplatissement une fois traduite en FND. Enfin, la définition 4.11 donne la construction de l'*aplatissement* de $\mathcal{PH}$: pour chaque action $h \in \mathcal{H}$, plusieurs sortes coopératives $f^{h,i}$ permettent de refléter chaque conjonction de $F_p(h)$, c'est-à-dire une pour chaque indice $i \in [\![1\,;n^h]\!]$. Cette construction permet d'obtenir la même dynamique que pour $\mathcal{PH}$ en reproduisant les préemptions possibles par d'autres actions plus prioritaires, comme établi par le théorème 4.1.

Il est à noter que la définition 4.11 ne s'applique qu'à des Frappes de Processus pseudo-canoniques. Cependant, des Frappes de Processus avec k classes de priorités quelconques sont *a fortiori* des Frappes de Processus pseudo-canoniques à condition d'ajouter une classe vide d'actions de priorité supérieure. En d'autres termes, il est possible d'ignorer le cas particulier des actions primaires (mettant à jour les sortes coopératives) si les Frappes de Processus ne sont pas pseudo-canoniques.

Définition 4.10 (Opérateur d'aplatissement ($F_{plat} : \mathcal{H} \to F$))**.** L'opérateur d'aplatissement des Frappes de Processus avec k classes de priorités est défini par :

$$\forall h \in \mathcal{H}, F_{plat}(h) \equiv \mathsf{frappeur}(h) \wedge \mathsf{cible}(h) \wedge \left(\bigwedge_{\substack{g \in \mathcal{H}^{(n)} \\ 1<n<\mathsf{prio}(h)}} \neg \left(\mathsf{frappeur}(g) \wedge \mathsf{cible}(g)\right) \right)$$

Lemme 4.4. *Soient $h \in \mathcal{H} \setminus \mathcal{H}^{(1)}$ et $s \in \mathcal{L}$.*

$$[F_p(h)]\,(s) \Leftrightarrow \left(\exists \sigma \subseteq s, [F_{plat}(h)]\,(\sigma)\right) \wedge \left(\forall g \in \mathcal{H}^{(1)}, [\neg F_p(g)]\,(s)\right) \ .$$

Démonstration. ($\Rightarrow$) Supposons $[F_p(h)]\,(s)$. Il n'existe donc aucune action primaire jouable dans s. Par ailleurs, $F_p(h) \Rightarrow F_{plat}(h)$ donc $F_{plat}(h) \not\equiv \bot$ et, par propriété d'une FND, au moins l'une des conjonctions de $F_{plat}(h)$ est vraie dans s. On suppose que la i^{e} conjonction est vraie dans s, avec $i \in [\![1\,;n^h]\!]$; on a alors : $\forall j \in [\![1\,;m^{h,i}]\!], p_{i,j} \in s$. Soit $\sigma \in \mathcal{L}^{\Diamond}_{\mathsf{dep}^i(h)}$ avec $\forall b \in \mathsf{dep}^i(h), \sigma[b] = s[b]$. On a alors immédiatement : $\sigma \subseteq s$, et, par construction de $\mathsf{dep}^i(h)$, $[F_{plat}(h)]\,(\sigma)$.

($\Leftarrow$) Supposons qu'il existe $\sigma \subseteq s$ tel que $[F_{plat}(h)]\,(\sigma)$, et qu'aucune action primaire n'est jouable dans s. On a alors immédiatement $[F_{plat}(h)]\,(s)$ car ajouter des processus au sous-état d'évaluation ne peut pas rendre la propriété fausse. De plus, comme aucune action primaire n'est jouable dans s, alors $\left[\left(\bigwedge_{g \in \mathcal{H}^{(1)}} \neg(\mathsf{frappeur}(g) \wedge \mathsf{cible}(g))\right)\right](s)$, donc $[F_p(h)]\,(s)$. □

Définition 4.11 (Aplatissement (flat))**.** Si $k \in \mathbb{N}^\bullet$ et $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H}^{\langle k\rangle})$ sont des Frappes de Processus pseudo-canoniques avec k classes de priorités, on note $\mathsf{flat}(\mathcal{PH}) = \overline{\mathcal{PH}} = (\overline{\Sigma}\,;\overline{\mathcal{L}}\,;(\overline{\mathcal{H}}^{(1)}\,;\overline{\mathcal{H}}^{(2)}))$ l'*aplatissement* de $\mathcal{PH}$, où :

- $\overline{\Sigma} = \Sigma \cup \Sigma_f$ où $\Sigma_f = \{f^{h,i} \mid h \in \mathcal{H} \wedge n^h \geq 1 \wedge i \in [\![1\,;n^h]\!]\}$;
- $\overline{\mathcal{L}} = \left(\bigotimes_{a\in\Sigma}\mathcal{L}_a\right) \times \left(\bigotimes_{f^{h,i}\in\Sigma_f}\mathcal{L}_{f^{h,i}}\right)$, où $\forall f^{h,i} \in \Sigma_f, \mathcal{L}_{f^{h,i}} = \{f_\sigma^{h,i} \mid \sigma \in \mathcal{L}^\Diamond_{\mathsf{dep}^i(h)}\}$;
- $\overline{\mathcal{H}}^{(1)} = \mathcal{H}^{(1)} \cup \{a_k \to f_\sigma^{h,i} \Rsh f_{\sigma'}^{h,i} \mid h \in \mathcal{H} \wedge f^{h,i} \in \Sigma_f \wedge a \in \mathsf{dep}^i(h) \wedge a_k \in \mathcal{L}_a \wedge f_\sigma^{h,i}, f_{\sigma'}^{h,i} \in \mathcal{L}_{f^{h,i}} \wedge \sigma[a] \neq a_k \wedge \sigma' = \sigma \Cap \{a_k\}\}$;
- $\overline{\mathcal{H}}^{(2)} = \{f_\sigma^{h,i} \to \mathsf{cible}(h) \Rsh \mathsf{bond}(h) \mid h \in \mathcal{H} \setminus \mathcal{H}^{(1)} \wedge f^{h,i} \in \Sigma_f \wedge f_\sigma^{h,i} \in \mathcal{L}_{f^{h,i}} \wedge [F_{plat}(h)]\,(\sigma)\}$.

De plus, étant donné un état $\overline{s} \in \overline{\mathcal{L}}$, on note $\lfloor\overline{s}\rfloor = s$ l'état correspondant dans $\mathcal{L}$: $\forall a \in \Sigma, s[a] = \overline{s}[a]$. À l'inverse, étant donné un état $s \in \mathcal{L}$, $\lceil s\rceil = \overline{s}$ est l'état correspondant dans $\overline{\mathcal{L}}$: $\forall a \in \Sigma, \overline{s}[a] = s[a]$ et $\forall f^{h,i} \in \Sigma_f, \overline{s}[f^{h,i}] = f_\sigma^{h,i}$ avec $f_\sigma^{h,i} \in \mathcal{L}_{f^{h,i}}$ et $\forall b \in \mathsf{dep}^i(h), \sigma[b] = s[b]$.

Nous notons que l'aplatissement $\mathsf{flat}(\mathcal{PH})$ de toutes Frappes de Processus avec k classes de priorités $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H}^{\langle k\rangle})$ sont des Frappes de Processus canoniques. En effet, une partie des sortes coopératives générées lors de cette traduction proviennent des Frappes de Processus d'origine, qui sont pseudo-canoniques et sont déjà contraintes de la même manière. L'autre partie constitue les sortes coopératives de l'ensemble Σ_f et leur définition respecte les critères 4.1 et 4.2.

Théorème 4.1 ($\mathcal{PH} \approx \mathsf{flat}(\mathcal{PH})$)**.** *Soient $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H}^{\langle k\rangle})$ des Frappes de Processus avec k classes de priorités, et $\overline{\mathcal{PH}} = \mathsf{flat}(\mathcal{PH}) = (\overline{\Sigma}\,;\overline{\mathcal{L}}\,;\overline{\mathcal{H}}^{\langle 2\rangle})$ leur aplatissement.*

1. *$\forall s, s' \in \mathcal{L}$, $s \to_{\mathcal{PH}} s' \implies \lceil s\rceil \rightsquigarrow_{\overline{\mathcal{PH}}} \lceil s'\rceil$, où $\rightsquigarrow_{\overline{\mathcal{PH}}}$ est une séquence finie de transitions $\to_{\overline{\mathcal{PH}}}$.*
2. *$\forall \overline{s}, \overline{s}' \in \overline{\mathcal{L}}$, $\overline{s} \to_{\overline{\mathcal{PH}}} \overline{s}' \implies \lfloor\overline{s}\rfloor = \lfloor\overline{s}'\rfloor \vee \lfloor\overline{s}\rfloor \to_{\mathcal{PH}} \lfloor\overline{s}'\rfloor$.*

Démonstration. (1) Soient $s, s' \in \mathcal{L}$ tels que $s \to_{\mathcal{PH}} s'$. Posons $\overline{s} = \lceil s\rceil$. D'après la dynamique des Frappes de Processus (définition 2.14), si $s \to_{\mathcal{PH}} s'$, alors il existe une action $h \in \mathcal{H}$ jouable dans s, telle que $s' = s \cdot h$. On a alors : $[F_p(h)]\,(s)$. Par construction de $\overline{\mathcal{PH}}$ (définition 4.11) :

- Si $h \in \mathcal{H}^{(1)}$, alors il existe $g = h \in \overline{\mathcal{H}}^{(1)}$, et on a : $\mathsf{frappeur}(g) \in \overline{s}$ et $\mathsf{cible}(g) \in \overline{s}$.
- Si, au contraire, $h \in \mathcal{H} \setminus \mathcal{H}^{(1)}$, alors il existe $g = f_\sigma^{h,i} \to \mathsf{cible}(h) \Rsh \mathsf{bond}(h) \in \overline{\mathcal{H}}^{(2)}$. De plus, d'après le lemme 4.4, il existe $\sigma \subseteq s$ tel que $[F_{plat}(h)]\,(\sigma)$ et, par construction de $\overline{s}$ (définition 4.11), $\overline{s}[f^{h,i}] = f_\sigma^{h,i}$.

Dans les deux cas, g est jouable dans $\overline{s}$. Par la suite, dans l'état $\overline{s} \cdot g$, les seules actions jouables sont celles dans $\mathcal{H}^{(1)}$ qui mettent à jour les sortes coopératives dans lesquelles $\mathsf{bond}(h) = \mathsf{bond}(g)$ est impliqué, directement ou indirectement, permettant donc d'accéder à l'état $\lceil s'\rceil$ en un nombre fini d'actions. Ainsi, $\lceil s\rceil \rightsquigarrow_{\overline{\mathcal{PH}}} \lceil s'\rceil$.

(2) soient $\overline{s}, \overline{s}' \in \overline{\mathcal{L}}$ tels que $\overline{s} \to_{\overline{\mathcal{PH}}} \overline{s}'$. Posons $s = \lfloor\overline{s}\rfloor$. D'après la dynamique des Frappes de Processus (définition 2.14), si $\overline{s} \to_{\mathcal{PH}} \overline{s}'$, alors il existe une action $g \in \overline{\mathcal{H}}$ jouable dans $\overline{s}$, telle que $\overline{s}' = \overline{s} \cdot h$.

- Si $g \in \overline{\mathcal{H}}^{(1)} \setminus \mathcal{H}^{(1)}$, alors $\lfloor\overline{s}\rfloor = \lfloor\overline{s}'\rfloor$ car seul le processus actif d'une sorte coopérative dans Σ_f a évolué.

- Sinon, si il existe $h = g \in \mathcal{H}^{(1)}$, alors h est jouable dans s et : $\lfloor \overline{s} \rfloor \rightarrow_{\mathcal{PH}} \lfloor \overline{s}' \rfloor$ avec $\lfloor \overline{s}' \rfloor = s \cdot h$.
- Autrement, si $g \in \overline{\mathcal{H}}^{(2)}$, on note : $g = f_{\sigma}^{h,i} \rightarrow b_j \Rsh b_k$. Par construction de l'aplatissement (définition 4.11), il existe $h \in \mathcal{H}$ tel que $[F_{plat}(h)](\sigma)$. Comme g est jouable, cela signifie qu'aucune action de $\mathcal{H}^{(1)}$ n'est jouable, et notamment que la sorte coopérative $f^{h,i}$ est déjà mise à jour, ce qui a pour conséquence que : $\sigma \subseteq s$. Ainsi, d'après lemme 4.4, h est jouable dans s, car $[F_p(h)](s)$, et $\lfloor \overline{s} \rfloor \rightarrow_{\mathcal{PH}} \lfloor \overline{s}' \rfloor$.

□

Nous notons que l'aplatissement donné à la définition 4.11 est, pour chaque action, exponentiel dans le nombre d'actions non primaires de priorité supérieure. En effet, il est nécessaire pour chaque action de convertir la propriété de jouabilité donnée par la définition 3.2 en une FND. Or la majorité des cas pratiques pour cette conversion sont proches du pire cas, qui est de complexité exponentielle en fonction du nombre d'atomes dans la propriété.

Pour finir, il est intéressant de noter que l'aplatissement de la définition 4.11 n'est pas optimal en nombre d'actions et de processus créés dans le modèle final. En effet, il est possible de simplifier le modèle $\mathsf{flat}(\mathcal{PH})$ de différentes manières, qui n'ont pas été prises en compte ici pour ne pas alourdir la définition. Nous donnons dans la suite quelques pistes pour obtenir des modèles plus simples mais au comportement équivalent, mais n'en donnons pas les preuves.

Simplification des propriétés d'aplatissement

La propriété d'aplatissement $F_{plat}(h)$ d'une action $h = a_i \rightarrow b_j \Rsh b_k$ peut être simplifiée à l'aide des propriétés suivantes, permettant d'éviter la création de certains éléments inutiles, comme des actions qui ne sont jamais jouables ou des coopérations qui sont toujours vraies :

- il n'est pas nécessaire de faire apparaître la cible d'une action dans sa propriété d'aplatissement car sa présence sera vérifiée par ailleurs, au moment du tir effectif de l'action, d'où : $b_j \equiv \top$;
- Tout processus $b_l \neq b_j$ de la même sorte que la cible empêche toujours la jouabilité de l'action, donc : $b_l \equiv \bot$;
- si c_p, c_q sont des processus différents ($c_p \neq c_q$) de la même sorte c, alors $c_p \wedge c_q \equiv \bot$.

Suppression des sortes coopératives superflues

Il existe deux cas pour lesquels il est possible de supprimer la sorte coopérative $f^{h,i}$ créée pour une action h dans le modèle aplati :

- si $F_{plat}(h) \equiv \top$, alors l'action h peut être traduite comme étant une auto-action (car elle est toujours jouable dès lors que la cible est présente) ;
- Si la i^{e} conjonction de $F_{plat}(h)$ consiste en un unique élément p, alors cette conjonction peut être traduite par une action simple de la forme : $p \rightarrow \mathsf{cible}(h) \Rsh \mathsf{bond}(h)$ sans avoir recours à une sorte coopérative (étant donné qu'en dehors de la cible, un seul processus, p, est requis).

Exemple. Les Frappes de Processus avec 3 classes de priorités de la figure 4.2 représentent un modèle de segmentation métazoaire inspiré de la figure 3.1 en page 45 donc les deux actions suivantes ont été retirées :

$$f_1 \to f_1 \not\Uparrow f_0 \quad \text{et} \quad f_0 \to c_1 \not\Uparrow c_0$$

Cela permet de s'intéresser uniquement à l'état stationnaire du système, qui consiste en l'oscillation alternative des sortes a et c. Ce modèle peut être aplati par la méthode donnée à la définition 4.11 en page 75 et les simplifications détaillées ci-dessus. Le résultat est le modèle de Frappes de Processus canoniques donné à la figure 4.3. Le modèle complet (de la figure 3.1) peut naturellement aussi être aplati à l'aide de la même méthode, mais le résultat est plus complexe du fait des deux actions « peu urgentes » supplémentaires.

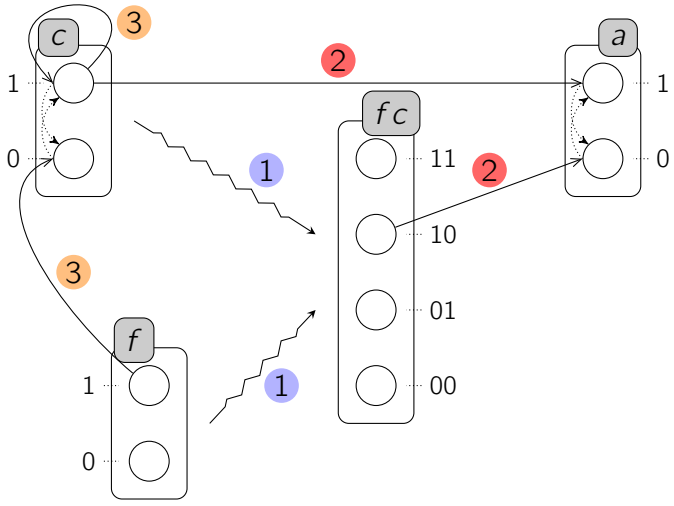


Figure 4.2 – Exemple de Frappes de Processus avec 3 classes de priorités, inspiré du modèle de la figure 2.6, dont deux actions ont été retirées pour supprimer l'interruption de l'avancée du front d'onde f. Les étiquettes numérotées (de 1 à 3) placées contre les flèches représentant les actions symbolisent leur appartenance à une classe de priorités donnée.

4.2.2 Aplatissement des Frappes de Processus avec arcs neutralisants

À l'instar de la section 4.2.1, il est possible de traduire les Frappes de Processus avec arcs neutralisants en Frappes de Processus canoniques. Le procédé est le même car il consiste, pour chaque action, en un calcul de propriété d'aplatissement qui est ici identique à la propriété de jouabilité. En effet, l'opérateur d'aplatissement de cette traduction est égal à l'opérateur de jouabilité donné à la définition 3.5 en page 51 : $F_{plat} = F_{an}$. Par la suite, une fois cette propriété traduite en FND, il est possible de réutiliser la traduction de la définition 4.11 afin d'obtenir un modèle canonique ayant la même dynamique, comme assuré par le théorème 4.1.

Cette traduction est elle aussi de complexité exponentielle dans le nombre d'actions préemptant chaque action. Cependant, on note que l'utilisation d'arcs neutralisants peut rendre cet aplatissement beaucoup plus efficace. En effet, contrairement aux Frappes de

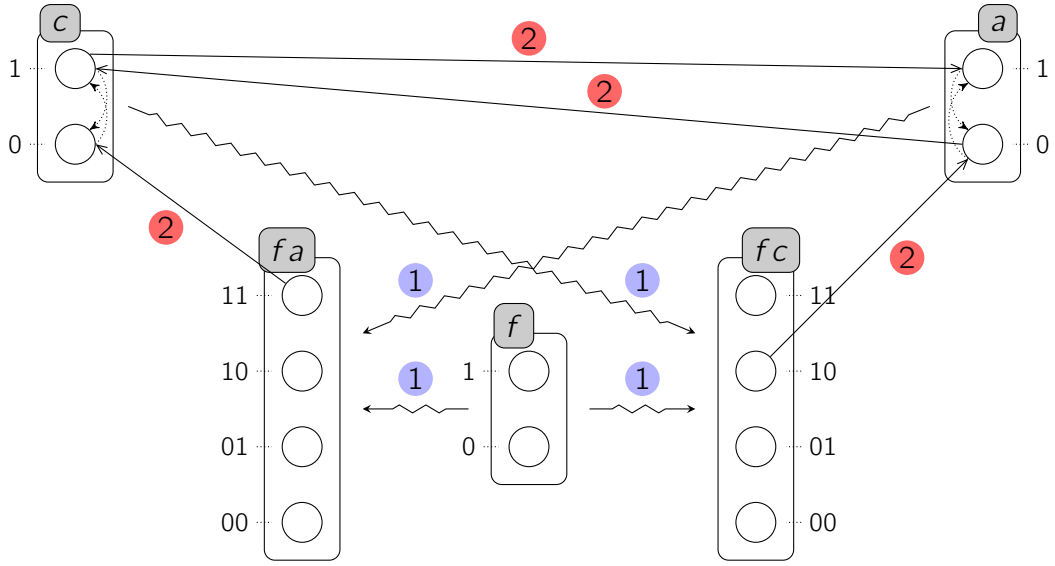


Figure 4.3 – Frappes de Processus canoniques issues de l'aplatissement du modèle de la figure 3.1 en page 45. Les étiquettes numérotées (1 et 2) placées contre les flèches représentant les actions symbolisent leur appartenance à une classe de priorités donnée.

Processus avec classes de priorités, les Frappes de Processus avec arcs neutralisants permettent une définition beaucoup plus fine des préemptions entre actions. Une des conséquences sur les modèles créés est un nombre bien moins important de relations préempteur/préempté entre les actions, rendant la traduction plus efficace.

4.2.3 Aplatissement des Frappes de Processus avec actions plurielles

Il est possible de traduire les Frappes de Processus avec actions plurielles en Frappes de Processus canoniques à l'aide d'outils précédemment développés. En effet, la définition 3.9 en page 59 offre une traduction des Frappes de Processus avec action plurielles en Frappes de Processus avec 4 classes de priorités, celles-ci pouvant être à leur tour traduites en Frappes de Processus canoniques à l'aide de la définition 4.11 en page 75. Globalement, cette traduction est donc exponentielle dans le nombre d'actions dans le modèle initial, car la traduction de chacune d'entre elles crée une sorte coopérative et plusieurs actions de priorités différentes, qui doivent par la suite être aplaties.

4.2.4 Représentation en Frappes de Processus avec actions plurielles

Les Frappes de Processus canoniques permettent notamment de représenter des coopérations entre processus. Il peut être intéressant d'observer ces coopérations du point de vue des Frappes de Processus avec actions plurielles, dont le formalisme est particulièrement adapté à la représentation des coopérations. Pour toutes Frappes de Processus canoniques $\mathcal{PH}$, nous proposons à la définition 4.12 une représentation alternative $\mathsf{plur}(\mathcal{PH})$ de ce modèle en Frappes de Processus avec actions plurielles, et nous montrons au théorème 4.2 que $\mathsf{plur}(\mathcal{PH})$ possède bien la même dynamique que $\mathcal{PH}$, aux mises à jour de sortes coopératives près. Cette traduction se base sur l'interprétation de chaque action de priorité 2 du modèle $\mathcal{PH}$:

- Si le frappeur de l'action est un processus de composant (Γ), la traduction est alors triviale ;
- Si à l'inverse, le frappeur est un processus de sorte coopérative (Δ), alors la sorte coopérative est étudiée et il y a autant d'actions multiples créées qu'il y a de configurations représentées par ce frappeur.

Autrement dit, les actions plurielles permettent d'avantageusement représenter les sortes coopératives en représentant les ensembles de processus requis pour activer un processus donné d'une sorte coopérative. La figure 4.4 est la traduction en Frappes de Processus avec actions plurielles du modèle simple de Frappes de Processus canoniques de la figure 4.1.

Définition 4.12. Soient $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$ des Frappes de Processus canoniques. On pose : $\mathsf{plur}(\mathcal{PH}) = (\overline{\Sigma}\,;\overline{\mathcal{L}}\,;\overline{\mathcal{H}})$ les Frappes de Processus avec actions plurielles telles que :

- $\overline{\Sigma} = \Gamma$;
- $\overline{\mathcal{L}} = \bigotimes_{a\in\Sigma'} \mathcal{L}_a$;
- $\overline{\mathcal{H}} = \{\big(ps \cup \{\mathsf{cible}(h)\}\big) \rightarrowtail \mathsf{bond}(h) \mid h \in \mathcal{H}^{(2)} \wedge ps \in \mathsf{FrappeursVirtuels}(h)\}$ avec, si on note $\mathsf{frappeur}(h) = a_i$:

$$\mathsf{FrappeursVirtuels}(h) = \begin{cases} \{\{a_i\}\} & \text{si } a \in \Gamma \\ \mathsf{procState}(a_i) & \text{si } a \in \Delta \end{cases}$$

De plus, pour tout état $s \in \mathcal{L}$, $\lceil s \rceil = \overline{s}$ est l'état correspondant dans $\overline{\mathcal{L}}$: $\forall a \in \overline{\Sigma}, \overline{s}[a] = s[a]$. À l'inverse, étant donné un état $\overline{s} \in \overline{\mathcal{L}}$, on note $\lfloor \overline{s} \rfloor = s$ l'état correspondant dans $\mathcal{L}$: $\forall a \in \Gamma, s[a] = \overline{s}[a]$ et $\forall f \in \Delta, s[f] = f_\sigma$ avec $f_\sigma \in \mathcal{L}_f$ et $\forall b \in \mathsf{comp}^1(f), \sigma[b] = s[b]$.

Théorème 4.2 ($\mathcal{PH} \approx \mathsf{plur}(\mathcal{PH})$)**.** *Soient $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$ des Frappes de Processus canoniques, et $\mathsf{plur}(\mathcal{PH}) = (\overline{\Sigma}\,;\overline{\mathcal{L}}\,;\overline{\mathcal{H}})$.*

1. $\forall s, s' \in \mathcal{L},\ s \to_{\mathcal{PH}} s' \Longrightarrow \lceil s \rceil = \lceil s' \rceil \vee \lceil s \rceil \to_{\mathsf{plur}(\mathcal{PH})} \lceil s' \rceil$,
2. $\forall \overline{s}, \overline{s}' \in \overline{\mathcal{L}},\ \overline{s} \to_{\mathsf{plur}(\mathcal{PH})} \overline{s}' \Longrightarrow \lfloor \overline{s} \rfloor \rightsquigarrow_{\mathcal{PH}} \lfloor \overline{s}' \rfloor$, *où $\rightsquigarrow_{\mathcal{PH}}$ est une séquence finie de transitions $\to_{\mathcal{PH}}$.*

Démonstration. (1) Soient $s, s' \in \mathcal{L}$ tels que $s \to_{\mathcal{PH}} s'$. Il existe donc une action $h \in \mathcal{H}$ telle que $s' = s \cdot h$.

- Si $h \in \mathcal{H}^{(1)}$, alors $\lceil s \rceil = \lceil s' \rceil$.
- Sinon, $h \in \mathcal{H}^{(2)}$; cela signifie qu'aucune action de priorité 1 n'est jouable, et qu'il existe $ps \in \mathsf{FrappeursVirtuels}(h)$ tel que $ps \cup \{\mathsf{cible}(h)\} \subseteq s$. D'après la définition 4.12, il existe une action $g \in \overline{\mathcal{H}}$ telle que $g = ps \cup \{\mathsf{cible}(h)\} \rightarrowtail \mathsf{bond}(h)$. Ainsi, g est jouable dans $\lceil s \rceil$ et : $\lceil s \rceil \cdot g = \lceil s \rceil \Cap \mathsf{frappeur}(h) = \lceil s' \rceil$. Donc $\lceil s \rceil \to_{\mathsf{plur}(\mathcal{PH})} \lceil s' \rceil$.

(2) Soient $\overline{s}, \overline{s}' \in \overline{\mathcal{L}}$ tels que $\overline{s} \to_{\mathsf{plur}(\mathcal{PH})} \overline{s}'$. Il existe donc une action $g \in \overline{\mathcal{H}}$ telle que $\overline{s}' = \overline{s} \cdot g$. De plus, cela signifie que $\mathsf{frappeur}(g) \subseteq \overline{s}$, donc $\mathsf{frappeur}(g) \subseteq \lfloor \overline{s} \rfloor$. Par construction de $\mathsf{plur}(\mathcal{PH})$, il existe une action $h \in \mathcal{H}^{(2)}$ telle que $\mathsf{frappeur}(g) = ps \cup \{\mathsf{cible}(h)\}$ et $\mathsf{bond}(h) = \mathsf{bond}(g)$, avec $ps \in \mathsf{FrappeursVirtuels}(h)$. Par définition de $\lfloor \overline{s} \rfloor$, toutes les sortes coopératives sont mises à jour dans $\lfloor \overline{s} \rfloor$, ce qui fait que h est jouable dans $\lfloor \overline{s} \rfloor$. Par ailleurs, d'après le lemme 4.1 en page 72, il existe un scénario $\delta \in \mathbf{Sce}(\lfloor \overline{s} \rfloor \cdot h)$ tel que

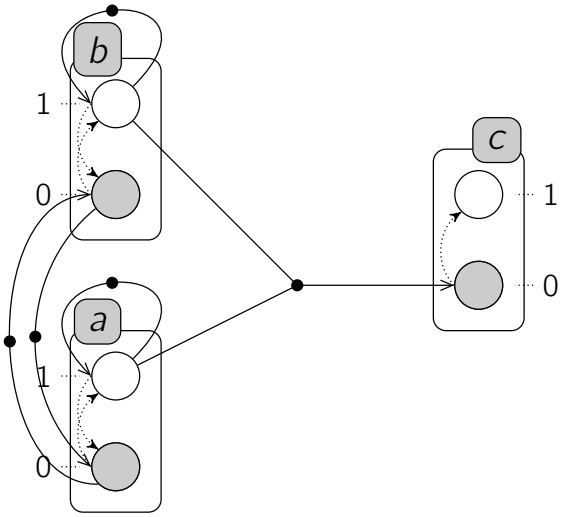


Figure 4.4 – Frappes de Processus avec actions plurielles $\mathsf{plur}(\mathcal{PH})$ issues de la traduction des Frappes de Processus canoniques $\mathcal{PH}$ de la figure 4.1 en page 71 d'après la définition 4.12. Chaque action plurielle est représentée par un point relié aux frappeurs invariants par des arcs (sans flèche) et aux cibles et bonds par un couple de flèches (respectivement en trait plein puis en trait pointillé). Ici, l'action plurielle $\{a_1, b_1, c_0\} \rightarrowtail \{c_1\}$ remplace la sorte coopérative ab et l'action $ab_{11} \rightarrow c_0 \nearrow c_1$ de la figure 4.1, qui permettaient de modéliser la coopération. Les processus grisés présentent un exemple d'état de départ : $\langle a_0, b_0, c_0 \rangle$.

$\lfloor \overline{s} \rfloor \cdot h \cdot \delta = \mathsf{update}(\lfloor \overline{s} \rfloor \cdot h)$. Enfin, par définition de update (définition 4.9 en page 72) on a : $\lfloor \overline{s} \rfloor \cdot h \cdot \delta = \lfloor \overline{s}' \rfloor$. Ainsi : $\lfloor \overline{s} \rfloor \rightsquigarrow_{\mathcal{PH}} \lfloor \overline{s}' \rfloor$. □

4.3 Analyse statique

L'objectif de cette section est de définir le problème de l'*atteignabilité* dans des Frappes de Processus, aussi appelée *accessibilité*, et de proposer une sous-approximation permettant de la résoudre efficacement dans les Frappes de Processus canoniques.

Le problème de l'atteignabilité dans les Frappes de Processus consiste à rechercher l'existence d'un scénario qui permette d'activer un ou plusieurs processus donné(s). Il peut se résumer à la question suivante : « Étant donné un état initial, existe-t-il un scénario partant de cet état et qui permette d'activer un processus donné ? » ou, de façon plus générale pour un ensemble de processus : « Étant donné un état initial ς et une séquence de processus ω donnés, existe-t-il un scénario δ jouable dans ς et permettant d'activer successivement chacun des processus de ω dans l'ordre ? » Ce problème d'atteignabilité peut parfois être résolu à l'aide des outils de *model checking* classiques. Cependant, de telles méthodes reposent généralement sur l'analyse de la dynamique complète du modèle. Pour de grands modèles, ces méthodes se heurtent donc à l'explosion combinatoire inhérente au calcul du graphe des états.

La méthode proposée ici repose en revanche sur une sous-approximation du modèle analysé. Cela permet d'éviter la complexité exponentielle de l'analyse exhaustive de la dynamique, car notre méthode possède une complexité polynomiale dans la taille du modèle

sous la condition que chaque sorte du modèle possède un nombre restreint de processus (une sorte de quatre processus ou moins satisfaisant ce critère). Cette méthode repose sur une succession d'analyses locales d'atteignabilités qui se concentrent sur les sortes plutôt que sur le modèle complet. Chaque atteignabilité est résolue sur une sorte en observant les actions qui permettent de faire bondir le processus actif vers le processus recherché. Comme ces actions sont éventuellement conditionnées par la présence d'un autre processus d'une autre sorte, cela crée d'autres atteignabilités locales qui doivent être résolues. Le problème est donc résolu récursivement, la condition d'arrêt étant soit une atteignabilité locale impossible (ce qui peut empêcher de conclure), soit un processus requis qui fait partie de l'état initial (ce qui consiste en une atteignabilité locale *triviale*). Cette méthode est inspirée du travail de Paulevé et al. (2012), qui portait sur l'atteignabilité dans les Frappes de Processus standards. Les définitions 4.13 à 4.21 sont issues de la thèse de Loïc Paulevé (2011). La contribution spécifique à cette thèse débute à la définition 4.22 et comprend cette définition et tous les résultats suivants.

Un certain nombre d'outils préliminaires nécessaires à la résolution du problème de l'atteignabilité dans des Frappes de Processus canoniques sont définis à la section 4.3.1. Le mécanisme de résolution des atteignabilités locales mentionné au paragraphe précédent est alors à son tour formalisé à la section 4.3.2 sous la forme d'un *graphe de causalité locale* : si ce graphe possède certaines propriétés, le théorème 4.3 en page 85 permet alors de conclure quand à un problème d'atteignabilité donné. Nous discutons aussi dans la suite du problèmes de l'atteignabilité simultanée d'un ensemble de processus (section 4.3.3). Nous proposons enfin une méthode permettant de raffiner cette approximation dans le cas d'atteignabilités successives (section 4.3.4).

On considère dans toute la suite de cette section un modèle de Frappes de Processus canoniques $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$ telles que définies à la définition 4.8.

4.3.1 Définitions préliminaires

L'atteignabilité d'un processus a_j d'une sorte a donnée depuis un autre processus a_i de la même sorte est le fait, depuis un état où a_i est actif, de pouvoir jouer un scénario menant dans un état où a_j est actif. La question de l'existence d'un tel scénario possède naturellement un intérêt particulier dans la résolution d'une atteignabilité locale ; c'est pourquoi on la représente sous la forme d'un *objectif*, noté $a_i \Rsh^* a_j$ (définition 4.13). De plus, on appelle *séquence d'objectifs* toute séquence dans laquelle la cible de chaque objectif est égale au bond de l'objectif précédent de la même sorte dans la séquence, s'il existe (définition 4.14).

Définition 4.13 (Objectif (**Obj**))**.** Si $a \in \Gamma$, l'atteignabilité d'un processus a_j depuis un processus a_i est appelé un *objectif*, noté $a_i \Rsh^* a_j$. L'ensemble de tous les objectifs est noté $\mathbf{Obj} \stackrel{def}{=} \{a_i \Rsh^* a_j \mid a \in \Gamma \wedge (a_i, a_j) \in \mathcal{L}_a \times \mathcal{L}_a\}$. Pour tout objectif $P = a_i \Rsh^* a_j \in \mathbf{Obj}$, on note $\mathsf{sorte}(P) \stackrel{def}{=} a$ la sorte de l'objectif P, $\mathsf{cible}(P) \stackrel{def}{=} a_i$ sa cible et $\mathsf{bond}(P) \stackrel{def}{=} a_j$ son bond. Enfin, P est dit *trivial* si $a_i = a_j$.

Définition 4.14 (Séquence d'objectif (**Obj**))**.** 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 $\mathsf{cible}(\omega_n) = a_i \Rightarrow \mathsf{dernier}_a(\omega_{1\ldots n-1}) \in \{\varnothing, a_i\}$. L'ensemble des séquences d'objectifs est référé par **OS**. Les définitions de $\mathsf{premier}_a$, $\mathsf{dernier}_a$, $\mathsf{support}$ et fin (définition 2.9 en page 27) sont étendues aux séquences d'objectifs en omettant de spécifier le cas des frappeurs.

La définition 4.15 introduit la notion de *contexte* qui étend celle d'état afin de pouvoir représenter un ensemble d'états initiaux possibles : plutôt que d'attribuer un seul processus actif à chaque sorte, comme pour un état, un contexte permet d'en attribuer plusieurs. La notion de recouvrement, précédemment définie sur un états (définition 2.11 en page 28) est étendue au cas d'un contexte dans la définition 4.16. Il permettra à la définition 4.22 en page 84 de saturer le contexte initial d'analyse avec des processus supplémentaires.

Définition 4.15 (Contexte (**Ctx**))**.** Un *contexte* ς associe à chaque sorte dans Σ un sous-ensemble non vide de ses processus : $\forall a \in \Sigma, \varsigma[a] \subseteq \mathcal{L}_a \wedge \varsigma[a] \neq \emptyset$. On note **Ctx** l'ensemble de tous les contextes.

Définition 4.16 (Recouvrement ($\Cap$: **Ctx** $\times$ $\wp$(**Proc**) $\rightarrow$ **Ctx**))**.** Pour tout contexte $\varsigma \in$ **Ctx** et tout ensemble de processus $ps \subset$ **Proc**, le recouvrement de ς par ps est noté $\varsigma \Cap ps$ et est défini par :

$$\forall a \in \Sigma, (\varsigma \Cap ps)[a] \stackrel{def}{=} \begin{cases} \{p \in ps \mid \mathsf{sorte}(p) = a\} & \text{si } \exists p \in ps, \mathsf{sorte}(p) = a, \\ \varsigma[a] & \text{sinon.} \end{cases}$$

Pour tout contexte $\varsigma \in$ **Ctx** et tout processus $a_i \in$ **Proc**, on note : $a_i \in \varsigma \stackrel{def}{\Leftrightarrow} a_i \in \varsigma[a]$, et pour tout état $ps \in \mathcal{L}$ ou ensemble de processus $ps \subset$ **Proc**, on note : $ps \subseteq \varsigma \stackrel{def}{\Leftrightarrow} \forall a_i \in ps, a_i \in \varsigma$. De plus, une séquence d'actions δ est *jouable* dans un contexte ς si et seulement si $\exists s \subseteq \varsigma, \delta \in$ **Sce**(s) ; on note alors : $\delta \in$ **Sce**(ς), et le jeu de δ dans ς est : $\varsigma \cdot \delta = \varsigma \Cap \mathsf{fin}(\delta)$.

Finalement, une séquence de bonds sur une sorte a (définition 4.17) est une séquence d'actions frappant a dans laquelle le bond de chaque action est égal à la cible de l'action suivante, en ignorant donc le frappeur de chaque action. Les séquences de bonds sont utilisées pour trouver les solutions locales d'un objectif donné. Une séquence de bonds sur a peut de plus être abstraite par l'ensemble de tous les frappeurs de ses actions qui ne sont pas dans a (définition 4.18). Cette abstraction permet de déplacer un objectif qui concerne une sorte a vers d'autres objectifs sur d'autres sortes. On note dans la suite : **Sol** $=$ $\wp$(**Proc**).

Définition 4.17 (Séquence de bonds (**BS**))**.** Une *séquence de bonds* ζ est une séquence d'actions telle que $\forall n \in \mathbb{I}^{\zeta}, n < |\zeta|, \mathsf{bond}(\zeta_n) = \mathsf{cible}(\zeta_{n+1})$. L'ensemble de toutes les séquences de bonds est appelé **BS**, et on note **BS**(P) l'ensemble de toutes les séquences de bonds *résolvant* un objectif P, appelé **BS**(P), qui est défini par :

$$\mathbf{BS}(a_i \Rsh^* a_j) \stackrel{def}{=} \{\zeta \in \mathbf{BS} \mid \mathsf{cible}(\zeta_1) = a_i \wedge \mathsf{bond}(\zeta_{|\zeta|}) = a_j\} \ .$$

On remarque que pour tout objectif $a_i \Rsh^* a_j \in$ **Obj**, **BS**$(a_i \Rsh^* a_j) = \emptyset$ s'il n'existe aucun moyen d'atteindre a_j depuis a_i. À l'inverse, la séquence vide appartient toujours à l'ensemble des séquences de bonds résolvant un objectif trivial : $\forall a_i \in$ **Proc**, $\varepsilon \in$ **BS**$(a_i \Rsh^* a_i)$.

Définition 4.18 (Séquence de bonds abstraite (**BS**$^{\wedge}$: **Obj** $\rightarrow$ $\wp$(**Sol**)))**.**

$$\mathbf{BS}^{\wedge}(P) \stackrel{def}{=} \{\zeta^{\wedge} \in \mathbf{Sol} \mid \zeta \in \mathbf{BS}(P), \nexists \zeta' \in \mathbf{BS}(P), \zeta'^{\wedge} \subsetneq \zeta^{\wedge}\} \ ,$$

où $\zeta^{\wedge} \stackrel{def}{=} \{\mathsf{frappeur}(\zeta_n) \mid n \in \mathbb{I}^{\zeta} \wedge \mathsf{sorte}(\mathsf{frappeur}(\zeta_n)) \neq \mathsf{sorte}(P)\}$.

4.3.2 Sous-approximation

On note $\gamma_\varsigma(\omega)$ l'ensemble des scénarios concrétisant une séquence d'objectifs ω dans le contexte ς (définition 4.19) et $\ell_\varsigma(\omega)$ est défini comme étant égal à $\gamma_\varsigma(\omega)$ si et seulement si, pour chaque état $s \subseteq \varsigma$, $\gamma_\varsigma(\omega) \cap \mathbf{Sce}(s) \neq \emptyset$ (définition 4.20).

Définition 4.19 ($\gamma_\varsigma : \mathbf{OS} \to \wp(\mathbf{Sce})$)**.** Pour toute séquence d'objectifs $\omega \in \mathbf{OS}$, $\gamma_\varsigma(\omega)$ est l'ensemble des scénarios minimaux concrétisant ω dans le contexte ς. Il est défini comme le plus grand ensemble satisfaisant les conditions suivantes :

(i) $\forall \delta \in \gamma_\varsigma(\omega), \exists s \subseteq \varsigma, \delta \in \mathbf{Sce}(s)$,

(ii) $\forall \delta \in \gamma_\varsigma(\omega), \exists \phi : \mathbb{I}^\omega \to \mathbb{I}^\delta, (\forall n, m \in \mathbb{I}^\omega, n < m \Leftrightarrow \phi(n) \leq \phi(m)), \forall n \in \mathbb{I}^\omega, \mathsf{bond}(\omega_n) \in \varsigma \cdot \delta_{1\ldots\phi(n)}$,

(iii) $\forall \delta, \delta' \in \gamma_\varsigma(\omega), |\delta| \leq |\delta'| \Rightarrow \delta \neq \delta'_{1\ldots|\delta|}$.

Définition 4.20 ($\ell_\varsigma : \mathbf{OS} \to \wp(\mathbf{Sce})$)**.**

$$\ell_\varsigma(\omega) \stackrel{def}{=} \begin{cases} \gamma_\varsigma(\omega) & \text{si } \forall s \in \mathcal{L}, s \subseteq \varsigma, \exists \delta \in \gamma_\varsigma(\omega), \delta \in \mathbf{Sce}(s) \\ \emptyset & \text{sinon.} \end{cases}$$

Lemme 4.5. $\varsigma \subseteq \varsigma' \wedge \ell_{\varsigma'}(\omega) \neq \emptyset \Rightarrow \ell_\varsigma(\omega) \neq \emptyset$.

Pour tout objectif $P \in \mathbf{Obj}$ et tout contexte $\varsigma \in \mathbf{Ctx}$, la définition 4.21 permet d'obtenir $\mathsf{maxCont}_\varsigma(\mathsf{sorte}(P), P)$ qui est l'ensemble des processus de $\mathsf{sorte}(P)$ requis pour résoudre P dans ς. Cette définition sera utile pour correctement résoudre les atteignabilités locales qui nécessitent indirectement un processus de leur propre sorte, c'est-à-dire autrement que par une auto-action.

Définition 4.21 ($\mathsf{maxCont}_\varsigma : \Sigma \times \mathbf{Obj} \to \wp(\mathbf{Proc})$)**.**

$$\mathsf{maxCont}_\varsigma(a, P) \stackrel{def}{=} \{p \in \mathbf{Proc} \mid \exists ps \in \mathbf{BS}^\wedge(P), \exists b_i \in ps, b = a \wedge p = b_i \vee b \neq a \wedge p \in \mathsf{maxCont}_\varsigma(a, b_j \Rightarrow^* b_i) \wedge b_j \in \varsigma[b]\} \ .$$

Pour une séquence d'objectifs ω et un contexte ς donnés, le *graphe de causalité locale* $\lceil \mathcal{B}^\omega_\varsigma \rceil$ (définition 4.22) représente une sous-approximation de l'atteignabilité de cette séquence d'objectifs dans ς. Pour cela, il relie les objectifs à des solutions à l'aide des séquences de bonds abstraites de la définition 4.18, ce qui produit de nouveaux objectifs résolus récursivement. Il s'agit donc d'un graphe dont les nœuds sont des éléments de $\mathbf{Proc} \cup \mathbf{Obj} \cup \mathbf{Sol}$, c'est-à-dire des processus, des objectifs et des *solutions* (c'est-à-dire des ensembles de processus) :

- Un nœud dans **Obj** représente un objectif requis pour la résolution de ω, soit faisant directement partie de la séquence d'objectifs ω, soit indirectement nécessaire à sa résolution ;
- Un nœud dans **Sol** représente un ensemble de processus nécessaires pour résoudre un objectif, c'est-à-dire un élément parmi ses séquences de bonds abstraites ;
- Un nœud dans **Proc** représente un processus requis pour la résolution, c'est-à-dire mentionné dans un nœud solution.

Un objectif $P \in \mathbf{Obj}$ est soluble si tous les processus contenus dans au moins une de ses abstractions de séquences de bonds $\mathbf{BS}^\wedge(P) \in \mathbf{Sol}$ (cf. définition 4.18) peuvent être

activés (équation (4.4)). Une telle solution représente donc un ensemble de processus qui doivent être activés pour la résolution de P (équation (4.5)). Si $a \in \Gamma$, l'atteignabilité d'un de ses processus a_i est approximée par la possibilité de résoudre tous les objectifs de la forme $a_j \Rsh^* a_i \in \mathbf{Obj}$ pour tout a_j dans le contexte initial ς (équation (4.6)) ; si $a \in \Delta$, l'atteignabilité de a_i est possible si tous les processus du sous-état $\mathsf{procState}(a_i)$ (cf. définition 4.6) qu'il représente sont atteignables (équation (4.7)). La résolution d'un objectif P peut nécessiter un processus p de $\mathsf{sorte}(P)$, autrement dit : $\mathsf{maxCont}(\mathsf{sorte}(P), P) \neq \emptyset$ (cf. définition 4.21) ; dans ce cas, P est *re-centré* en p (équation (4.8)) afin de s'assurer que la résolution intermédiaire de $\mathsf{cible}(P) \Rsh^* p$ n'interfère pas. Enfin, les équations (4.1), (4.2) et (4.3) assurent que l'ensemble des nœuds est complet.

Étant donné que le processus actif de chaque sorte peut évoluer au cours de la résolution, le graphe de causalité locale $\lceil \mathcal{B}^\omega_\varsigma \rceil$ est obtenu par *saturation* avec tous les processus qu'il contient, c'est-à-dire en recouvrant le contexte initial ς par $\mathsf{procs}(V, E)$, défini par :

$$\mathsf{procs}(V, E) = (V \cap \mathbf{Proc}) \cup \{\mathsf{cible}(P), \mathsf{bond}(P) \mid P \in V \cap \mathbf{Obj}\} \ .$$

Ce recouvrement est effectué autant de fois que nécessaire ; le graphe de causalité locale est donc re-calculé avec cette saturation jusqu'à ce qu'il n'évolue plus — autrement dit, jusqu'à atteindre un point fixe.

Définition 4.22. Le graphe de causalité locale $\lceil \mathcal{B}^\omega_\varsigma \rceil \stackrel{def}{=} (\lceil V^\omega_\varsigma \rceil, \lceil E^\omega_\varsigma \rceil)$ est défini par : $\lceil \mathcal{B}^\omega_\varsigma \rceil \stackrel{def}{=} \mathbf{pppf}\{\mathcal{B}^\omega_\varsigma\}\left(\mathcal{B}^\omega_\varsigma \mapsto \mathcal{B}^\omega_{\varsigma \Cap \mathsf{procs}(\mathcal{B}^\omega_\varsigma)}\right)$, où $\mathcal{B}^\omega_\varsigma \stackrel{def}{=} (V^\omega_\varsigma, E^\omega_\varsigma)$ est le plus petit graphe respectant $V^\omega_\varsigma \subseteq \mathbf{Proc} \cup \mathbf{Obj} \cup \mathbf{Sol}$ et $E^\omega_\varsigma \subseteq V^\omega_\varsigma \times V^\omega_\varsigma$ tel que :

$$\omega \subseteq V^\omega_\varsigma \tag{4.1}$$
$$P \in V^\omega_\varsigma \cap \mathbf{Obj} \Rightarrow \mathsf{bond}(P) \in V^\omega_\varsigma \tag{4.2}$$
$$(x, y) \in E^\omega_\varsigma \Rightarrow y \in V^\omega_\varsigma \tag{4.3}$$
$$P \in V^\omega_\varsigma \cap \mathbf{Obj} \wedge ps \in \mathbf{BS}(P) \Rightarrow (P, ps) \in E^\omega_\varsigma \tag{4.4}$$
$$ps \in V^\omega_\varsigma \cap \mathbf{Sol} \wedge a_i \in ps \Rightarrow (ps, a_i) \in E^\omega_\varsigma \tag{4.5}$$
$$a \in \Gamma \wedge a_i \in V^\omega_\varsigma \cap \mathbf{Proc} \wedge a_j \in \varsigma \Rightarrow (a_i, a_j \Rsh^* a_i) \in E^\omega_\varsigma \tag{4.6}$$
$$a \in \Delta \wedge a_i \in V^\omega_\varsigma \cap \mathbf{Proc} \wedge ps \in \mathsf{procState}(a_i) \Rightarrow (a_i, ps) \in E^\omega_\varsigma \tag{4.7}$$
$$P \in V^\omega_\varsigma \cap \mathbf{Obj} \wedge q \in \mathsf{maxCont}_\varsigma(\mathsf{sorte}(P), P) \Rightarrow (P, q \Rsh^* \mathsf{bond}(P)) \in E^\omega_\varsigma \tag{4.8}$$

Exemple. La figure 4.5 en page 87 représente le graphe de causalité locale associé au modèle de Frappes de Processus canoniques de la figure 4.1 en page 71, pour la question de l'accessibilité de c_1 depuis l'état $\langle a_1, b_0, c_0, ab_{10}\rangle$. Nous discutons en page 86 des conclusions qui peuvent en être tirées.

Au sein de ce graphe de causalité locale, un arc $(p, ps) \in \mathbf{Proc} \times \mathbf{Sol}$ est dit *cohérent* (définition 4.23) si aucun des processus dans ps n'est « compromis » par un processus successeur du nœud ps, c'est-à-dire si, pour tout processus de ps, il n'existe pas de processus différent de la même sorte parmi tous les successeurs de ps. Si tous les arcs du graphe sont cohérents, alors le théorème 4.3 donne une condition suffisante pour la concrétisation de la séquence d'objectifs ω dans le contexte ς, qui est basée directement sur ce graphe $\lceil \mathcal{B}^\omega_\varsigma \rceil$.

Définition 4.23 (Arc cohérent)**.** Un arc $(x, y) \in E^\omega_\varsigma$ est dit *cohérent* si et seulement si $(x, y) \in \lceil E^\omega_\varsigma \rceil \cap (\mathbf{Proc} \times \mathbf{Sol}) \Rightarrow y$ n'a aucun successeur $a_j \in \lceil V^\omega_\varsigma \rceil \cap \mathbf{Proc}$ tel que $\exists a_i \in y, \mathsf{sorte}(a_i) = \mathsf{sorte}(a_j) \wedge a_i \neq a_j$.

Théorème 4.3 (Sous-Approximation)**.** *Étant données des Frappes de Processus canoniques* $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$*, un contexte* $\varsigma \in \mathbf{Ctx}$ *et une séquence d'objectifs* $\omega \in \mathbf{OS}$*, si le graphe* $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$ *ne contient aucun cycle, que tous ses nœuds objectifs possèdent au moins une solution et que tous ses arcs sont cohérents, alors* $\ell_{\varsigma}(\omega) \neq \emptyset$*.*

Démonstration. On note dans la suite : $\lceil E^{\omega}_{\varsigma} \rceil^{X}_{Y} = \lceil E^{\omega}_{\varsigma} \rceil \cap (X \times Y)$, avec X, Y parmi **Proc**, **Obj** et **Sol**, et : $max\varsigma = \varsigma \Cap \mathsf{procs}(\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil)$ le contexte accepté par $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$.

Soit $(a_i, ps) \in \lceil E^{\omega}_{\varsigma} \rceil^{\mathsf{Proc}}_{\mathsf{Sol}}$ un arc liant un processus requis de sorte coopérative à une solution et supposons que tous les enfants de ps sont concrétisables. On étiquette tous les processus de ps par un entier : $ps = \{p_n\}_{n \in \mathbb{I}^{ps}}$. Montrons par récurrence que pour tout $n \in [\![0\,;|ps|]\!]$, il existe un scénario δ_n tel que : $\forall i \in [\![1\,;n]\!], (s \cdot \delta_n)[\mathsf{sorte}(p_i)] = p_i$.

- Le cas $\delta_0 = \varepsilon$ est immédiat.
- Soit $n \in [\![0\,;|ps|-1]\!]$. On suppose qu'il existe δ_n tel que décrit ci-dessus. Posons $q = (s \cdot \delta_n)[\mathsf{sorte}(p_{n+1})]$. Par hypothèse, (a_i, ps) est cohérent (définition 4.23) et tous les processus de ps sont des processus de composants ; cela signifie qu'aucun des processus requis pour résoudre p_{n+1} n'est un autre processus de la même sorte qu'un processus de ps. Par conséquent, il existe un scénario $\delta' \in \ell_{s\cdot\delta_n}(q \Rsh^{*} p_{n+1})$ tel que $\forall i \in [\![1\,;n+1]\!], (s \cdot \delta_n \cdot \delta')[\mathsf{sorte}(p_i)] = p_i$. Finalement, d'après le lemme 4.1, il existe un scénario $\delta'' \in \mathbf{Sce}^{1}(s \cdot \delta_n \cdot \delta')$ tel que $\mathsf{update}(s \cdot \delta_n \cdot \delta') = s \cdot \delta_{n+1}$ avec $\delta_{n+1} = \delta_n \cdot \delta' \cdot \delta''$, et d'après le lemme 4.3 : $\forall i \in [\![1\,;n+1]\!], (s \cdot \delta_{n+1})[\mathsf{sorte}(p_i)] = p_i$

Ainsi, $\delta = \delta_{|ps|}$ existe, et étant données ses propriétés, on a immédiatement : $(s \cdot \delta)[a] = a_i$ et $\mathsf{update}(s \cdot \delta) = s \cdot \delta$.

Soit un état $s \in L$ tel que $s \subseteq max\varsigma$. Étant donné qu'il n'y a aucun cycle dans $\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil$, montrons par récurrence que pour tout objectif $P \in \lceil V^{\omega}_{\varsigma} \rceil \cap \mathbf{Obj}$ tel que $\mathsf{cible}(P) \in s$, $\exists \delta \in \ell_s(P)$.

- Si $(P, \emptyset) \in \lceil E^{\omega}_{\varsigma} \rceil^{\mathsf{Obj}}_{\mathsf{Sol}}$, soit on a $\mathsf{cible}(P) = \mathsf{bond}(P)$ et $\delta = \varepsilon$, soit on a $\forall \zeta \in \mathbf{BS}(P), \zeta \in \mathbf{Sce}(s) \wedge \mathsf{sorte}(\zeta) = \{\mathsf{sorte}(P)\}$ et dans ce cas $\delta = \delta_1 \cdot \zeta_1 \cdot \cdots \cdot \delta_{|\zeta|} \cdot \zeta_{|\zeta|}$ est une séquence valide donnée par le lemme 4.2.
- Supposons que tous les objectifs qui sont les successeurs de P sont concrétisables. Si $\exists (P, Q) \in \lceil E^{\omega}_{\varsigma} \rceil^{\mathsf{Obj}}_{\mathsf{Obj}}$, alors, par hypothèse, $\ell_s(\mathsf{cible}(P) \Rsh^{*} \mathsf{cible}(Q) :: Q) \neq \emptyset$, et donc $\ell_s(P) \neq \emptyset$. Sinon, d'après la définition 4.21, la concrétisation des successeurs de P ne requiert aucun processus de la sorte $\mathsf{sorte}(P)$. De plus, il existe $\zeta \in \mathbf{BS}(P)$ tel que $(P, \zeta^{\wedge}) \in \lceil E^{\omega}_{\varsigma} \rceil^{\mathsf{Obj}}_{\mathsf{Sol}}$. Montrons par récurrence que pour tout $n \in \mathbb{I}^{\zeta}$, il existe un scénario δ_n tel que $(s \cdot \delta_n)[\mathsf{sorte}(P)] = \mathsf{bond}(\zeta_n)$.

  Supposons que δ_n existe et posons $\zeta_n = b_i \rightarrow a_j \Rsh a_k$. Par hypothèse, il existe $\delta' \in \ell_{s\cdot\delta_n}(\star \Rsh^{*} b_i)$ avec $\mathsf{sorte}(P) \notin \mathsf{sorte}(\delta')$ (définition 4.21). D'après le lemme 4.1, il existe $\delta'' \in \mathbf{Sce}^{1}(s \cdot \delta')$ tel que $\mathsf{update}(s \cdot \delta') = s \cdot \delta' \cdot \delta''$. De plus, $(s \cdot \delta' \cdot \delta'')[b] = b_j$ (D'après le lemme 4.2 si $b \in \Gamma$ ou le lemme 4.3 si $b \in \Delta$). Ainsi, $\delta_{n+1} = \delta_n \cdot \delta' \cdot \delta'' \cdot \zeta_n$.

  On a donc :$\delta_{|\zeta|} \in \ell_s(P)$.

Finalement, étant donné $\ell_{max\varsigma}(\omega) \neq \emptyset$, et d'après le lemme 4.5, on a : $\ell_{\varsigma}(\omega) \neq \emptyset$. □

Remarque. Le théorème 4.3 peut s'appliquer à tout graphe de causalité locale $\widehat{\lceil \mathcal{B}^{\omega}_{\varsigma} \rceil}$ construit à partir d'un graphe $\widehat{\mathcal{B}^{\omega}_{\varsigma}} = (\widehat{V^{\omega}_{\varsigma}}, \widehat{E^{\omega}_{\varsigma}})$ où $\widehat{V^{\omega}_{\varsigma}} \cap \mathbf{Sol} \subset V^{\omega}_{\varsigma} \cap \mathbf{Sol}$. En effet, cela

revient à réduire l'ensemble initial des solutions, ce qui réduit aussi l'ensemble de nœuds processus et objectifs utilisés. La solution est alors davantage contrainte, mais le résultat est toujours valable. Cela revient en fait à s'interdire certaines solutions, c'est-à-dire à calculer l'atteignabilité sur un graphe privé de certaines actions. Ainsi, si la sous-approximation est non-conclusive, il est possible de la tester sur tous les graphes comportant un sous-ensemble des nœuds solutions, ce qui permet notamment de supprimer certains cycles et parfois d'obtenir un graphe de causalité locale sur lequel il est possible de conclure. Cette recherche exhaustive est cependant exponentielle dans le nombre de nœuds solutions, mais il est possible de l'orienter de façon à trouver rapidement un sous-ensemble permettant de conclure, par exemple en retirant en priorité les solutions qui forment un cycle.

Remarque. Paulevé et al. (2012) ont proposé une méthode de sur-approximation qui se base sur un graphe de causalité locale construit de façon similaire, et permet de réfuter une atteignabilité au sein d'un modèle de Frappes de Processus standards. Il est intéressant de noter que cette sur-approximation est toujours valable sur les Frappes de Processus canoniques à condition de l'appliquer sur la version fusionnée du modèle considéré (cf. définition 4.11 en page 75). Cela permet d'obtenir un résultat supplémentaire en concluant dans certains cas quant à l'impossibilité d'atteindre un processus donné.

Exemple. En ce qui concerne le modèle de Frappes de Processus canoniques de la figure 4.1 en page 71, la sous-approximation développée au théorème 4.3 ne conclut pas quant à l'accessibilité de c_1 depuis l'état $\langle a_1, b_0, c_0, ab_{10}\rangle$. En effet, comme le montre la figure 4.5, l'arc représenté en double trait liant le nœud processus ab_{11} à son unique solution n'est pas cohérent selon la définition 4.23, ce qui empêche l'application du théorème.

De même, la sur-approximation de Paulevé et al. (2012) (appliquée à $\mathcal{PH}'$) renvoie aussi un résultat non-conclusif, du fait que les deux approches ne peuvent pas être conclusives en même temps pour des raisons de cohérence mathématique. La méthode d'analyse statique ne répond donc globalement pas sur cet exemple, et de façon plus générale sur tous les exemples dont l'atteignabilité recherchée est rendue impossible par simple ajout de classes de priorités.

On note pour finir que le théorème 4.3 est conclusif sur l'atteignabilité de c_1 depuis $\langle a_1, b_0, c_0, ab_{10}\rangle$ dans les Frappes de Processus canoniques $\mathcal{PH}''$, où :

$$\mathcal{PH}'' = (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}'^{(2)}))$$

$$\text{avec :}\quad \mathcal{H}'^{(2)} = (\mathcal{H}^{(2)} \setminus \{a_0 \to b_0 \not\Uparrow b_1, b_0 \to a_0 \not\Uparrow a_1\}) \cup \{a_0 \to a_0 \not\Uparrow a_1, b_0 \to b_0 \not\Uparrow b_1\}$$

En effet, dans ce cas les processus a_0 et b_0 du graphe de la figure 4.5 sont permutés, ce qui rend tous les arcs cohérents.

Comme nous l'avons vu, l'analyse statique développée dans cette section est une approximation, et peut retourner un résultat non-conclusif ; Le modèle de la figure 4.1 traité ci-dessus en est un exemple. Une partie de ces cas non-conclusifs apparaissent notamment pour un motif particulier, mis en valeur par la notion de cohérence de la définition 4.23. Cela est dû notamment au fait que la méthode de sur-approximation n'a pas été raffinée dans le présent travail, ce qui mène à des cas non-conclusifs lorsque l'ajout de priorités empêche certains comportements.

D'autres situations peuvent aussi empêcher de conclure : c'est notamment le cas des atteignabilités nécessitant des « allers-retours », c'est-à-dire l'activation d'un processus p plusieurs fois pendant la résolution. Si d'autres requis sont intercalés entre les différentes occurrences de p, le graphe de causalité locale va alors présenter un cycle, ce qui empêche

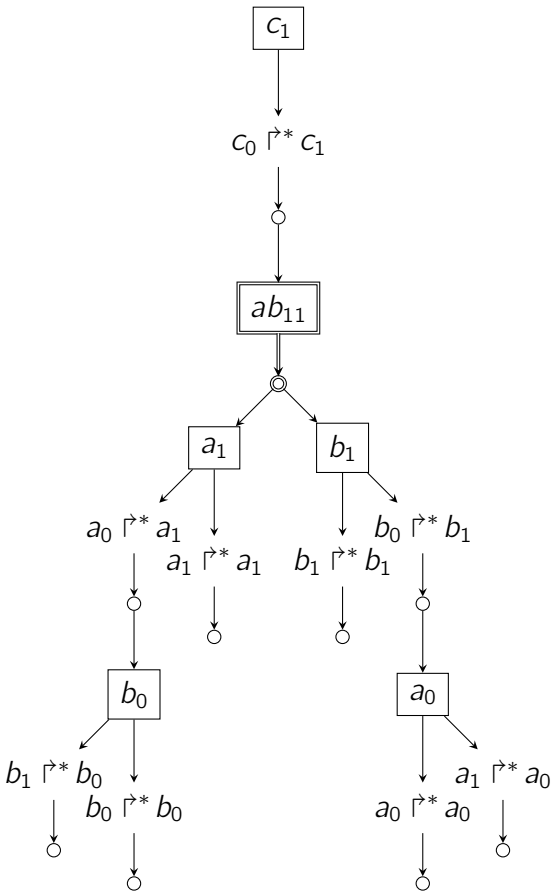


Figure 4.5 – Le graphe de causalité locale des Frappes de Processus de la figure 4.1 pour l'objectif $\omega = c_0 \Rsh^* c_1$ et le contexte initial $\varsigma = \langle a_1, b_0, c_0, ab_{10} \rangle$. Les nœuds rectangulaires représentent les éléments de **Proc**, les nœuds sans bordure sont les éléments de **Obj** et les cercles sont les éléments de **Sol**. Le processus ab_{11}, ainsi que son unique solution et l'arc qui les relie, sont mis en valeur avec des traits doubles car il s'agit du principal ajout de la méthode présentée à la section 4.3.2. Il est à noter que l'arc dessiné avec un trait double n'est pas cohérent au sens de la définition 4.23. En effet, sa cible est la solution $\{a_1, b_1\}$, or l'un de ses successeurs indirects est a_0, qui est un autre processus de la même sorte que a_1 (et le même raisonnement fonctionne pour b_0).

l'utilisation du théorème 4.3. L'une des alternatives est alors de détecter et d'expliciter cette séquentialité, ce qui permet par exemple d'utiliser le résultat qui sera présenté à la section 4.3.4.

4.3.3 Atteignabilité d'un sous-état

La propriété d'atteignabilité développée à la section 4.3.2 sur les Frappes de Processus canoniques ne traite l'atteignabilité d'un ensemble de processus que de façon séquentielle. Cependant, il est possible de vérifier l'atteignabilité d'un sous-état (autrement dit, l'atteignabilité simultanée d'un ensemble de processus) à l'aide d'une sorte coopérative.

En effet, soient $\mathcal{PH} = (\Sigma, \mathcal{L}, (\mathcal{H}^{(1)}, \mathcal{H}^{(2)}))$ des Frappes de Processus canoniques et supposons que l'ont cherche à étudier l'atteignabilité d'un sous-état $ps \in \mathcal{L}_S^\Diamond$, avec $S \subset \Sigma$. On pose alors : $\mathcal{PH}' = (\Sigma', \mathcal{L}', (\mathcal{H}'^{(1)}, \mathcal{H}'^{(2)}))$ les Frappes de Processus canoniques telles que :

- $\Sigma' = \Sigma \cup \{\tau, \rho\}$,
- $\mathcal{L}' = \mathcal{L} \times \mathcal{L}_\tau \times \mathcal{L}_\rho$, où $\mathcal{L}_\tau = \mathcal{L}_S^\Diamond$ et $\mathcal{L}_\rho = \{\rho_0, \rho_1\}$,
- $\mathcal{H}'^{(1)} = \mathcal{H}^{(1)} \cup \{a_i \to \tau_\sigma \Rsh \tau_{\sigma'} \mid a \in S, \sigma, \sigma' \in \mathcal{L}_\tau, \sigma[a] \neq a_i \wedge \sigma' = \sigma \Cap a_i\}$,
- $\mathcal{H}'^{(2)} = \mathcal{H}^{(2)} \cup \{\tau_{ps} \to \rho_0 \Rsh \rho_1\}$.

Cette transformation consiste donc à ajouter au modèle une sorte coopérative τ sur toutes les sortes de S, et un composant ρ qui ne puisse changer de processus que lorsque le sous-état ps est présent (ce qui est déterminé par τ). Ainsi, l'atteignabilité du sous-état ps depuis un contexte initial ς dans $\mathcal{PH}$ est équivalente à celle du processus ρ_1 depuis le contexte $\varsigma \cup \{\rho_0\}$ dans $\mathcal{PH}'$ (l'état initial de τ n'a pas d'importance et peut être arbitrairement choisi), qui peut être traitée grâce au théorème 4.3.

Nous pouvons donc répondre à des questions d'atteignabilité simultanée de plusieurs processus directement à l'aide des Frappes de Processus canoniques (définition 4.8) et de l'analyse statique développée pour ce formalisme (théorème 4.3). Cette méthode peut naturellement être adaptée pour répondre quant à l'accessibilité d'un ensemble de sous-états. Il est à noter cependant que le nombre de processus de la sorte coopérative τ croît exponentiellement avec le nombre de sortes dans S, ce qui peut fortement impacter la vitesse de résolution de l'analyse statique. Pour pallier cela, il est possible de « factoriser » cette sorte coopérative comme expliqué en page 33.

4.3.4 Raffinement de la sous-approximation séquentielle

Dans cette section, nous donnons une alternative à la condition suffisante du théorème 4.3 qui permet de prendre en compte la séquentialité des objectifs plutôt que de les considérer simultanément, tel que cela est fait dans la version actuelle. Comme les objectifs sont pris en compte individuellement, une telle approche ne prend en compte qu'un sous-ensemble des scénarios possibles. Cependant, en se concentrant à chaque itération sur une plus petite partie du réseau, cette sous-approximation séquentielle peut s'avérer plus souvent conclusive.

Définissions une séquence d'objectifs $\omega = a_i \Rsh^* a_j :: \omega'$ avec $a_i \neq a_j$ et un état $s \in \mathcal{L}$ tel que $s[a] = a_i$. On peut remarquer que tout scénario atteignant a_j inclut nécessairement l'une des séquences de bonds dans $\mathbf{BS}(a_i \Rsh^* a_j)$ et, en particulier, tout scénario minimal atteignant a_j termine dans un état où son présents à la fois a_j et le frappeur a_k de la

dernière action d'une des séquences de bonds dans $\mathbf{BS}(a_i \Rsh^* a_j)$. Si la sorte b d'un tel frappe est de surcroît une sorte coopérative ($b \in \Delta$), cela signifie alors aussi que l'un des sous-états dans $\mathsf{procState}(b_k)$ est inclus dans l'état final. La définition 4.24 définit $\mathsf{derprocs}(a_i \Rsh^* a_j)$ comme étant l'ensemble des ensembles de processus qui peuvent être présents juste après avoir atteint a_j.

D'après le théorème 4.3, on peut déduire que pour tout scénario $\delta \in \ell_\varsigma(P)$, il existe un ensemble de processus $ps \in \mathsf{derprocs}(P)$ tel que $ps \subset (s \cdot \delta)$. Donc, si $\ell_{\varsigma' \Cap ps}(\omega') \neq \emptyset$, avec $\varsigma' = \varsigma \Cap \mathsf{procs}(\lceil \mathcal{B}^P_\varsigma \rceil)$, il existe alors un scénario δ' concrétisant ω' depuis l'état $(s \cdot \delta)$. Ainsi, le scénario $\delta :: \delta'$ concrétise ω.

Définition 4.24 ($\mathsf{derprocs} : \mathbf{Obj} \to \wp(\wp(\mathbf{Proc}))$)**.** Pour tout objectif $a_i \Rsh^* a_j \in \mathbf{Obj}$, $\mathsf{derprocs}(a_i \Rsh^* a_j)$ est défini comme le plus grand ensemble tel que : $\forall lps \in \mathsf{derprocs}(a_i \Rsh^* a_j), lps \in \wp(\mathbf{Proc})$,

1. $a_j \in lps$,
2. $\exists \zeta \in \mathbf{BS}(a_i \Rsh^* a_j), \mathsf{sorte}(\mathsf{frappeur}(\zeta_{|\zeta|})) \neq \mathsf{sorte}(\mathsf{bond}(\zeta_{|\zeta|})) \Rightarrow \mathsf{frappeur}(\zeta_{|\zeta|}) \in lps$,
3. $\forall b_j \in lps, b \in \Delta \Rightarrow \exists ps \in \mathsf{procState}(b_j), ps \subset lps$,
4. $\nexists lps' \in \mathsf{derprocs}(a_i \Rsh^* a_j), lps' \subsetneq lps$.

Théorème 4.4 (Sous-approximation séquentielle)**.** *Pour toutes Frappes de Processus canoniques* $(\Sigma ; \mathcal{L} ; \mathcal{H}^{\langle 2 \rangle})$*, tout contexte* $\varsigma \in \mathbf{Ctx}$ *et toute séquence d'objectifs* $\omega = P :: \omega' \in \mathbf{OS}$*,* $\ell_\varsigma(P) \neq \emptyset \wedge \forall ps \in \mathsf{derprocs}(P), \ell_{\varsigma' \Cap ps}(\omega') \neq \emptyset \Rightarrow \ell_\varsigma(\omega) \neq \emptyset$*, où* $\varsigma' = \varsigma \Cap \mathsf{procs}(\lceil \mathcal{B}^P_\varsigma \rceil)$*.*

Démonstration. Si $\ell_\varsigma(P) \neq \emptyset$, alors pour tout état $s \in \mathcal{L}, s \subset \varsigma$, il existe un scénario $\delta \in \ell_\varsigma(P) \cap \mathbf{Sce}(s)$. D'après la définition 4.24, et en s'inspirant de la démonstration du théorème 4.3, il existe $ps \in \mathsf{derprocs}(P)$ tel que $(s \cdot \delta) \subset \varsigma' \Cap ps$. Ainsi, si $\ell_{\varsigma' \Cap ps}(\omega') \neq \emptyset$, alors il existe un scénario $\delta' \in \ell_{\varsigma' \Cap ps}(\omega')$ tel que $\delta' \in \mathbf{Sce}(s \cdot \delta)$. Par conséquent, $\delta :: \delta'$ est un scénario jouable dans s. Donc, pour tout $s \in \mathcal{L}, s \subset \varsigma$, il existe un scénario concrétisant ω. D'où : $\ell_\varsigma(\omega) \neq \emptyset$. □

Chapitre 5

Expressivité des Frappes de Processus et positionnement par rapport à d'autres formalismes

Nous montrons dans ce chapitre que les différentes sémantiques de Frappes de Processus sont équivalentes à un certain nombre de formalismes répandus. Nous prouvons notamment que les Frappes de Processus canoniques permettent de représenter tout réseau discret asynchrone, un formalisme très répandu dans la représentation des réseaux de régulation biologique. À l'inverse, nous proposons une méthode pour inférer les modèles de Thomas sous-jacents à des Frappes de Processus canoniques : il est possible d'inférer tous les types d'influence et les paramètres discrets, et d'énumérer tous les modèles compatibles avec ces résultats. Par ailleurs, les Frappes de Processus avec actions plurielles s'avèrent être équivalentes aux réseaux d'automates synchronisés. Enfin, nous proposons des traductions vers les réseaux de Petri et le formalisme Biocham.

La traduction vers le modèle de Thomas a fait l'objet de deux publications : (Folschette, Paulevé, Inoue, Magnin & Roux, 2012*a*) et (Folschette, Paulevé, Inoue, Magnin & Roux, 2012*b*) et d'une soumission en journal en cours de *review*. La traduction depuis les réseaux discrets asynchrones a, quant à elle, été publiée dans (Folschette, Paulevé, Magnin & Roux, 2013).

Le sujet du chapitre courant est l'expressivité des Frappes de Processus, dans les différentes sémantiques présentées au chapitre 3, par rapport à d'autres modélisations couramment utilisées. Ce travail de positionnement comporte un intérêt évident lorsqu'il s'agit d'interagir avec d'autres types de modèles. Ainsi, les Frappes de Processus peuvent bénéficier des capacités d'analyse d'autres modélisations, moyennant une traduction, qui peuvent s'avérer efficaces pour certains problèmes. À l'inverse, traduire un formalisme en Frappes de Processus permet par exemple de pouvoir y appliquer les méthodes de représentation et d'analyse proposées, et notamment l'analyse statique détaillée au chapitre 4.

Nous présentons ainsi à la section 5.1 une traduction des réseaux discrets asynchrones en Frappes de Processus canoniques. Cette traduction est fondamentale pour l'étude des réseaux de régulation biologique car elle permet la traduction d'un modèle très utilisé directement en un formalisme de Frappes de Processus qu'il est possible d'étudier à l'aide de l'analyse statique. La section 5.2 propose, à l'inverse, une méthode pour inférer l'ensemble des modèles de Thomas sous-jacents à un modèle de Frappes de Processus canoniques. L'inférence du graphe des interactions, par exemple, conserve certaines propriétés comme la présence de cycles exploitables par les résultats relatifs aux conjectures de Thomas. Par ailleurs, l'inférence des paramètres discrets permet de restreindre l'espace des possibles en matière de paramétrisations compatibles avec la dynamique du modèle étudié, facilitant ainsi la recherche de la paramétrisation représentant le comportement recherche. Les modèles de Thomas compatibles avec ces deux inférences sont ceux dont la paramétrisation est complétée lorsque certains paramètres n'ont pas pu être inférés ; nous proposons une méthode pour les énumérer en accord toujours avec la dynamique de modèle de Frappes de Processus canoniques étudié.

Nous nous intéressons par ailleurs aux liens avec d'autres modélisations qui ne sont pas totalement asynchrones, à l'aide des Frappes de Processus avec actions plurielles. Nous montrons notamment à la section 5.3 que ce type de Frappes de Processus est équivalent aux plus classiques automates synchronisés. Nous donnons pour cela les deux traductions adéquates et exhibons les preuves correspondantes. Par ailleurs, nous proposons aussi à la section 5.4 une traduction des Frappes de Processus avec actions plurielles vers les réseaux de Petri utilisant des arcs de lecture et des arcs inhibiteurs. Cette traduction permet de montrer le lien d'inclusion entre les deux formalismes, et ouvre des perspectives dans l'utilisation des réseaux de Petri pour l'analyse des réseaux de régulation biologique représentés sous la forme de Frappes de Processus. Pour finir, nous proposons à la section 5.5 une traduction des systèmes d'équations tels que définis dans le formalisme de Biocham vers les Frappes de Processus avec actions plurielles. Nous nous intéressons naturellement à la sémantique booléenne de Biocham, qui est par ailleurs asynchrone et qui ajoute une couche supplémentaire d'indéterminisme sur chaque réaction (en autorisant la consommation d'une partie arbitraire des réactifs). Cette traduction tire naturellement parti des similitudes entre Biocham et les Frappes de Processus avec actions plurielles, et permet d'affirmer qu'un tel système d'équations biochimiques peut toujours être exprimé en Frappes de Processus avec actions plurielles, et donc être analysé avec les outils adéquats.

Les résultats de ce chapitre sont représentés par les flèches fines reliant les différentes formes de Frappes de Processus aux autres formalismes discrets représentés à la figure 1.1 en page 11.

5.1 Traduction depuis les réseaux discrets asynchrones

Nous proposons dans cette section une traduction des réseaux discrets asynchrones en Frappes de Processus canoniques, et nous en montrons la validité par une preuve de bisimulation faible. Les réseaux discrets asynchrones ont été préalablement définis à la définition 2.6 en page 22 du chapitre 2. Il s'agit de modèles proches du modèle de Thomas, mais en comportant pas de restriction unitaire de la dynamique, et présentant des fonctions d'évolution en lieu et place de paramètres discrets. Un réseau discret asynchrone se présente sous la forme d'un couple $\mathsf{RDA} = (\mathcal{G} ; F)$ où $\mathcal{G} = (\Gamma ; E)$ est un graphe des interactions et F

est un ensemble de fonctions $f_x : \mathcal{G}^{-1}(x) \to [\![0\,;l_x]\!]$ pour tout composant $x \in \Gamma$, où $\mathcal{G}^{-1}(x)$ est l'ensemble des prédécesseurs de x dans le graphe des interactions. La dynamique d'un tel réseau est la suivante : il existe une transition $s \to_{\text{RDA}} s'$ si et seulement si un unique composant x évolue entre s et s' de façon à ce que : $s'[x] = f_x(s)$. Les Frappes de Processus canoniques, définies à la section 4.1 en page 67, permettent une représentation presque immédiate des réseaux discrets asynchrones, à condition de créer les sortes coopératives adéquates.

La traduction proposée à la définition 5.1 associe deux sortes à chaque composant a dans RDA :

- une sorte du même nom pour représenter ce composant,
- une sorte coopérative f^a représentant sa fonction d'évolution f_a, et dont les états sont donc une combinaison des états de ses régulateurs.

De plus, les actions primaires sont définies de façon à correctement mettre à jour les sortes coopératives, et les actions secondaires le sont de façon à ce que chaque sorte coopérative f^a interagisse avec son composant a de la façon dont la fonction d'évolution correspondante le permet. Nous montrons de plus au théorème 5.1 que le modèle obtenu est faiblement bisimilaire au réseau discret asynchrone d'origine.

Définition 5.1 (Frappes de Processus équivalentes (toPH))**.** Soit RDA $= (\mathcal{G}\,;F)$ un réseau discret asynchrone, avec $\mathcal{G} = (\Gamma\,;E)$. On note toPH(RDA) $= (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$ les Frappes de Processus canoniques équivalentes à RDA, définies par :

- $\Sigma = \Gamma \cup \{f^a \mid a \in \Gamma\}$,
- $\mathcal{L} = \bigotimes_{a\in\Gamma} \mathcal{L}_a \times \bigotimes_{a\in\Gamma} \mathcal{L}_{f^a}$ l'ensemble des états , où :

$$\forall a \in \Gamma, \mathcal{L}_a = \{a_i \mid i \in [\![0\,;l_a]\!]\}$$
$$\forall a \in \Gamma, \mathcal{L}_{f^a} = \begin{cases} \mathcal{L}^{\Diamond}_{\mathcal{G}^{-1}(a)} & \text{si } \mathcal{G}^{-1}(a) \neq \emptyset \\ \{f^a_\emptyset\} & \text{sinon} \end{cases},$$

- $\mathcal{H}^{(1)} = \{b_k \to f^a_\sigma \Rsh f^a_{\sigma'} \mid a \in \Gamma \wedge b \in \mathcal{G}^{-1}(a) \wedge b_k \in \mathcal{L}_b \wedge f^a_\sigma \in \mathcal{L}_{f^a} \wedge \sigma[b] \neq b_k \wedge \sigma' = \sigma \Cap b_k\}$,
- $\mathcal{H}^{(2)} = \{f^a_\sigma \to a_j \Rsh a_k \mid a \in \Gamma \wedge f^a_\sigma \in \mathcal{L}_{f^a} \wedge a_j, a_k \in \mathcal{L}_a \wedge j \neq k \wedge f_a([\![\sigma]\!]) = k\}$.

Pour tout état $s \in \mathcal{S}$ de RDA, $(\!|s|\!) = \overline{s}$ est l'état correspondant dans toPH(RDA), défini par : $\forall a \in \Gamma, s[a] = k \Rightarrow \overline{s}[a] = a_k$ et $\forall a \in \Gamma, \overline{s}[f^a] = f^a_\sigma$ avec $f^a_\sigma \in \mathcal{L}_{f^a}$ et $\forall b \in \mathcal{G}^{-1}(a), \sigma[b] = \overline{s}[b]$.
À l'inverse, pour tout état $\overline{s} \in \mathcal{L}^{\Diamond}$ de toPH(RDA), $[\![\overline{s}]\!] = s$ est l'état correspondant dans RDA avec : $\forall a \in \text{sortes}(\overline{s}), \overline{s}[a] = a_k \Rightarrow s[a] = k$.

Théorème 5.1 (RDA $\approx$ toPH(RDA))**.** *Soit* RDA $= (\mathcal{G}\,;F)$ *un réseau discret asynchrone. On a :*

1. $\forall s, s' \in \mathcal{S},\ s \to_{\text{RDA}} s' \implies (\!|s|\!) \rightsquigarrow_{\mathcal{PH}} (\!|s'|\!)$, *où* $\rightsquigarrow_{\mathcal{PH}}$ *est une séquence finie de transitions* $\to_{\mathcal{PH}}$.
2. $\forall \overline{s}, \overline{s}' \in \mathcal{L},\ \overline{s} \to_{\mathcal{PH}} \overline{s}' \implies [\![\overline{s}]\!] = [\![\overline{s}']\!] \vee [\![\overline{s}]\!] \to_{\text{RDA}} [\![\overline{s}']\!]$

Démonstration. On pose : toPH(RDA) $= (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$.

(1) Soient $s, s' \in \mathcal{S}$ tels que $s \to_{\mathsf{RDA}} s'$. Cela signifie qu'il existe un composant $a \in \Gamma$ tel que : $s'[a] = f_a(\sigma)$ et $\forall b \in \Gamma, b \neq a \Rightarrow s[b] = s'[b]$, où $\sigma \in \mathcal{L}^{\Diamond}_{\mathcal{G}^{-1}(a)}$ tel que $\sigma \subseteq (\!|s|\!)$. Posons : $j = s[a]$ et $k = s'[a]$; d'après la définition 5.1, il existe une action $f^a_\sigma \to a_j \Rsh a_k \in \mathcal{H}^{(2)}$ avec $f_a(\sigma) = k$ Par définition de $(\!|s|\!)$, on a $a_k \in (\!|s|\!)$ et $f^a_\sigma \in (\!|s|\!)$, et aucune action de $\mathcal{H}^{(1)}$ n'est jouable dans $(\!|s|\!)$; ainsi, h est jouable dans $(\!|s|\!)$, d'où : $(\!|s|\!) \to_{\mathcal{PH}} (\!|s|\!) \cdot h$. De plus, d'après le lemme 4.1 en page 72, $(\!|s|\!) \cdot h \rightsquigarrow_{\mathcal{PH}} \mathsf{update}((\!|s|\!) \cdot h)$. Enfin, comme update met à jour des sortes coopératives, on a : $\mathsf{update}((\!|s|\!) \cdot h) = (\!|s'|\!)$.

(2) Soient $\overline{s}, \overline{s}' \in \mathcal{L}$ tels que $\overline{s} \to_{\mathcal{PH}} \overline{s}'$. Cela signifie qu'il existe une action $h \in \mathcal{H}$ telle que : $s' = s \cdot h$. Si $h \in \mathcal{H}^{(1)}$, alors $[\![\overline{s}]\!] = [\![\overline{s}']\!]$, par définition de $[\![\overline{s}]\!]$. En revanche, si $h \in \mathcal{H}^{(2)}$, alors d'après la définition 5.1, il existe $a \in \Gamma$, $\sigma \in \mathcal{L}^{\Diamond}_{\mathcal{G}^{-1}(a)}$ et $a_j, a_k \in \mathcal{L}_a$, tels que : $h = f^a_\sigma \to a_j \Rsh a_k$, avec $f_a([\![\sigma]\!]) = k$ et $j \neq k$. Ainsi, $\forall b \in \mathcal{G}^{-1}(a)$, $\sigma[b] = b_i \Rightarrow [\![\overline{s}]\!][b] = i$. D'où : $[\![\overline{s}]\!] \to_{\mathsf{RDA}} [\![\overline{s}']\!]$ car $f_a([\![\sigma]\!]) = k$. □

5.2 Inférence du modèle de Thomas

Partant d'un modèle de Frappes de Processus canoniques, il peut être intéressant d'obtenir le modèle de Thomas sous-jacent. Une telle traduction n'est pas triviale pour plusieurs raisons :

- Les deux formalismes possèdent des approches différentes de la dynamique : les Frappes de Processus sont beaucoup plus atomiques que le modèle de Thomas, alors que celui-ci s'intéresse à des influences globales entre composants ;
- Certains comportements permis par les Frappes de Processus ne le sont pas par un modèle de Thomas ;
- La représentation par paramètres discrets nécessite d'analyser la dynamique pour les retrouver — là où l'inférence de portes logiques serait facilitée par l'analyse des sortes coopératives.

Nous donnons dans cette section une méthode permettant d'inférer le graphe des interactions (section 5.2.2) et les paramètre discrets (section 5.2.3) du modèle de Thomas sous-jacent à des Frappes de Processus canoniques. Nous donnons aussi plusieurs critères pour que cette inférence soit possible. Il est à noter néanmoins que cette inférence peut être partielle, notamment au niveau des paramètres discrets qui ne peuvent parfois pas être inférés car le comportement représenté par les Frappes de Processus n'est pas entièrement traduisible en modèle de Thomas. Cependant, nous proposons aussi une méthode d'énumération des paramétrisations compatibles (section 5.2.4) qui se base sur :

- les paramètres déjà inférés,
- l'existence d'une partie de la dynamique des Frappes de Processus considérées permettant de justifier le paramètre,
- plusieurs critères de modélisation pertinents.

Enfin, nous proposons à la section 5.2.5 quelques pistes d'implémentation de notre méthode en programmation par ensemble de réponses, un paradigme de programmation logique déclarative qui est particulièrement adapté à la représentation et à la résolution de problèmes combinatoire, typiquement de complexité NP. Nous illustrons notamment cette partie à l'aide du problème de l'énumération des paramétrisations compatibles avec un ensemble partiel de paramètres inférés.

Cette traduction est naturellement intéressante à rapprocher de la traduction inverse, permettant d'obtenir la dynamique généralisée d'un modèle de Thomas en Frappes de Processus standards (Paulevé et al., 2011*a*, p. 176).

Les résultats présentés dans cette section ont fait l'objet de deux publications (Folschette, Paulevé, Inoue, Magnin & Roux, 2012*b*; Folschette, Paulevé, Inoue, Magnin & Roux, 2012*a*)[1]. De plus, l'inférence des paramètres a été reprise d'un travail préexistant (Paulevé et al., 2011*a*).

Dans toute cette section, nous considérons un modèle de Frappes de Processus canoniques $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;(\mathcal{H}^{(1)}\,;\mathcal{H}^{(2)}))$.

5.2.1 Inférence du graphe des interactions

Un graphe des interactions (définition 2.1 en page 17) est une représentation abstraite des influences directes, positives ou négatives, entre les composants d'un système. Comme discuté à la section 2.1.3 en page 22, le graphe des interactions permet de caractériser efficacement les propriétés dynamiques globales du système, à l'aide notamment de résultats comme les conjectures de Thomas, qui apportent des résultats sur la présence d'oscillations ou d'états stables multiples.

Dans le cas d'un processus de modélisation d'un réseau de régulation biologique, le modèle de Thomas est le point de départ de la spécification du modèle. Cependant, il est courant que le graphe des interactions initialement conçu contienne des influences qui n'ont pas d'impact sur la dynamique. La méthode que nous proposons dans la suite s'appuie directement sur la dynamique d'un modèle de Frappes de Processus canoniques, ce qui produit des graphes des interactions minimaux, et permet d'affiner les conclusions de telles méthodes d'analyse statique.

L'intuition de cette inférence est que seuls les composants (les sortes dans Γ) figureront dans le graphe des interactions ; les sortes coopératives (dans Δ) sont uniquement étudiées pour comprendre les actions « indirectes » entre composants.

5.2.1.1 Frappes de Processus bien-formées

Nous notons que dans cette section, les indices des processus de composants possèdent une importance particulière, notamment pour contraindre le fait que la dynamique doit être unitaire (critère 5.1). Autrement dit, si on suppose que ces indices représentent des niveaux d'expression discrets ordonnés, par exemple si b_0, b_1 et b_2 représentent le fait que le composant b est présent respectivement en faible, moyenne et forte concentration, alors une action de la forme $a_1 \to b_0 \not\Uparrow b_2$ n'est pas autorisée ; en revanche, deux actions $a_1 \to b_0 \not\Uparrow b_1$ et $a_1 \to b_1 \not\Uparrow b_2$ le sont. Naturellement, toute autre relations d'ordre entre les indices est admissible, à condition qu'une contrainte d'unicité similaire puisse être définie.

$$\Gamma = \{a \in \Sigma \mid \nexists b_i \to a_j \not\Uparrow a_k \in \mathcal{H}, |j-k| > 1\} \tag{5.1}$$

Critère 5.1 (Dynamique unitaire)**.** Toutes les actions secondaires de $\mathcal{PH}$ ne font pas de bond à plus d'un processus d'écart : $\forall a_i \to b_j \not\Uparrow b_k \in \mathcal{H}^{(2)}, |j-k| = 1$

[1] Ce travail a été réalisé dans le cadre d'une collaboration avec Katsumi Inoue, marquée par un stage doctoral de trois mois en 2012 dans l'Inoue Laboratory, au National Institute of Informatics (Tokyo, Japon).

Exemple. Nous nous intéresserons dans la suite à l'inférence du graphe des interactions des deux modèles de Frappes de Processus canoniques représentés aux figures 5.1 et 5.2. Ceux-ci possèdent tous deux une dynamique unitaire selon le critère 5.1, et sont donc compatibles avec l'inférence du graphe des interactions proposée à la section suivante. Il est à noter notamment que les actions de mise à jour de la sorte coopérative du modèle de la figure 5.2 ne sont pas unitaires, mais ces actions ne sont pas prises en compte par ce critère.

Remarque. Le critère 5.1 est naturellement vérifié pour tout modèle booléen, c'est-à-dire tel que : $\forall a \in \Gamma, |\mathcal{L}_a| = 2$.

Remarque. Il est possible de ne pas prendre en compte le critère 5.1, à condition de s'affranchir de l'aspect unitaire de la dynamique du modèle de Thomas. Les résultats de cette section restent alors théoriquement applicables.

Nous considérons dans la suite que les Frappes de Processus canoniques $\mathcal{PH}$ respectent le critère 5.1.

5.2.2 Inférence des interactions

L'inférence de cette section est directement inspirée des travaux de Richard (2010), qui déduit les influences d'un graphe des interactions en fonction des évolutions des différents composants, selon l'état de ses régulateurs.

Pour tout composant a, les *prédécesseurs* de a, notés $\mathsf{pred}(a)$, sont toutes les sortes ayant au moins une action frappant a. Les *régulateurs* de a, en revanche, notés $\mathsf{reg}(a)$, sont tous les composants qui influent sur a, soit directement, soit à travers une sorte coopérative. Il est à noter que les régulateurs définis de cette manière seront potentiellement des régulateurs de a dans le modèle de Thomas inféré, tels que définis en page 17, ce qui explique pourquoi ces deux définitions sont proches.

$$\forall a \in \Gamma, \mathsf{pred}(a) \stackrel{def}{=} \{b \in \Sigma \mid \exists h \in \mathcal{H}, \mathsf{sorte}(\mathsf{frappeur}(h)) = b \wedge \mathsf{sorte}(\mathsf{cible}(h)) = a\}$$

$$\forall a \in \Gamma, \mathsf{reg}(a) \stackrel{def}{=} \{\mathsf{comp}^1(b) \mid b \in \mathsf{pred}(a)\}$$

Où $\mathsf{comp}^1(b)$ fait référence aux composants qui régulent la sorte coopérative b, autrement dit aux sortes que b représente (cf. définition 4.4 en page 69).

L'étude des influences d'un composant b régulant un autre composant a nécessite d'étudier le groupe de régulateurs de b qui vont influencer conjointement a. Ces groupes de régulateurs sont aisément déterminés en observant les sortes coopératives. Nous proposons ici de définir les groupes de régulateurs comme étant les composants connexes d'un graphe reliant tous les régulateurs de a qui sont représentés par une même sorte coopérative :

$$\forall a \in \Gamma, X(a) \stackrel{def}{=} \mathcal{C}\big((\mathsf{reg}(a), \{\{b, c\} \subset \mathsf{comp}^1(v) \mid v \in \mathsf{pred}(a) \cap \Delta\})\big)$$

Où $\mathcal{C}(G)$ représente l'ensemble des composantes connexes du graphe non orienté G.

Pour étudier l'influence d'un groupe de régulateurs g sur un composant a, nous effectuons une analyse exhaustive de toutes les configurations possibles de g. Pour cela, il est nécessaire de définir un sous-état σ sur les sortes de g, et et de compléter ce sous-état par les processus de sorte coopérative qui représentent l'état des composants dans g. Nous définissons pour cela l'ensemble $\mathsf{allFocals}_g^a(\sigma)$ qui contient l'état des sortes de g, celui de a, et celui de toutes les sortes coopératives frappant a.

$$\forall a \in \Gamma, \forall g \in X(a), \forall \sigma \in \mathcal{L}^{\Diamond}_{g \cup \{a\}},$$
$$\mathsf{allFocals}^{a}_{g}(\sigma) = \{\sigma[b] \mid b \in \mathsf{pred}(a) \cap \Gamma\} \cup \{\mathsf{focals}^{1}_{\sigma}(b) \mid b \in \mathsf{pred}(a) \cap \Delta\}$$

Avec :

$$\mathsf{focals}^{1}_{\sigma}(b) = \mathsf{focals}^{1}_{s \Cap \sigma}(b)$$

où le choix de $s \in \mathcal{L}$ est indifférent d'après le point (iv) de la définition 4.5 en page 69.

Enfin, il est possible d'étudier localement la dynamique de a en fonction du sous-état σ d'un groupe de régulateurs g donné ; cette dynamique locale se concentre donc uniquement sur les actions frappant a. En effet, en faisant varier l'un des composants $b \in g$ et en observant le résultat sur l'évolution de a (tendance à l'augmentation ou à la diminution de son niveau d'expression), il est possible d'en déduire l'influence locale de b sur a pour un niveau d'expression de b donné. Pour cela, nous appelons $B_a(\sigma)$ l'ensemble des processus vers lesquels a peut évoluer depuis le sous-état σ ; naturellement, si aucune action ne frappe a dans σ, alors $B_a(\sigma) = \sigma[a]$.

$$\forall g \in X(a), \forall \sigma \in \mathcal{L}^{\Diamond}_{g \cup \{a\}}, B_a(\sigma) \stackrel{def}{=} \begin{cases} F_a(\sigma) & \text{si } F_a(\sigma) \neq \emptyset \\ \{\sigma[a]\} & \text{si } F_a(\sigma) = \emptyset \end{cases}$$
$$\text{où : } F_a(\sigma) \stackrel{def}{=} \{a_k \in \mathcal{L}_a \mid \exists b \in \Sigma, \exists b_i \to a_j \Rsh a_k \in \mathcal{H}, \{b_i, a_j\} \subset \mathsf{allFocals}^{a}_{g}(\sigma)\}$$

La proposition 5.1 détaille l'inférence de toutes les influences locales existant entre les composants, c'est-à-dire celles qui se produisent pour un seuil donné t. L'idée principale derrière cette inférence est la suivante : s'il existe une une influence positive (resp. négative) d'un composant b sur un autre composant a, alors augmenter le niveau d'expression de b va potentiellement faire faire augmenter (resp. diminuer) le niveau d'expression de a, au moins dans certaines configurations (équation (5.2)). Ainsi, ces influences locales se séparent en influences positives et négatives, ce qui représente de potentiels arcs dans le graphe des interactions final. De plus, l'étude des influences sur les groupes de régulateurs d'un composant a permet aussi d'étudier les auto-influences de a (équation (5.3)) ce qui permettra potentiellement d'inférer des auto-arcs. Finalement, il est nécessaire d'étudier le cas particulier où a ne possède pas de régulateurs (équation (5.4)). Nous notons que cette méthode ignore naturellement tous les cas où il n'est pas possible de distinguer une influence d'un composant sur un autre.

Proposition 5.1 (Inférence des influences)**.** *Nous définissons l'ensemble* $\hat{E}_+$ *(resp.* $\hat{E}_-$*) des* influences locales positives *(resp.* négatives*) pour tout composant* $a \in \Gamma$ *par :*

$$\begin{aligned}
&\forall b \in \mathrm{reg}(a), \forall s \in \{+,-\}, b \xrightarrow{t+1} a \in \hat{E}_s \\
&\qquad \iff \exists g \in X(a), b \in g, \exists \sigma \in \mathcal{L}^{\Diamond}_{g\cup\{a\}}, \{b_t, b_{t+1}\} \subset \mathcal{L}_b \wedge \qquad (5.2)\\
&\qquad\quad b_t \in \sigma, \exists a_j \in B_a(\sigma), \exists a_k \in B_a(\sigma\{b_{t+1}\}), s = \mathsf{signe}(k-j) \\
&\forall s \in \{+,-\}, a \xrightarrow{t+1} a \in \hat{E}_s \\
&\qquad \iff \exists g \in X(a), \exists \sigma \in \mathcal{L}^{\Diamond}_{g\cup\{a\}}, \{a_t, a_{t+1}\} \subset \mathcal{L}_a \wedge a_t \in \sigma, \qquad (5.3)\\
&\qquad\quad \exists a_j \in B_a(\sigma), \exists a_k \in B_a(\sigma\{a_{t+1}\}), s = \mathsf{signe}(k-j) \\
&\forall s \in \{+,-\}, a \xrightarrow{t+1} a \in \hat{E}_s \\
&\qquad \iff \mathrm{reg}(a) = \emptyset \wedge \{a_t, a_{t+1}\} \subset \mathcal{L}_a, \qquad (5.4)\\
&\qquad\quad \exists a_j \in B_a(\langle a_t\rangle), \exists a_k \in B_a(\langle a_{t+1}\rangle), s = \mathsf{signe}(k-j)
\end{aligned}$$

La fonction signe sur les entiers relatifs a été définie à la section 1.6 en page 12.

Nous sommes alors en mesure d'inférer les arcs du graphe des interactions final, à partir de ces ensembles d'influences locales positives et négatives. En effet, nous pouvons inférer une influence (globale) positive ou négative d'un composant vers un autre s'il n'existe que des influences locales correspondantes du même signe. Une influence non-signée est inférée si, à l'inverse, il existe au moins deux influences locales correspondantes de signes différents. Enfin, le seuil de chaque influence (quel que soit son signe) est égal au seuil minimum pour lequel une influence locale a été trouvée. Nous formalisons cette inférence dans la proposition 5.2.

Proposition 5.2 (Inférence du graphe des interactions)**.** *Nous inférons* $\mathcal{G} = (\Gamma; E)$ *à l'aide de la proposition 5.1 comme suit :*

$$\begin{aligned}
E_+ &= \{a \xrightarrow{+,t} b \mid \nexists a \xrightarrow{t'} b \in \hat{E}_- \wedge t = \min\{r \mid a \xrightarrow{r} b \in \hat{E}_+\}\} \\
E_- &= \{a \xrightarrow{-,t} b \mid \nexists a \xrightarrow{t'} b \in \hat{E}_+ \wedge t = \min\{r \mid a \xrightarrow{r} b \in \hat{E}_-\}\} \\
E_\circ &= \{a \xrightarrow{\circ,t} b \mid \exists a \xrightarrow{t'} b \in \hat{E}_+ \wedge \exists a \xrightarrow{t''} b \in \hat{E}_- \\
&\qquad\qquad \wedge t = \min\{r \mid a \xrightarrow{r} b \in \hat{E}_- \cup \hat{E}_+\}\}
\end{aligned}$$

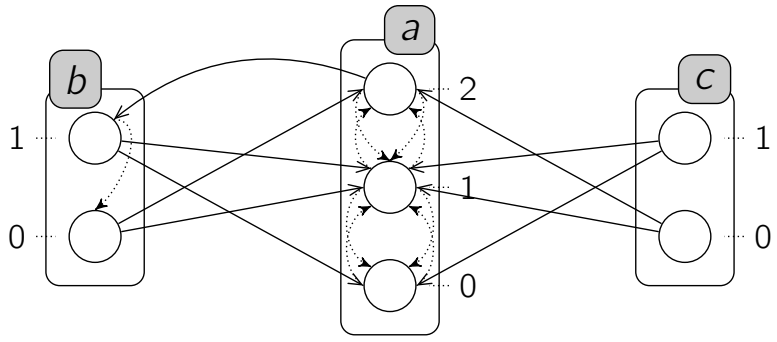


Figure 5.1 – Exemple de Frappes de Processus canoniques avec trois composants : a, b et c. Ce modèle ne comporte aucune sorte coopérative. La dynamique de ce modèle est unitaire car elle respecte bien le critère 5.1 en page 95. L'inférence du graphe des interactions peut donc être effectuée sur ce modèle.

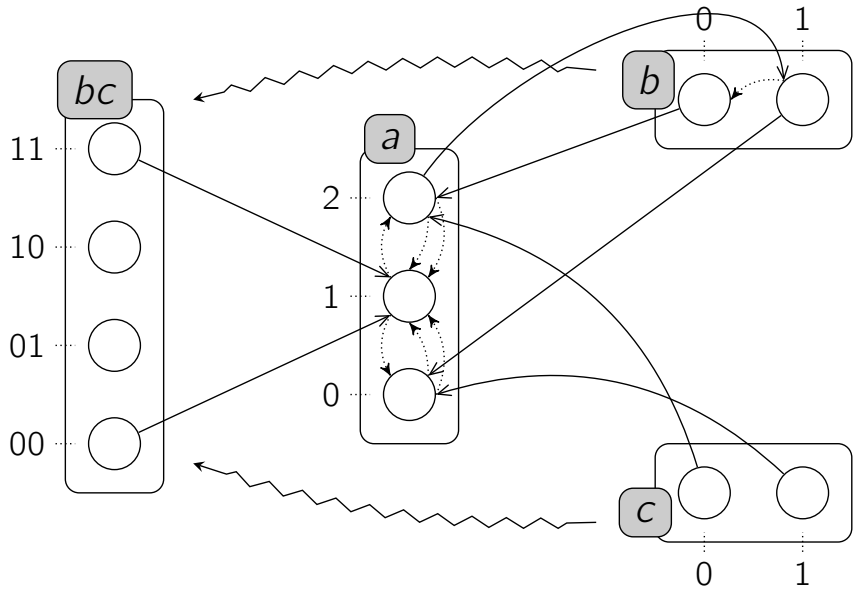


Figure 5.2 – Raffinement du modèle de Frappes de Processus canoniques de la figure 5.1 à l'aide d'une sorte coopérative bc : les deux actions $b_1 \to a_1 \Rsh a_2$ et $c_1 \to a_1 \Rsh a_2$ sont ainsi remplacées par une action $bc_{11} \to a_1 \Rsh a_2$; de même, les actions $b_0 \to a_1 \Rsh a_0$ et $c_1 \to a_1 \Rsh a_0$ sont remplacées par $bc_{00} \to a_1 \Rsh a_0$. Ce modèle possède aussi une dynamique unitaire d'après le critère 5.1.

Exemple. L'application de l'inférence du graphe des interactions aux Frappes de Processus canoniques de la figure 5.1 donne le graphe représenté à la figure 5.3, contenant les arcs suivants :

$$E_+ = \{b \xrightarrow{+1} a, c \xrightarrow{+1} a, a \xrightarrow{+1} a, b \xrightarrow{+1} b, c \xrightarrow{+1} c\}$$
$$E_- = \{a \xrightarrow{-2} b\} \qquad E_\circ = \emptyset$$

Ce graphe des interactions est proche de celui qui avait été proposé à la figure 2.1(gauche) en page 18 bien qu'il ne soit pas équivalent, car chaque composant comporte une auto-action positive. Les auto-actions sur b et c sont la conséquence d'une stabilité globale sur plusieurs sous-états : en effet, c n'évolue jamais, et b n'évolue pas non plus lorsque a_2 n'est pas actif. L'auto-action sur a est principalement causée par sa nature multi-valuée.

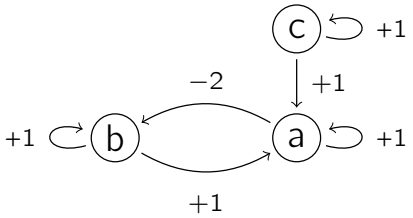


Figure 5.3 – Graphe des interactions inféré depuis les Frappes de Processus de la figure 5.2.

Exemple. Le modèle raffiné de Frappes de Processus canoniques représenté à la figure 5.2 comporte une sorte coopérative bc qui permet de modéliser la coopération de b_1 et c_1 pour faire bondir a_1 en a_2, ainsi que la coopération de b_0 et c_0 pour faire bondir a_1 en a_0. Ces deux coopérations étaient représentées dans le modèle de la figure 5.1 par des actions indépendantes. Cependant, ce modèle raffiné produit le même résultat en termes

de graphe des interactions, c'est-à-dire le graphe de la figure 5.3, ce qui s'explique par le fait que les composants b et c produisent le même type de régulation sur a dans les deux cas (avec des actions indépendantes ou à travers une sorte coopérative).

Exemple. L'ajout d'une action $a_2 \to b_0 \not\Rightarrow b_1$ aux Frappes de Processus canoniques raffinées de la figure 5.2 modifie le résultat de l'inférence. En effet, dans ce cas deux arcs non-signés vers b sont inférés en lieu et place des arcs signés précédents :

$$E_+ = \{b \xrightarrow{+1} a, c \xrightarrow{+1} a, a \xrightarrow{+1} a, c \xrightarrow{+1} c\}$$
$$E_- = \emptyset \qquad\qquad E_\circ = \{a \xrightarrow{\circ 2} b, b \xrightarrow{\circ 1} b\}$$

Cela est dû au fait que les actions $a_2 \to b_1 \not\Rightarrow b_0$ et $a_2 \to b_0 \not\Rightarrow b_1$ introduisent des oscillations causées uniquement par le processus a_2, ce qui implique une influence locale à la fois positive et négative, et est impossible à représenter au sein d'un modèle de Thomas.

5.2.3 Inférence des paramètres

Une fois obtenu le graphe des interactions inféré selon la méthode proposée à la section précédente, il est ensuite possible d'inférer une partie des paramètres discrets propres à un modèle de Thomas, en fonction de la dynamique des Frappes de Processus canoniques d'origine. Cette inférence repose à nouveau sur une exploration exhaustive des comportements possibles du modèle en fonction de l'état des prédécesseurs de chaque composant. Cependant, cette inférence peut être partielle si le comportement modélisé ne peut pas être représenté à l'aide d'un modèle de Thomas.

Ces résultats sont équivalents à ceux présentés par Paulevé et al. (2011*a*), auxquels nous ajoutons la notion de *Frappes de Processus bien formées pour l'inférence des paramètres*, définie au critère 5.2, et qui stipule que pour toute régulation de a par b, tous les processus de $\mathsf{niveaux}(b \to a)$ (resp. $\overline{\mathsf{niveaux}}(b \to a)$) possèdent la même influence sur a.

Critère 5.2 (Frappes de Processus bien formées pour l'inférence des paramètres)**.** Des Frappes de Processus canoniques sont *bien formées pour l'inférence des paramètres* si et seulement si leur dynamique est unitaire (critère 5.1) et si le graphe des interactions $(\Gamma ; E)$ inféré par proposition 5.2 vérifie :

$$\forall a \in \Gamma, \forall b \in \mathcal{G}^{-1}(a), \forall N \in \{\mathsf{niveaux}(b \to a), \overline{\mathsf{niveaux}}(b \to a)\}, \forall i, j \in N,$$
$$\forall c \in \Sigma, ((b \neq a \wedge c = a) \vee (\exists \upsilon \in \mathsf{pred}(a), c \in \mathsf{conn}^1(\upsilon) \wedge b \in \mathsf{comp}^1(c)),$$
$$b_i \to c_k \not\Rightarrow c_l \in \mathcal{H} \Leftrightarrow b_j \to c_k \not\Rightarrow c_l \in \mathcal{H}$$

On souhaite dans la suite inférer le paramètre discret $K_{a,\omega}$, pour un composant $a \in \Gamma$ et un ensemble $\omega \subset \mathcal{G}^{-1}(a)$ de ressources donnés. Cette inférence se base, à l'instar de l'inférence du graphe des interactions, sur une analyse exhaustive des sous-états des régulateurs de a. Pour chaque sorte $b \in \mathcal{G}^{-1}(a)$, on définit une configuration $C^b_{a,\omega}$ (équation (5.5)) qui recense tous les processus qui interagissent avec a dans tous les sous-états représentés par l'ensemble de ressources ω. La configuration d'une sorte coopérative υ régulant a est l'ensemble des processus focaux correspondant à ces sous-états (équation (5.6)). Enfin, $C_{a,\omega}$ fait référence à l'union de toutes ces configurations (équation (5.7)).

$$\forall b \in \Gamma,\ C^b_{a,\omega} \stackrel{def}{=} \begin{cases} \mathsf{niveaux}(b \to a) & \text{si } b \in \omega, \\ \overline{\mathsf{niveaux}}(b \to a) & \text{si } b \notin \omega, \\ L_b & \text{sinon} \end{cases} \tag{5.5}$$

$$\forall \upsilon \in \mathsf{pred}(a) \cap \Delta, C^\upsilon_{a,\omega} \stackrel{def}{=} \{\mathsf{focals}^1_\sigma(\upsilon) \mid \sigma \in \bigotimes_{b \in \mathsf{comp}^1(\upsilon)} C^b_{a,\omega}\} \tag{5.6}$$

$$C_{a,\omega} \stackrel{def}{=} \bigcup_{b \in \mathsf{pred}(a)} C^b_{a,\omega} \tag{5.7}$$

Pour inférer le paramètre recherché, nous calculons les *processus focaux* de a, qui sont les processus vers lesquels tend le niveau d'expression de a en présence de certains autres processus (définition 5.2). Ainsi, $\mathsf{foc}(a, S, T)$ donne l'ensemble des processus de a accessibles en partant de n'importe quel processus dans S, et à condition de ne jouer que des actions dont le frappeur est dans T. Cette notion se base sur un graphe recensant tous les bonds que peuvent faire les processus de a ; si ce graphe est acyclique, alors l'ensemble des processus focaux est l'ensemble des processus de a qui ne sont pas frappés — et vers lesquels a va avoir tendance à évoluer.

Définition 5.2 ($\mathsf{foc}(a, S, T)$)**.** L'ensemble des *processus focaux* de $a \in \Gamma$ depuis $S \subset \mathcal{L}_a$ pour le sous-état $T \subset$ **Proc** est donné par :

$$\mathsf{foc}(a, S, T) \stackrel{def}{=} \begin{cases} \{a_i \in V \mid \nexists (a_i, a_j) \in E\} & \text{si } (V, E) \text{ est acyclique,} \\ \emptyset & \text{sinon} \end{cases}$$

où (V, E) est le graphe orienté suivant :

$$E \stackrel{def}{=} \{(a_j\,;a_k) \in (\mathcal{L}_a \times \mathcal{L}_a) \mid \exists b_i \to a_j \Rsh a_k \in \mathcal{H}^{(2)}, b_i \in T \wedge a_j \in S\}$$
$$V \stackrel{def}{=} S \cup \{a_k \in \mathcal{L}_a \mid \exists (a_j\,;a_k) \in E\}$$

Le paramètre $K_{a,\omega}$ détermine les niveaux d'expression vers lesquels tend a en présence de la configuration $C_{a,\omega}$. Cette valeur peut être calculée à l'aide de foc qui permet justement de retrouver les processus focaux en présence de certaines ressources. Ainsi, on peut en conclure que $K_{a,\omega} = \mathsf{foc}(a, C^a_{a,\omega}, C_{a,\omega})$ dans tous les cas où cette valeur est un intervalle non vide (proposition 5.3).

Proposition 5.3 (Inférence des paramètres)**.** *Soient $\mathcal{PH} = (\Sigma, \mathcal{L}, \mathcal{H})$ des Frappes de Processus bien formées pour l'inférence des paramètres, $\mathcal{G} = (\Gamma, E)$ le graphe des interactions inféré pour $\mathcal{PH}$ et $\omega \subset \mathsf{Res}_a()$ un ensemble de ressources de a. Si $\mathsf{foc}(a, C^a_{a,\omega}, C_{a,\omega})$ est un intervalle non vide, avec $\mathsf{foc}(a, C^a_{a,\omega}, C_{a,\omega}) = [\![a_i\,;a_j]\!]$, alors $K_{a,\omega} = [\![i\,;j]\!]$.*

Exemple. Si on l'applique aux Frappes de Processus de la figure 5.1, la méthode d'inférence des paramètres donnée dans cette section est conclusive sur la majorité des para-

mètres et donne :

$$K_{a,\emptyset} = [\![0\,;0]\!] \qquad K_{b,\emptyset} = [\![0\,;0]\!]$$
$$K_{a,\{a\}} = [\![0\,;0]\!] \qquad K_{b,\{a\}} = [\![0\,;0]\!]$$
$$K_{a,\{c\}} = [\![1\,;1]\!] \qquad K_{b,\{b\}} = [\![1\,;1]\!]$$
$$K_{a,\{b\}} = [\![1\,;1]\!] \qquad K_{b,\{a,b\}} = [\![0\,;0]\!]$$
$$K_{a,\{b,c\}} = [\![1\,;1]\!] \qquad K_{c,\emptyset} = [\![0\,;0]\!]$$
$$K_{a,\{a,b,c\}} = [\![2\,;2]\!] \qquad K_{c,\{c\}} = [\![1\,;1]\!]$$

Seuls les paramètres $K_{a,\{a,b\}}$ et $K_{a,\{a,c\}}$ n'ont pas pu être inférés. Cela est la conséquence directe de l'absence de coopération explicite entre b et c sur a, ce qui crée des oscillations sur a lorsque b_1 et c_0 ou b_0 et c_1 sont présents simultanément.

Exemple. Le modèle raffiné de Frappes de Processus canoniques de la figure 5.2 comporte une coopération explicite entre b et c à l'aide de la sorte coopérative bc. Cela permet notamment à l'inférence d'être davantage conclusive ; ainsi, les résultats sont les mêmes que dans l'exemple précédent, en dehors des deux paramètres suivants dont la valeur a pu être inférée :

$$K_{a,\{a,b\}} = [\![1\,;1]\!] \qquad K_{a,\{a,c\}} = [\![1\,;1]\!]$$

En observant la proposition 5.3, on constate que l'inférence de certains paramètres peut ne pas être possible. Cela peut être notamment dû à des coopérations mal définies entre les régulateurs d'un composant : lorsque deux régulateurs frappent un même composant de façon indépendante, leurs actions peuvent avoir des effets opposés, créant des oscillations dans la dynamique. Un tel indéterminisme ne peut pas être représenté à l'aide d'un modèle de Thomas étant donné que dans une configuration donnée des ressources, un composant possède un unique attracteur, représenté par le paramètre discret correspondant, et ne peut donc évoluer que dans une seule direction. Il est possible de résoudre ces cas non conclusifs (autrement dit, de supprimer ces comportements oscillants) en raffinant le modèle à l'aide de suppressions d'actions ou en s'assurant que les coopérations sont correctement définies à l'aide de sortes coopératives afin d'éviter des influences opposées depuis des régulateurs concurrents.

5.2.4 Énumération des paramétrisations admissibles

Lors de la construction d'un modèle de Thomas, trouver la paramétrisation compatible avec le comportement désiré est nécessaire pour obtenir un modèle complet. Cependant, cette étape possède une complexité inhérente à ce type de formalisme, car le nombre de paramètres que contient le modèle croît exponentiellement dans la taille du graphe des interactions (plus précisément, dans le nombre de régulations vers chaque composant). La méthode d'inférence des paramètres présentée précédemment permet cependant d'obtenir certaines informations sur ces paramètres en fonction de la dynamique des Frappes de Processus canoniques étudiées. Ces informations permettent donc de restreindre l'espace des paramétrisations possibles, et donc d'obtenir plus facilement le modèle recherché.

En d'autres termes, lors de l'inférence d'un modèle de Thomas selon la méthode décrite précédemment, il arrive que certains paramètres ne puissent pas être inférés. Le modèle obtenu est alors partiel, et correspond à un ensemble plus ou moins large de modèles complets. En énumérant les valeurs possibles de chaque paramètre, il est envisageable de retrouver l'ensemble des modèles *compatibles* avec ces valeurs.

Nous délimitons tout d'abord la validité d'un paramètre (critère 5.3) afin d'assurer que toutes les transitions dans le modèle de Thomas résultant sont permises par la dynamique des Frappes de Processus canoniques étudiées. Cette propriété est vérifiée en s'assurant, pour chaque configurations de ressources possibles, de l'existence d'une frappe faisant bondir le processus d'une sorte vers le paramètres correspondant. Ainsi, conjointement avec le fait que les Frappes de Processus étudiées sont bien formées pour l'inférence des paramètres, nous assurons que pour toute transition dans le modèle de Thomas inféré, il existe une transition équivalente dans les Frappes de Processus d'origine. Nous remarquons par ailleurs que les paramètres inférés à l'aide de la proposition 5.3 en page 101 vérifient déjà cette propriété.

Critère 5.3 (Validité d'un paramètre)**.** Un paramètre $K_{a,\omega}$ est *valide* pour les Frappes de Processus $\mathcal{PH}$ si et seulement si :

$$\forall a_i \in C^a_{a,\omega}, a_i \notin K_{a,\omega} \Longrightarrow (\exists c_k \to a_i \overset{\curvearrowright}{} a_j \in \mathcal{H}, c_k \in C^c_{a,\omega} \\ \wedge a_i < K_{a,\omega} \Rightarrow j > i \wedge a_i > K_{a,\omega} \Rightarrow j < i)$$

Nous utilisons de plus plusieurs contraintes de modélisation (tirées de Bernot, Cassez, Comet, Delaplace, Müller & Roux, 2007) afin d'assurer une cohérence des paramètres avec les signes des régulations du graphe des interactions préalablement inféré. L'*hypothèse des valeurs extrêmes* (critère 5.4) stipule que les niveaux extrêmes d'un composant a (c'est-à-dire 0 et l_a) doivent chacun apparaître dans au moins un paramètre. L'*hypothèse d'activité* (critère 5.5) stipule en outre que toutes les régulations doivent être fonctionnelles, c'est-à-dire que pour chaque régulateur d'un composant, il existe au moins une configuration dans laquelle la présence ou l'absence de ce régulateur modifie le paramètre considéré. Enfin, l'*hypothèse de monotonicité* (critère 5.6) stipule qu'ajouter un activateur (resp. inhibiteur) aux ressources d'un composant ne peut qu'augmenter (resp. diminuer) la valeur du paramètre considéré. La relation d'ordre $\leq_{[\![]\!]}$ entre deux paramètres discrets s'applique à des segments et est définie à la section 1.6 en page 12.

Critère 5.4 (Hypothèse des valeurs extrêmes)**.** Soit $\mathcal{G} = (\Gamma, E)$ un graphe des interactions. Une paramétrisation K sur $\mathcal{G}$ satisfait l'*hypothèse des valeurs extrêmes* si et seulement si :

$$\forall b \in \Gamma, \mathcal{G}^{-1}(b) \neq \emptyset \Longrightarrow \exists \omega \subset \mathcal{G}^{-1}(b), 0 \in K_{b,\omega} \wedge \exists \omega' \subset \mathcal{G}^{-1}(b), l_b \in K_{b,\omega'}$$

Critère 5.5 (Hypothèse d'activité)**.** Soit $\mathcal{G} = (\Gamma, E)$ un graphe des interactions. Une paramétrisation K sur $\mathcal{G}$ satisfait l'*hypothèse d'activité*

$$\forall b \in \Gamma, \forall a \in \mathcal{G}^{-1}(b), \exists \omega \subset \mathcal{G}^{-1}(b), K_{b,\omega} \neq K_{b,\omega \cup \{a\}}$$

Critère 5.6 (Hypothèse de monotonicité)**.** Soit $\mathcal{G} = (\Gamma, E)$ un graphe des interactions. Une paramétrisation K sur $\mathcal{G}$ satisfait l'*hypothèse de monotonicité* si et seulement si :

$$\forall b \in \Gamma, \forall A^+ \subset \{a \in \Gamma \mid a \xrightarrow{+,t} b \in E_+\}, \forall A^- \subset \{a \in \Gamma \mid a \xrightarrow{-,t} b \in E_-\}, \\ K_{b,\omega \cup A^-} \leq_{[\![]\!]} K_{b,\omega \cup A^+}$$

Exemple. Les paramètres inférés pour le modèle de Frappes de Processus canoniques de la figure 5.1 sont donnés en page 101. Cette paramétrisation est partielle car les paramètres $K_{a,\{a,b\}}$ et $K_{a,\{a,c\}}$ ne peuvent pas être inférés à l'aide de la proposition 5.3. Il est cependant possible d'énumérer les paramétrisations compatibles avec les paramètres déjà trouvés et les critères 5.3 à 5.6 présentés dans cette section. Cette énumération produit 9 paramétrisations différentes qui correspondent aux trois valeurs possibles pour chacun des paramètres qui n'ont pas pu être inférés :

$$K_{a,\{a,b\}} \in \{[\![1\,;1]\!], [\![1\,;2]\!], [\![2\,;2]\!]\}$$
$$K_{a,\{a,c\}} \in \{[\![1\,;1]\!], [\![1\,;2]\!], [\![2\,;2]\!]\}$$

Nous notons la propriété suivante pour toutes ces solutions : $0 \notin K_{a,\{a,b\}} \wedge 0 \notin K_{a,\{a,c\}}$. Cela est causé par l'hypothèse de monotonicité (critère 5.6) qui stipule notamment dans ce cas précis :

$$K_{a,\{b\}} \leq_{[\![]\!]} K_{a,\{a,b\}} \wedge K_{a,\{c\}} \leq_{[\![]\!]} K_{a,\{a,c\}}$$

Enfin, nous notons que $[\![1\,;1]\!]$ appartient aux valeurs possibles des deux paramètres. Cette énumération permet donc notamment bien de retrouver les valeurs attendues, inférées sur le modèle raffiné de la figure 5.2.

5.2.5 Pistes d'implémentation

Nous abordons dans cette section quelques principes de programmation par ensemble de réponse, aussi appelée ASP pour *Answer Set Programming* (Baral, 2003). Cette forme de programmation logique déclarative possède le double avantage de canaliser la réflexion autour d'un problème sur sa modélisation, et non sur sa résolution (comme pour les techniques habituelles de programmation) tout en proposant des outils de résolution puissants. La résolution des problèmes formulés dans le cadre de ce travail est par exemple effectuée à l'aide du *solver* **Clingo** (Gebser, Kaminski, Kaufmann & Schaub, 2014) qui possède des heuristiques efficaces pour la résolution de nombreux problèmes complexes.

ASP a été utilisé dans le cadre de l'inférence du modèle de Thomas pour la résolution des inférences individuelles du graphe des interactions et des paramètres, mais aussi pour l'énumération des paramétrisations compatibles dans le cas où certains paramètres ne peuvent être inférés. Nous illustrons d'ailleurs l'application d'ASP à l'énumération des paramétrisations compatibles, car cette partie de l'implémentation présente des exemples illustratifs intéressants, du fait notamment qu'ASP est particulièrement adapté à l'énumération de solutions.

La définition d'un problème en ASP suit généralement trois étapes principales illustrées dans la suite :

- la définition des données du problème,
- la création d'ensembles de réponse,
- le filtrage des réponses non satisfaisantes.

Les méthodes de modélisation en ASP ont été abondamment illustrées au cours de nombreux tutoriels (Gebser & Schaub, 2013).

5.2.5.1 Syntaxe des règles simples

La définition d'un programme en ASP repose sur des règles de la forme :

$$\underbrace{H}_{\text{tête}} \leftarrow \underbrace{A_1, A_2, \ldots, A_n, \neg B_1, \neg B_2, \ldots, \neg B_m}_{\text{corps}}.$$

Dans une telle règle, le *corps* et consiste en un ensemble d'atomes (de la forme A_i) et de négation d'atomes (de la forme $\neg B_i$). Dans le cas de *règles simples*, la *tête* comprend un unique atome (H). Schématiquement, une telle règle stipule que si les atomes $A_1, A_2, \ldots, A_n$ sont vrais et qu'il n'est pas possible d'établir que les atomes $B_1, B_2, \ldots, B_m$ sont vrais (négation par l'échec), alors H doit être vrai. La sémantique d'une règle sera détaillée à la section 5.2.5.2.

Un atome est composé d'un prédicat et d'une série d'arguments (potentiellement vide), c'est-à-dire de la forme :

$$p(x_1, x_2, \ldots, x_r)$$

où p est le *prédicat* et $x_1, x_2, \ldots, x_r$ sont les *arguments*. La valeur r, qui détermine le nombre d'arguments d'un atome, est appelée *arité*. Chaque argument peut être une *constante*, qui représente une donnée fixe (typiquement un nom de composant, un niveau d'expression...) ou une *variable*, qui peut prendre la valeur de n'importe quelle constante, comme détaillé à la section 5.2.5.3. Dans cet exemple, nous restreignons les constantes à des valeurs numériques où à des lettres minuscules (p. ex. a, b, c, 1, 2, ...) tandis que les variables seront représentées par une lettre majuscule (p. ex. A, P, Q, ...).

Exemple. Considérons le modèle de Frappes de Processus canoniques de la figure 5.1. Dans la suite, nous utiliserons l'atome d'arité 2 suivant :

$$component(x, n)$$

pour établir la présence d'un composant x dans le modèle, dont le niveau d'expression maximal est $l_x = n$.

5.2.5.2 Réponses et ensembles de réponse

Un programme ASP est un ensemble de règles selon la syntaxe décrite ci-dessus. Résoudre un programme ASP signifie trouver un *ensemble de réponse*, qui est un ensemble minimal d'atomes respectant toutes les règles. Afin de définir formellement cette notion d'ensemble de réponse, nous appelons *règle positive* toute règle ne contentant pas de négation d'atome (notée $\neg$), et *programme positif* tout programme ne contenant que des règles positives. Une règle positive a donc la forme suivante :

$$H \leftarrow A_1, A_2, \ldots, A_n.$$

Si S est un ensemble d'atomes, une règle positive est dite *satisfaite* par S si et seulement si : $H \in S \vee \exists i \in [\![1\,;n]\!], A_i \notin S$. Étant donnée cette définition de la satisfaction d'une règle, l'ensemble de réponse d'un programme positif Π est est l'ensemble minimal (unique) d'atomes S satisfaisant toutes les règles de Π.

Dans le cas général, tous les programmes ne sont pas positifs. Dans le cas d'un programme non positif Π et d'un ensemble d'atomes S, on note Π^S la *réduction* de Π par S, définie en :

- supprimant toutes les règles possédant une négation d'atome $\neg B_i$ dans leur corps telle que $B_i \in S$,
- supprimant toutes les négations d'atomes dans le corps des règles restantes.

L'ensemble d'atomes S est alors qualifié d'ensemble de réponse du programme non positif Π si et seulement si il s'agit d'un ensemble de réponse pour le programme positif Π^S. Nous notons que plusieurs ensembles de réponse peuvent être solutions du même programme non défini. Dans la pratique, un *solver* tel que Clingo peut être configuré pour les énumérer tous ou pour n'en trouver qu'un.

Exemple. Considérons le programme ASP suivant :

$$a \leftarrow \neg b.$$
$$b \leftarrow \neg a.$$

Ce programme n'est pas positif, et il contient les deux ensembles de réponse suivants : $\{a\}$ et $\{b\}$.

Il est à noter qu'il est possible de définir une règle ne comportant pas de corps (c'est-à-dire comportant zéro atomes et négations d'atomes). Une telle règle est aussi appelée un *fait*, et son atome de tête appartient par conséquence à tous les ensembles de réponse du programme. ainsi, les données représentant le modèle à étudier (à savoir le modèle de Frappes de Processus canoniques original, mais aussi le graphe des interactions et les paramètres inférés) sont exprimés en ASP à l'aide de faits.

Exemple. Afin de définir les trois composants contenus dans la figure 5.1, nous utilisons le programme suivant :

$$component(a, 2).$$
$$component(b, 1).$$
$$component(c, 1).$$

Ce programme ne contient que des faits utilisant le prédicat Γ précédemment mentionné à la page précédente. Par ailleurs, les symboles a, b, c, 1 et 2 représentent des constantes. Ce programme produit un seul ensemble de réponse, qui contient simplement les trois atomes qui y sont mentionnés.

5.2.5.3 Variables

L'utilisation de variables, déjà évoquée précédemment, permet de généraliser des règles. Par exemple, pour décrire les ensembles de tous les niveaux d'expression de chaque composant (c'est-à-dire l'ensemble $[\![0\,;l_a]\!]$ pour tout $a \in component$), il est envisageable d'utiliser un atome de la forme : $component\_levels(a, k)$ pour stipuler que : $k \in [\![0\,;l_a]\!]$. Les variables viennent alors en aide pour énumérer toutes les constantes entières k correspondant à un niveau d'expression possible : avant la résolution, une étape de *ground* permet de remplacer, dans chaque règle, toutes les variables par toutes les constantes possibles ; en d'autres termes, ce processus permet d'obtenir un programme équivalent sans variable. La règle suivante, par exemple, contient trois variables (A, K et M) et permet d'énumérer l'ensemble des niveaux d'expression possibles de chaque composant dans le modèle :

$$component\_levels(A, K) \leftarrow component(A, M), 0 \leq K \leq M.$$

où la notation $\leq$ possède la même signification que l'opérateur mathématique.

Exemple. Afin de représenter tous les niveaux d'expression des composants du modèle de la figure 5.1, nous utilisons le programme suivant, qui reprend les faits du programme proposé à la page ci-contre :

$$component(a, 2).$$
$$component(b, 1).$$
$$component(c, 1).$$
$$component\_levels(A, K) \leftarrow component(A, M), 0 \leq K \leq M.$$

L'ensemble de réponse suivant est alors obtenu :

$$\{ \quad component(a, 2), \quad component(b, 1), \quad component(c, 1), \quad component\_levels(a, 0), \quad component\_levels(a, 1), \quad component\_levels(a, 2), \quad component\_levels(b, 0), \quad component\_levels(b, 1), \quad component\_levels(c, 0), \quad component\_levels(c, 1). \quad \}$$

5.2.5.4 Règles de cardinalité

En tant qu'extension des règles simples, les *règles de cardinalité* s'avèrent utiles pour énumérer des ensembles de réponse. La tête d'une règle de cardinalité spécifie un ensemble d'atomes H et deux entiers min et max, et on note une telle règle :

$$min \{ H \} max \leftarrow A_1, A_2, \ldots, A_n, \neg B_1, \neg B_2, \ldots, \neg B_m.$$

Pour chaque règle de ce type, autant d'ensembles de réponse sont créés, de façon à ce que chaque ensemble de réponse S vérifie :

$$min \leq |S \cap H| \leq max$$

et chaque atome $H_i \in S \cap H$ respecte la règle simple suivante :

$$H_i \leftarrow A_1, A_2, \ldots, A_n, \neg B_1, \neg B_2, \ldots, \neg B_m.$$

en d'autres termes, tous les ensembles de réponse générés contiennent un sous-ensemble de l'ensemble H dont la cardinalité est comprise entre min et max, et pour lesquels la condition dans le corps de la cardinalité est vérifiée. L'ensemble des atomes $H = \{H_1, H_2, \ldots, H_p\}$ est souvent décrit de la façon suivante : $H = \{P \mid Q\}$, qui est un raccourci pour qualifier l'ensemble des atomes de la forme P et pour lesquels Q est vérifié.

Les règles de cardinalité s'avèrent pratiques pour énumérer toutes les paramétrisations possibles en créant des ensembles de réponse multiples. Pour des raisons d'implémentation, un libellé unique est assigné à chaque ensemble possible de ressources pour un composant donné. Ainsi, dans la suite, nous désignerons par ω_p l'ensemble des ressources d'un composant donné a libellé par p, et K_{a,ω_p} est naturellement le paramètre relatif à cet ensemble. Nous notons que libeller les ensembles de ressources d'un composant est naturellement identique à libeller ses paramètres. Supposons dans la suite que :

- *param_label*(a, p) stipule que p est un libellé valide pour un ensemble de ressources d'un composant a (et en conséquence, K_{a,ω_p} est un paramètre valide) ;
- *param*(a, p, i) signifie : $i \in K_{a,\omega_p}$;
- *infered_param*(a, p) permet de préciser que l'inférence du paramètre K_{a,ω_p} a été conclusive (selon la proposition 5.3).

Il est alors possible d'énumérer les valeurs possibles de chaque paramètre pour lequel la proposition 5.3 n'a pas été conclusive, à l'aide de la règle de cardinalité suivante :

$$1 \{ \mathit{param}(A, P, I) \mid \mathit{component\_levels}(A, I) \} \infty \leftarrow \mathit{param\_label}(A, P), \neg \mathit{infered\_param}(A, P).$$

Cette règle ne s'applique donc qu'à chaque paramètre P de tout composant A (grâce au prédicat *param_label*) dont la valeur est toujours inconnue ($\neg$*infered_param*), et stipule que tout niveau d'expression I du composant considéré (*component_levels*) peut appartenir au paramètre (*param*). De plus, la limite inférieure de cette règle de cardinalité est 1, ce qui force chaque paramètre énuméré à contenir au moins une valeur, mais aucune limite supérieure n'est spécifiée (∞) pour la taille de chaque paramètre, qui est donc limitée par le nombre de niveaux d'expression du composant considéré. En d'autres termes, cette règle de cardinalité crée autant d'ensembles de réponse qu'il n'y a de paramétrisations *candidates* (et non nécessairement admissibles) telles que pour chaque paramètre K_{a,ω_p} qui n'a pas pu être inféré par la proposition 5.3, alors $K_{a,\omega_p} \subset [\![0\,;l_a]\!] \wedge K_{a,\omega_p} \neq \emptyset$. Cette énumération ignore donc totalement la notion de paramétrisation admissible donnée par les critères 5.3 à 5.6 ou même le fait que les paramètres doivent être des intervalles.

Exemple. Dans le cadre du modèle de la figure 5.1, les paramètres $K_{a,\{a,b\}}$ et $K_{a,\{a,c\}}$ ne peuvent pas être inférés à l'aide de la proposition 5.3. La règle de cardinalité précédente permet de produire 49 paramétrisations candidates, dans lesquelles ces deux paramètres peuvent prendre toutes les valeurs possibles suivantes :

$$(K_{a,\{a,b\}}\,;K_{a,\{a,c\}}) \in \{[\![0\,;0]\!], [\![1\,;1]\!], [\![2\,;2]\!], [\![0\,;1]\!], [\![1\,;2]\!], [\![0\,;2]\!], \{0,2\}\}^2$$

les autres paramètres restant identiques aux valeurs qui ont été inférées. Il est donc à noter que $\{0, 2\}$ appartient à l'ensemble des paramétrisations candidates étant donné qu'aucune règle n'a encore été définie pour spécifier qu'un paramètre doit être un intervalle.

5.2.5.5 Contraintes

Enfin, une *contrainte* est une règle sans tête :

$$\leftarrow A_1, A_2, \ldots, A_n, \neg B_1, \neg B_2, \ldots, \neg B_m.$$

Une contrainte est satisfaite si et seulement si son corps n'est pas satisfait, ce qui permet d'invalider un ensemble de réponse contenant une combinaison non désirée d'atomes. En d'autres termes, si le corps d'une contrainte s'avère vrai pour un ensemble d'atomes donné, alors celui-ci ne peut pas être un ensemble de réponse. Dans le cadre de l'énumération des paramètres, par exemple, les contraintes sont particulièrement utiles pour filtrer les paramétrisations non désirées, car contenant des paramètres qui ne sont pas des intervalles, ou ne respectant pas les critères 5.3 à 5.6. Supposons dans la suite que :

- *less_active*(a, p, q) stipule que ω_p est un ensemble de ressources d'un composant a contenant plus d'activateurs et moins d'inhibiteurs que ω_q (de façon large) ;
- *param_inf*(a, p, q) signifie : $K_{a,\omega_p} \leq_{[\![]} K_{a,\omega_q}$.

L'hypothèse de monotonicité (critère 5.6) est alors formulée sous la forme de la contrainte suivante :

$$\leftarrow \textit{less\_active}(A, P, Q), \neg \textit{param\_inf}(A, P, Q).$$

En effet, cette contrainte supprime tous les résultats de paramétrisations où des paramètres K_{A,ω_P} et K_{A,ω_Q} existent tels que A est moins activé par l'ensemble de ressources ω_P qu'il ne l'est par ω_Q, mais où $K_{A,\omega_Q} <_{[\![]} K_{A,\omega_P}$, violant ainsi l'hypothèse de monotonicité. Les autres critères peuvent naturellement être formulés de façon analogue.

Exemple. L'ensemble des paramétrisations candidates données par la règle de cardinalité proposée à la page précédente peut être filtré à l'aide de contraintes. Par exemple, une contrainte permet de n'obtenir que des paramètres intervalles, et d'éliminer les paramètres contenant des « trous » comme $\{0, 2\}$. L'ajout d'une telle contrainte dans le programme réduit l'ensemble des valeurs possibles pour les paramètres $K_{a\{a,b\}}$ et $K_{a,\{a,c\}}$ à :

$$(K_{a,\{a,b\}} ; K_{a,\{a,c\}}) \in \{[\![0 ; 0]\!], [\![1 ; 1]\!], [\![2 ; 2]\!], [\![0 ; 1]\!], [\![1 ; 2]\!], [\![0 ; 2]\!]\}^2$$

Par la suite, du fait que les paramètres $K_{a,\{b\}} = [\![1 ; 1]\!]$ et $K_{a,\{c\}} = [\![1 ; 1]\!]$ ont été inférés, l'hypothèse de monotonicité (critère 5.6 et formulée en ASP ci-dessus) supprime toutes les paramétrisations telles que : $0 \in K_{a,\{a,b\}} \vee 0 \in K_{a,\{a,c\}}$. Au final, les valeurs possibles restantes pour les deux paramètres non inférés sont :

$$(K_{a,\{a,b\}} ; K_{a,\{a,c\}}) \in \{[\![1 ; 1]\!], [\![2 ; 2]\!], [\![1 ; 2]\!]\}^2$$

Tous ces candidats respectent les critères 5.3, 5.4 et 5.5 et cet ensemble n'est donc pas davantage filtré par les contraintes restantes. Nous avons ainsi expliqué comment est obtenu le résultat détaillé en page 104.

Nous avons brièvement décrit dans cette section la façon dont les programmes ASP viennent en aide dans l'expression et la résolution de problèmes complexes. Ce paradigme trouve une application particulièrement intéressante dans l'énumération de paramétrisations compatibles (telle que détaillée à la section 5.2.4) : chaque paramétrisation générée l'est dans un ensemble d'atomes distinct, et des contraintes d'intégrité sont formulées pour filtrer les ensembles qui ne respectent pas les critères d'admissibilité. Les ensembles de réponse restants sont les paramétrisations intéressantes pour ce problème, c'est-à-dire celles respectant tous les critères, ce qui permet de réduire le nombre de paramétrisations à considérer pour obtenir la dynamique voulue pour le modèle. Il est à noter cependant que toutes les étapes d'inférence du modèle de Thomas, y compris l'inférence du graphe des interactions (section 5.2.2) et l'inférence des paramètres (section 5.2.3) ont été codées en ASP afin de bénéficier de l'efficacité de ce paradigme, bien que les approches en soient différentes.

Nous notons pour finir que la complexité générale de cette méthode est exponentielle dans le nombre de régulateurs de chaque composant, et linéaire dans le nombre total de composants. Ainsi, de grands modèles peuvent théoriquement être traités, à la condition que le nombre de régulateurs de chaque composant soit faible, ce qui est généralement le cas des réseaux de régulation biologique.

5.3 Équivalence avec les réseaux d'automates synchronisés

Nous nous intéressons ici au lien entre les Frappes de Processus avec actions plurielles et les réseaux d'automates synchronisés. Nous montrons notamment que ces deux formalismes sont équivalents et nous exhibons pour cela deux traductions d'un formalisme vers l'autre. Cette équivalence est intéressante car elle montre clairement le lien entre ce formalisme de Frappes de Processus et celui plus répandu des réseaux d'automates synchronisés. En effet, chaque action plurielle $A \rightarrowtail B$ possède la même dynamique qu'un ensemble de transitions synchronisées partant chacune d'un processus de l'ensemble A et arrivant dans le processus de la même sorte de l'ensemble $A \Cap B$[2].

Nous rappelons tout d'abord la définition d'un réseau d'automates synchronisés (définition 5.3) ainsi que la relation de transition entre deux états d'un tel modèle (définition 5.4) ce qui permet d'en définir la dynamique.

Définition 5.3 (Réseau d'automates synchronisés)**.** Un *réseau d'automates synchronisés* est un quadruplet $\mathsf{RAS} = (\Sigma\,;\mathcal{L}\,;I\,;T)$ où :

- $\Sigma \stackrel{def}{=} \{a, b, \ldots\}$ est l'ensemble fini et dénombrable des *automates* ;
- $\mathcal{L} \stackrel{def}{=} \bigotimes_{a\in\Sigma} \mathcal{L}_a$ est l'ensemble fini des *états*, où $\mathcal{L}_a = \{a_0, \ldots, a_{l_a}\}$ est l'ensemble fini et dénombrable des *états locaux* de l'automate $a \in \Sigma$ et $l_a \in \mathbb{N}^*$, chaque état local appartenant à un unique automate : $\forall (a_i\,;b_j) \in \mathcal{L}_a \times \mathcal{L}_b, a \neq b \Rightarrow a_i \neq b_j$;
- $I \stackrel{def}{=} \{\ell_1, \ldots, \ell_m\}$ est l'ensemble fini des *libellés* de transitions ;
- $T \stackrel{def}{=} \{a_i \xrightarrow{\ell} a_j \mid a \in \Sigma \wedge a_i \in \mathcal{L}_a \wedge \ell \in I\}$ est l'ensemble fini des *transitions* entre états locaux.

Pour tout libellé $\ell \in I$, on note ${}^\bullet\ell \stackrel{def}{=} \{a_i \mid a_i \xrightarrow{\ell} a_j \in T\}$ et $\ell^\bullet \stackrel{def}{=} \{a_j \mid a_i \xrightarrow{\ell} a_j \in T\}$. L'ensemble des états locaux des automates est dénoté par $\mathbf{\acute{E}L} \stackrel{def}{=} \bigcup_{a\in\Sigma} \mathcal{L}_a$. Enfin, étant donné un état global $s \in \mathcal{L}$, $s(a) = a_i \in \mathcal{L}_a$ fait référence à l'état local de l'automate $a \in \Sigma$.

Définition 5.4 (Sémantique des réseaux d'automates ($\rightarrow_{\mathsf{RAS}}$))**.** Étant donné un réseau d'automates synchronisés $\mathsf{RAS} = (\Sigma\,;\mathcal{L}\,;I\,;T)$, un libellé ℓ est dit *jouable* dans un état $s \in \mathcal{L}$ si et seulement si : $\forall a_i \in {}^\bullet\ell, s(a) = a_i$. Dans ce cas, on note $(s \cdot \ell)$ l'état résultant du jeu de toutes les transitions libellées par ℓ, défini par : $s \cdot \ell = s \Cap \ell^\bullet$. De plus, on note alors : $s \rightarrow_{\mathsf{RAS}} (s \cdot \ell)$.

Remarque. Nous notons que les réseaux d'automates synchronisés ainsi définis sont non-déterministes, tant au niveau global du modèle qu'au niveau local des automates. Cette vision s'oppose à d'autres sémantiques des réseaux d'automates comme celles de Richard (2010) ou de Remy et al. (2008), qui définissent la dynamique de leurs modèles à l'aide de fonctions de transition locales, qui sont par définition déterministes. Ces fonctions ont en effet la forme : $f_a : \mathcal{L} \rightarrow \mathcal{L}_a$ et associent donc à chaque état global du modèle un état local (unique) pour chaque automate. La définition des réseaux d'automates synchronisés que nous proposons ici (définition 5.3) n'empêche en revanche pas l'existence de deux libellés $\ell_1, \ell_2 \in I$ tels que ${}^\bullet\ell_1 = {}^\bullet\ell_2$ mais $\ell_1{}^\bullet \neq \ell_2{}^\bullet$. Cela implique notamment l'existence de deux

[2] La notation $A \Cap B$, formalisée à la définition 5.5 à la page ci-contre, représente l'ensemble où chaque processus de A a été remplacé par le processus de B de la même sorte, s'il existe.

transitions entre état locaux $a_i \xrightarrow{\ell_1} a_j$ et $a_i \xrightarrow{\ell_2} a_k$ avec $a_j \neq a_k$, d'où un non-déterminisme au niveau des automates.

Pour tout modèle de Frappes de Processus avec actions plurielles $\mathcal{PH}$, la définition 5.6 propose une traduction de $\mathcal{PH}$ en un réseau d'automates synchronisés $\mathsf{toRAS}(\mathcal{PH})$ équivalent, et le théorème 5.2 établit la bisimilarité entre les deux modèles. La notation $\Cap$ qui est utilisée dans la définition qualifie le recouvrement d'un ensemble de processus de sortes distinctes par un autre comprenant uniquement des processus issus des mêmes sortes (définition 5.5). Cette notion de recouvrement est une extension du recouvrement d'un état par un ensemble de processus tel que précédemment formalisé à la définition 2.11 en page 28.

Définition 5.5 (Recouvrement ($\Cap : \mathbf{Proc}^{\Diamond} \times \mathbf{Proc}^{\Diamond} \to \mathbf{Proc}^{\Diamond}$))**.** Étant donné un sous-état désordonné $ps \in \mathbf{Proc}^{\Diamond}$ et un processus $a_i \in \mathbf{Proc}$, tel que $a \in \mathsf{sortes}(ps)$, on définit : $(ps \Cap a_i) = (ps \setminus \mathcal{L}_a) \cup \{a_i\}$. On étend de plus cette définition au recouvrement par un ensemble de processus de sortes distinctes $ps' \in \mathbf{Proc}^{\Diamond}$ tel que $\mathsf{sortes}(ps') \subset \mathsf{sortes}(ps)$ comme étant le recouvrement de ps par chaque processus de ps' : $ps \Cap ps' = ps \underset{a_i \in ps'}{\Cap} a_i$.

Définition 5.6 (Réseau d'automates équivalent (toRAS))**.** Le réseau d'automates synchronisés équivalent aux Frappes de Processus avec actions plurielles $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ est défini par : $\mathsf{toRAS}(\mathcal{PH}) = (\Sigma\,;\mathcal{L}\,;I\,;T)$, où :

- $I = \{\ell_h \mid h \in \mathcal{H}\}$;
- $T = \{a_i \xrightarrow{\ell_h} a_j \mid h \in \mathcal{H} \wedge h = A \rightarrowtail B \wedge a_i \in A \wedge a_j \in A \Cap B\}$.

Théorème 5.2 ($\mathcal{PH} \approx \mathsf{toRAS}(\mathcal{PH})$)**.** *Soient $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ des Frappes de Processus avec actions plurielles. On a :*

$$\forall s, s' \in \mathcal{L}, s \to_{\mathcal{PH}} s' \iff s \to_{\mathsf{toRAS}(\mathcal{PH})} s' \quad .$$

Démonstration. Soient $s, s' \in \mathcal{L}$. On pose : $\mathsf{toRAS}(\mathcal{PH}) = (\Sigma\,;\mathcal{L}\,;I\,;T)$.

($\Rightarrow$) Supposons que $s \to_{\mathcal{PH}} s'$, c'est-à-dire qu'il existe une action $h \in \mathcal{H}$ telle que $s' = s \cdot h$. Posons : $h = A \rightarrowtail B$. D'après la définition 5.6, l'existence de cette action dans $\mathcal{PH}$ implique celle d'un libellé ℓ_h dans $\mathsf{toRAS}(\mathcal{PH})$ ainsi que de l'ensemble de transitions $T_h = \{a_i \xrightarrow{\ell_h} a_j \mid a_i \in A \wedge a_j \in A \Cap B\}$. Autrement dit, ${}^\bullet\ell_h = A$, donc ℓ_h est jouable dans s si et seulement si $A \subseteq s$. De plus, $\ell_h{}^\bullet = \mathsf{invariant}(h) \cup B$, donc $(s \cdot \ell_h) = s \Cap (\mathsf{invariant}(h) \cup B) = s \Cap B = s'$ car $\mathsf{invariant}(h) \subseteq A \subseteq s$.

($\Leftarrow$) Supposons que $s \to_{\mathsf{toRAS}(\mathcal{PH})} s'$, c'est-à-dire qu'il existe un libellé $\ell \in I$ et un ensemble de transitions ayant ce libellé : $T_\ell = \{a_i \xrightarrow{\ell} a_j \in T\}$, tels que $s' = s \cdot \ell$. D'après la définition 5.6, cela signifie notamment qu'il existe une action $h = A \rightarrowtail B \in \mathcal{H}$ telle que $\ell = \ell_h$, et que : $T_\ell = \{a_i \xrightarrow{\ell} a_j \mid a_i \in A \wedge a_j \in A \Cap B\}$. Étant donné que $\mathsf{invariant}(h)$ et $\mathsf{cible}(h)$ forment une partition de A, T_ℓ peut être découpé en deux ensembles, selon les invariants et les cibles de h : $T_\ell = \{a_i \xrightarrow{\ell} a_i \mid a_i \in \mathsf{invariant}(h)\} \cup \{a_i \xrightarrow{\ell} a_j \mid a_i \in \mathsf{cible}(h) \wedge a_j \in B\}$. Ainsi, $s' = s \Cap (\mathsf{invariant}(h) \cup B) = s \Cap B = s \cdot h$. □

Pour finir, nous proposons à la définition 5.7 la traduction inverse d'un réseau d'automates synchronisés RAS en des Frappes de Processus avec actions plurielles équivalentes $\mathsf{toPH}(\mathsf{RAS})$. Le théorème 5.3 stipule que le modèle obtenu est bien bisimilaire au modèle

d'origine. Enfin, le théorème 5.4 résume les résultats de cette section en statuant l'équivalence d'expressivité entre les Frappes de Processus avec actions plurielles et les réseaux d'automates synchronisés.

Définition 5.7 (Frappes de Processus équivalentes (toPH))**.** Les Frappes de Processus avec actions plurielles équivalentes au réseau d'automates synchronisés $\mathsf{RAS} = (\Sigma, \mathcal{L}, I, T)$ sont définies par $\mathsf{toPH}(\mathsf{RAS}) = (\Sigma, \mathcal{L}, \mathcal{H})$, où :

$$\mathcal{H} = \{{}^\bullet\ell \rightarrowtail B \mid \ell \in I \wedge B = \ell^\bullet \setminus \{a_i \in \textbf{ÉL} \mid a_i \xrightarrow{\ell} a_i \in T\}\}$$

Théorème 5.3 ($\mathsf{RAS} \approx \mathsf{toPH}(\mathsf{RAS})$)**.** *Soit* $\mathsf{RAS} = (\Sigma\,;\mathcal{L}\,;I\,;T)$ *un réseau d'automates synchronisés. On a :*

$$\forall s, s' \in \mathcal{L}, s \rightarrow_{\mathsf{RAS}} s' \iff s \rightarrow_{\mathsf{toPH(RAS)}} s' \ .$$

Démonstration. Soient $s, s' \in \mathcal{L}$. On pose : $\mathsf{toPH}(\mathsf{RAS}) = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$.

($\Rightarrow$) Supposons que $s \rightarrow_{\mathsf{RAS}} s'$, c'est-à-dire qu'il existe un libellé $\ell \in I$ et un ensemble de transitions ayant ce libellé : $T_\ell = \{a_i \xrightarrow{\ell} a_j \in T\}$, tels que $s' = s{\cdot}\ell$. D'après la traduction donnée à la définition 5.7, il existe donc une action $h = A \rightarrowtail B \in \mathcal{H}$ telle que $A = {}^\bullet\ell$ et $B = \ell^\bullet \setminus \{a_i \in \textbf{ÉL} \mid a_i \xrightarrow{\ell} a_i \in T\}$. Or $s' = s \Cup \ell^\bullet = s \Cup (B \cup \{a_i \in \textbf{ÉL} \mid a_i \xrightarrow{\ell} a_i \in T\}) = s \Cup B$ car $\{a_i \in \textbf{ÉL} \mid a_i \xrightarrow{\ell} a_i \in T\} \subseteq s$. Ainsi, h est jouable dans s et $s' = s \cdot h$.

($\Leftarrow$) Supposons que $s \rightarrow_{\mathsf{toPH(RAS)}} s'$, c'est-à-dire qu'il existe une action $h = A \rightarrowtail B \in \mathcal{H}$ telle que $s' = s \cdot h$. D'après la traduction de la définition 5.7, cela signifie qu'il existe un libellé $\ell \in I$ et un ensemble de transitions ayant ce libellé : $T_\ell = \{a_i \xrightarrow{\ell} a_j \in T\}$, tels que : $A = {}^\bullet\ell$ et $B = \ell^\bullet \setminus \{a_i \in \textbf{ÉL} \mid a_i \xrightarrow{\ell} a_i \in T\}$. Comme h est jouable dans s, alors $A \subseteq s$, donc ℓ est aussi jouable dans s. De plus, $s' = s \cdot h = s \Cup B = s \Cup (B \cup \mathsf{invariant}(h)) = s \Cup \ell^\bullet$. □

Théorème 5.4 (Équivalence entre réseaux d'automates synchronisés et Frappes de Processus avec actions plurielles)**.** *Les Frappes de Processus avec actions plurielles sont aussi expressives que les réseaux d'automates synchronisés.*

Démonstration. D'après les définitions 5.6 et 5.7 et les théorèmes 5.2 et 5.3 associés, tout modèle de Frappes de Processus avec actions plurielles peut être représenté à l'aide d'un réseau d'automates synchronisés, et inversement. □

5.4 Traduction en réseaux de Petri

Nous présentons dans cette section une méthode de traduction des Frappes de Processus avec actions plurielles en réseaux de Petri avec arcs inhibiteurs et arcs de lecture (Peterson, 1977). Les réseaux de Petri, du nom de leur créateur Carl Adam Petri (1962), permettent une représentation compacte des systèmes discrets concurrents sous la forme de places contenant des jetons qui représentent leur niveau d'expression, et de transitions qui consomment et produisent ces jetons.

La traduction des Frappes de Processus en réseaux de Petri peut s'inscrire dans une volonté plus générale d'utiliser ce formalisme pour représenter des processus de régulation biologique (Chaouiya, 2007). De plus, cela peut permettre d'appliquer des méthodes de

dépliage (Baldan, Busi, Corradini & Michele Pinna, 2000) afin de réduire le parallélisme dans les modèles et faciliter en partie l'analyse par *model checking*.

La traduction proposée dans cette section fait correspondre, pour chaque sorte des Frappes de Processus avec actions plurielles initiales, une place dans le réseau de Petri produit. Les jetons de cette place représentent alors le processus actif dans la sorte correspondante. Les actions sont quand à elles représentées par une transition comportant les arcs ordinaires, de lecture et inhibiteurs nécessaire à reproduire la dynamique recherchée. Il est à noter que toutes les sémantiques de Frappes de Processus (avec actions plurielles, arcs neutralisants ou classes de priorités, mais aussi standards) peuvent être traduites en réseaux de Petri à l'aide de la traduction proposée ici, moyennant les traductions présentées aux chapitres 3 et 4. Cela montre notamment que les Frappes de Processus sont expressivement incluses dans les réseaux de Petri. De plus, l'avantage de la traduction proposée ci-après est de ne représenter qu'une place par sorte, d'être bornée et d'utiliser le système d'accumulation des jetons pour représenter les processus, même dans un cas multi-valué (non-booléen). La bornitude que nous prouvons à la fin de cette section est intéressante car cette propriété est indécidable dans le cas général (Hack, 1976) ; de plus, cela permet de mieux délimiter l'expressivité de ce type de modèle (les réseaux de Petri bornés avec arcs inhibiteurs n'étant pas plus expressifs que ceux qui n'en comportent pas). Cette traduction se distingue donc de celle proposée par Paulevé (2011, p. 47) qui fait correspondre une place à chaque processus, et produit donc un réseau de Petri sauf et n'utilisant pas d'arcs inhibiteurs.

Nous commençons par rappeler la notion de réseau de Petri avec des arcs de lecture et des arcs inhibiteurs (définition 5.8). Un réseau de Petri est composé de *places* et de *transitions*. Chaque place contient à tout instant un certain nombre de *jetons* qui tiennent le rôle de niveau d'expression courant de la place. Les transitions, quant à elles, sont reliées aux places par des *arcs ordinaires*, des *arcs de lecture* et des *arcs inhibiteurs* ; ces arcs déterminent la condition de tir de la transition en fonction des places auxquelles elle est reliée, ainsi que le nombre de jetons consommés et produits par son tir. Les arcs ordinaires permettent d'établir le nombre de jetons consommés et produits par le tir d'une transition, tandis que les arcs de lecture permettent de s'assurer de la présence d'un certain nombre de jetons dans une place et les arcs inhibiteurs permettent de s'assurer de la propriété inverse. Il est donc possible de passer d'un état (appelé *marquage*) à un autre en tirant une transition, et à la condition de respecter ces règles (définition 5.9).

Définition 5.8 (Réseau de Petri (RdP))**.** Un *réseau de Petri* est un 6-uplet $\mathsf{RdP} = (P\,;T\,;Pre\,;Post\,;Lect\,;Inh)$ où :

- P est l'ensemble fini des *places*,
- T est l'ensemble fini des *transitions*,
- $Pre \subset P \times \mathbb{N} \times T$ est l'ensemble des ordinaires arcs entrant dans une transition,
- $Post \subset T \times \mathbb{N} \times P$ est l'ensemble des arcs ordinaires sortant d'une transition,
- $Lect \subset P \times \mathbb{N} \times T$ est l'ensemble des arcs de lecture,
- $Inh \subset P \times \mathbb{N} \times T$ est l'ensemble des arcs inhibiteurs,

avec : $P \cap T = \emptyset$ et $P \cup T \neq \emptyset$. De plus, chaque arc est unique ; autrement dit, pour chacun des ensembles $X \in \{Pre, Post, Lect, Inh\}$, on a :

$$\forall (a\,;i\,;b) \in X, \forall (a'\,;i'\,;b') \in X, (a = a' \wedge b = b') \Rightarrow i = i'$$

On note par ailleurs, pour tout couple d'éléments $a, b \in P \cup T$: $X(p,t) = i$ si $(a\,;i\,;b) \in X$ ou 0 sinon.
Un *marquage* d'un réseau de Petri est une fonction $M : P \to \mathbb{N}$. Un *réseau de Petri avec marquage initial* est un couple $(\mathsf{RdP}\,;M_0)$ où RdP est un réseau de Petri et M_0 est un marquage de RdP.

Définition 5.9 (Sémantique des réseaux de Petri ($\to_{\mathsf{RdP}}$))**.** Étant donnés un réseau de Petri $\mathsf{RdP} = (P\,;T\,;Pre\,;Post\,;Lect\,;Inh)$ et un marquage M, une transition $t \in T$ est dite *jouable* dans M si et seulement si :

$$\forall p \in P, M(p) \geq Pre(p,t) \wedge M(p) \geq Lect(p,t) \wedge M(p) < Inh(p,t)$$

Dans ce cas, on note $(M \cdot t)$ le marquage résultant du jeu de cette transition depuis M, défini par :

$$\forall p \in P, (M \cdot t)(p) = M(p) - Pre(p,t) + Post(t,p)$$

De plus, on note alors : $M \to_{\mathsf{RdP}} (M \cdot t)$.

Nous proposons à la définition 5.10 une traduction des Frappes de Processus avec actions plurielles vers les réseaux de Petri tels que définis plus haut. Cette définition fait correspondre, à chaque sorte du modèle initial, une place dans le réseau de Petri produit, dont le nombre de jetons figurera le niveau d'activité. De même à chaque action correspond une transition, dont le déclenchement est contraint par des arcs de lecture et des arcs inhibiteurs pour s'assurer que le nombre de jetons des places correspondant aux frappeurs est le bon. Enfin, des arcs ordinaires permettent de consommer et produire les jetons nécessaires pour représenter tous les bonds.

Il est à noter que pour cette traduction, les indices des processus ont un sens particulier car ils représentent le nombre de jetons utilisés pour représenter chaque processus. Par exemple, une sorte hypothétique a comprenant trois processus a_0, a_1 et a_2 sera représentée par une place pouvant structurellement contenir jusqu'à 2 jetons (cette contrainte n'étant pas inhérente au modèle mais à la forme des arcs de la traduction). Ainsi, les indices des processus du modèle initial doivent être choisis dans $\mathbb{N}$.

Nous montrons ensuite au théorème 5.5 que cette traduction conserve la dynamique ; autrement dit, que le modèle traduit est fortement bisimilaire au modèle original.

Définition 5.10 (Réseau de Petri équivalent (toRdP))**.** Le réseau de Petri équivalent aux Frappes de Processus avec actions plurielles $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ est défini par : $\text{toRAS}(\mathcal{PH}) = (P\,;T\,;Pre\,;Post\,;Lect\,;Inh)$, où :

- $P = \Sigma$,
- $T = \mathcal{H}$,
- $Pre = \{(b, j-k, h) \mid h \in \mathcal{H} \wedge b_j \in \text{cible}(h) \wedge b_k \in \text{bond}(h) \wedge k - j < 0\}$
- $Post = \{(h, k-j, b) \mid h \in \mathcal{H} \wedge b_j \in \text{cible}(h) \wedge b_k \in \text{bond}(h) \wedge k - j > 0\}$
- $Lect = \{(a, i, h) \mid h \in \mathcal{H} \wedge a_i \in \text{frappeur}(h) \wedge i > 0\}$
- $Inh = \{(a, i+1, h) \mid h \in \mathcal{H} \wedge a_i \in \text{frappeur}(h) \wedge i < l_a\}$

De plus, pour tout état $s \in \mathcal{L}$, on note M^s le marquage correspondant, défini par : $\forall a \in P, M^s(a) = i$ tel que $s[a] = a_i$.

Théorème 5.5 ($\mathcal{PH} \approx \text{toRdP}(\mathcal{PH})$)**.** *Soient $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ des Frappes de Processus avec actions plurielles. On a :*

$$\forall s, s' \in \mathcal{L}, s \to_{\mathcal{PH}} s' \iff M^s \to_{\text{toRdP}(\mathcal{PH})} M^{s'} \quad .$$

Démonstration. Soient $s, s' \in \mathcal{L}$. On pose : $\text{toRdP}(\mathcal{PH}) = (P\,;T\,;Pre\,;Post\,;Lect\,;Inh)$. Pour toute action $h \in \mathcal{H}$, on notera dans la suite :

- $Pre_h = \{(b, j-k, h) \mid b_j \in \text{cible}(h) \wedge b_k \in \text{bond}(h) \wedge k - j < 0\}$,
- $Post_h = \{(h, k-j, b) \mid b_j \in \text{cible}(h) \wedge b_k \in \text{bond}(h) \wedge k - j > 0\}$,
- $Lect_h = \{(a, i, h) \mid a_i \in \text{frappeur}(h) \wedge i > 0\}$,
- $Inh_h = \{(a, i+1, h) \mid a_i \in \text{frappeur}(h) \wedge i < l_a\}$.

($\Rightarrow$) Supposons que $s \to_{\mathcal{PH}} s'$, c'est-à-dire qu'il existe une action $h \in \mathcal{H}$ telle que $s' = s \cdot h$. Posons : $h = A \rightarrowtail B$. D'après la définition 5.10, l'existence de cette action dans $\mathcal{PH}$ implique celle d'une transition $h \in T$ dans $\text{toRdP}(\mathcal{PH})$, ainsi que des arcs suivants : $Pre_h \subset Pre$, $Post_h \subset Post$, $Lect_h \subset Lect$ et $Inh_h \subset Inh$. Or, étant donné que h est jouable dans s, on a : $\forall a_i \in \text{frappeur}(h), a_i \in s$, d'où : $\forall a \in \text{sortes}(\text{frappeur}(h)), M^s(a) = i$. De plus, comme $\text{cible}(h) \subset \text{frappeur}(h)$, on en déduit : $\forall b \in \text{sortes}(\text{cible}(h)), M^s(b) = j > j - k$. Ainsi, h est jouable dans RdP d'après la définition 5.9. De plus, d'après cette même définition, $\forall b \in \text{sortes}(\text{bond}(h)), (M^s \cdot h)(b) = j + (k - j) = k$ et $\forall a \in P \setminus \text{sortes}(\text{bond}(h)), (M^s \cdot h)(a) = M^s(a)$. Ainsi, $(M^s \cdot h) = M^{s \cdot h} = M^{s'}$.

($\Leftarrow$) Supposons que $M^s \to_{\text{toRdP}(\mathcal{PH})} M^{s'}$, c'est-à-dire qu'il existe une transition $h \in T$ telle que $M^{s'} = M^s \cdot h$. D'après la définition 5.10, il existe alors une action $h = A \rightarrowtail B \in \mathcal{H}$ dans $\mathcal{PH}$, ainsi que les ensembles d'arcs suivants dans $\text{toRdP}(\mathcal{PH})$: $Pre_h \subset Pre$, $Post_h \subset Post$, $Lect_h \subset Lect$ et $Inh_h \subset Inh$. Comme h est jouable dans M, cela signifie alors, d'après la définition 5.9, que : $\forall p \in \text{frappeur}(h), M^s(p) = s[p]$ (ainsi que : $\forall p \in \text{cible}(h), M^s(p) = s[p] \geq Pre(p, t)$). Ainsi, h est jouable dans s car $a \subseteq s$. Par ailleurs, $s \cdot h = s \Cap B$; or, $\forall b \in \text{sortes}(\text{bond}(h)), (M^s \cdot h)(b) = j + (k - j) = k$ et $\forall a \in P \setminus \text{sortes}(\text{bond}(h)), (M^s \cdot h)(a) = M^s(a)$. Ainsi, $s \cdot h = s'$. □

Pour terminer, nous montrons au théorème 5.6 que le réseau de Petri obtenu par la traduction d'un modèle de Frappes de Processus avec actions plurielles d'après la définition 5.10 est borné si on lui associe un marquage initial correspondant à un état possible des Frappes de Processus considérées. En réalité, un tel réseau de Petri est toujours borné,

car il n'est pas possible d'ajouter des jetons à l'infini dans une place ; cependant, nous limitons notre résultat aux marquages correspondant à des états possibles pour des raisons de simplicité. Ce résultat est important car il en découle que certaines propriétés deviennent décidables, ce qui n'est pas le cas sur des réseaux de Petri non bornés s'ils comportent des arcs inhibiteurs (Roux & Lime, 2004).

Théorème 5.6 (Bornitude de toRdP($\mathcal{PH}$))**.** *Pour toutes Frappes de Processus avec actions plurielles $\mathcal{PH} = (\Sigma\,;\mathcal{L}\,;\mathcal{H})$ et tout état $s \in \mathcal{L}$,* (toRdP($\mathcal{PH}$)$\,;M^s$) *est k-borné, avec $k = \max_{a\in\Sigma}(|\mathcal{L}_a|)$.*

Démonstration. D'après le théorème 5.5, $\mathcal{PH}$ et toRdP($\mathcal{PH}$) sont bisimilaires. Autrement dit, le marquage de toute place $a \in P$ ne peut pas dépasser son plafond l_a dans $\mathcal{PH}$. □

5.5 Traduction depuis la sémantique booléenne Biocham

Nous proposons dans cette section une traduction des modèles Biocham dans la sémantique booléenne en Frappes de Processus avec actions plurielles. Biocham (pour *Biochemical Abstract Machine*, voir Fages et al., 2004) est un environnement logiciel pour la modélisation et l'analyse de systèmes biochimiques. Il propose notamment une syntaxe basée sur des règles de réactions pour représenter un système de réactions biochimiques, et un simulateur booléen permettant d'exécuter le système. Ce simulateur possède la particularité d'interpréter le modèle comme étant un modèle booléen, où les composants peuvent être présents ou absents, sans prendre en compte les coefficients stœchiométriques ou les paramètres cinétiques éventuels contenus dans les équations.

Notre traduction s'appuie sur les ressemblances entre le formalisme booléen de Biocham et les Frappes de Processus avec actions plurielles, comme discutées au début de la section 3.3. Pour chaque équation de Biocham, cette traduction fait correspondre un ensemble d'actions qui reproduit toutes les dynamiques (non-déterministes) possibles. Nous montrons enfin que cette traduction produit effectivement un modèle équivalent, ce qui montre que les Frappes de Processus sont au moins aussi expressives que la sémantique booléenne de Biocham.

Nous définissons dans la suite la notion de système d'équations biochimiques (définition 5.11) et sa dynamique dans la sémantique booléenne de Biocham (définition 5.12). Une équation biochimique est un triplet de la forme $X \xrightarrow{Y} Z$ où X, Y et Z sont des ensembles de composants ayant respectivement le rôle de réactifs, catalyseurs et produits. La signification intuitive équation est la suivante : si, dans un état donné du système, tous les réactifs et tous les catalyseurs sont présents, alors il est possible de créer tous les produits et de consommer une partie des catalyseurs. Il est à noter que nous cherchons à garder l'expressivité la plus large de cette sémantique, qui autorise de jouer une équation même si certains produits (dans Z) sont déjà présents dans l'état considéré. Par ailleurs, seule une partie des réactifs (dans X) est consommée, et ce de façon non-déterministe, afin de conserver l'ensemble des dynamiques possibles d'un modèle non booléen (et comportant donc des coefficients stœchiométriques).

Définition 5.11 (Système d'équations biochimiques)**.** Un *système d'équations biochimiques* tel que décrit par Biocham est un ensemble de réactions :

$$E = \{X \xrightarrow{Y} Z \mid X \cap Y = Y \cap Z = X \cap Z = \emptyset\}$$

Par ailleurs, on note C^E l'ensemble de tous les composants mentionnés dans E :

$$C^E = \bigcup_{X \xrightarrow{Y} Z \in E} X \cup Y \cup Z$$

Pour toute équation biochimique $X \xrightarrow{Y} Z \in E$, les éléments de X sont appelés les *réactifs*, ceux de Y sont les *catalyseurs* et ceux de Z sont les *produits*.
Un *état* d'un système d'équations biochimiques est un ensemble $S \subset C^E$.

Remarque. La syntaxe véritable de Biocham permet d'intégrer des catalyseurs implicites, en relâchant la contrainte $X \cap Z = \emptyset$; autrement dit, il est possible d'avoir $X \cap Z \neq \emptyset$. Dans ce cas, les éléments de $X \cap Z$ sont aussi des catalyseurs, car ils conditionnent le jeu de la réaction mais ne sont pas modifiés par celle-ci. On peut réécrire de telles équations de la forme suivante, tout en assurant la même dynamique : $(X \setminus Z) \xrightarrow{Y \cup (X \cap Z)} (Z \setminus X)$. Par ailleurs, il est naturellement possible de représenter une réaction d'équilibre $X \xleftrightarrow{Y} Z$ par deux réactions biochimiques $X \xrightarrow{Y} Z$ et $Z \xrightarrow{Y} X$.

Définition 5.12 (Sémantique booléenne d'un système d'équations biochimiques)**.** Soit E un système d'équations biochimiques et $S \subset C^E$ un état de ce système. On note : $S \rightarrow_E S'$ si et seulement si :

$$S \neq S' \wedge \exists X \xrightarrow{Y} Z \in E, X \cup Y \subset S \wedge \exists X' \subset X, S' = (S \setminus X') \cup Z$$

Remarque. Comme expliqué plus haut, cette définition prend compte de la sémantique non-déterministe de Biocham pour ce qui est de la consommation des réactifs. Il est possible de rendre cette définition déterministe (du point de vue de chaque réaction) en imposant $X' = X$ pour toutes les transitions. De même, cette définition ne prend pas en compte la présence éventuelle de produits dans le milieu avant de jouer une équation ; autrement dit, certains des produits d'une équation biochimique peuvent se trouver dans le milieu au moment où elle est jouée.

La définition 5.13 propose une traduction des systèmes d'équations biochimiques en Frappes de Processus avec actions plurielles. Le modèle obtenu comporte autant de sortes booléennes que de composants mentionnés ; autrement dit, pour chaque composant a mentionné, il existe une sorte avec deux processus a_0 et a_1 dans le modèle obtenu. De plus, pour chaque équation biochimique $X \xrightarrow{Y} Z$, un ensemble d'actions est créé, chaque action étant de la forme : $(X_1 \cup Y_1 \cup Z'_0 \cup (Z \setminus Z')_1) \rightarrowtail (X'_0 \cup Z'_1)$, où X et Y sont les réactifs et catalyseurs nécessaires à l'initiation de la réaction, Z' représente les produits qui seront effectivement créés (et $Z \setminus Z'$ représente ceux qui sont déjà présents) et X' représente le sous-ensemble des réactifs qui seront consommés (défini de façon totalement non-déterministe). Enfin, le théorème 5.7 stipule que le modèle obtenu est fortement bisimilaire — et possède donc strictement la même dynamique.

Définition 5.13 (Frappes de Processus équivalentes (toPH))**.** Soit E un système d'équations biochimiques. On note $\mathsf{toPH}(E) = (\Sigma, \mathcal{L}\,; \mathcal{H})$ les Frappes de Processus avec actions plurielles équivalentes, définies par :

- $\Sigma = C^E$,
- $\mathcal{L} = \bigotimes_{a\in\Sigma} \mathcal{L}_a$, où $\forall a \in \Sigma, \mathcal{L}_a = \{a_0, a_1\}$,
- $\mathcal{H} = \{(X_1 \cup Y_1 \cup Z'_0 \cup (Z \setminus Z')_1) \rightarrowtail (X'_0 \cup Z'_1) \mid X \xrightarrow{Y} Z \in E \wedge X' \subset X \wedge Z' \subset Z \wedge (X' \cup Z') \neq \emptyset\}$,

où, pour tout $i \in \{0, 1\}$ et tout $N \subset C^E$, on note : $N_i = \{a_i \mid a \in N\}$.
Enfin, pour tout état $S \subset C^E$, on note $(\!(S)\!) = S_1 \cup (C^E \setminus S)_0$ l'état correspondant dans $\mathsf{toPH}(E)$.

Remarque. Pour supprimer le non-déterminisme de Biocham, tel qu'expliqué à la remarque à la page précédente, il est possible de modifier la définition 5.13 afin d'imposer pour chaque action : $X' = X$, ou de faire les simplifications correspondantes.

Théorème 5.7 ($\mathcal{PH} \approx \mathsf{toPH}(E)$)**.** *Soit E un système d'équations biochimiques. On a :*

$$\forall S, S' \subset C^E, S \to_E S' \iff (\!(S)\!) \to_{\mathsf{toPH}(E)} (\!(S')\!) \ .$$

Démonstration. Posons : $\mathsf{toPH}(E) = (\Sigma, \mathcal{L}\,; \mathcal{H})$. Soient $S, S' \subset C^E$.

($\Rightarrow$) Supposons que $S \to_E S'$. D'après la définition 5.12, cela signifie, outre $S \neq S'$, qu'il existe une équation $X \xrightarrow{Y} Z \in E$ telle que $X \cup Y \subset S$, et qu'il existe un ensemble $X' \subset X$ tel que $S' = (S \setminus X') \cup Z$. Posons $Z' = Z \cap S$ et $h = (X_1 \cup Y_1 \cup Z'_0 \cup (Z \setminus Z')_1) \rightarrowtail (X'_0 \cup Z'_1)$. Comme $S \neq S'$, on note que $X' \neq \emptyset \vee Z' \neq \emptyset$. D'après la définition 5.13, on a donc : $h \in \mathcal{H}$. Par ailleurs, on a alors : $X_1 \cup Y_1 \subseteq (\!(S)\!)$, et par définition de Z', $Z'_0 \subseteq (\!(S)\!)$ et $(Z \setminus Z')_1 \subseteq (\!(S)\!)$, donc h est jouable dans $(\!(S)\!)$. Enfin, $(\!(S')\!) = (\!((S \setminus X') \cup Z)\!) = (\!((S \setminus X') \cup Z')\!) = (\!(S \setminus X')\!) \Cap Z'_1 = ((\!(S)\!) \Cap Z'_1) \Cap X'_0 = (\!(S)\!) \Cap (Z'_1 \cup X'_0)$ car X, Y et Z sont disjoints, et X' et Z' aussi.

($\Leftarrow$) Supposons que $(\!(S)\!) \to_{\mathsf{toPH}(E)} (\!(S')\!)$. Cela signifie qu'il existe une action $h \in \mathcal{H}$ telle que $(\!(S')\!) = (\!(S)\!) \cdot h$. Par ailleurs, l'existence d'une telle action implique, d'après la définition 5.13, l'existence d'une équation $X \xrightarrow{Y} Z \in E$ telle que : $h = (X_1 \cup Y_1 \cup Z'_0 \cup (Z \setminus Z')_1) \rightarrowtail (X'_0 \cup Z'_1)$ avec $X' \subset X$, $Z' \subset Z$ et $(X' \cup Z') \neq \emptyset$. Comme h est jouable dans $(\!(S)\!)$, cela signifie notamment que $X_1 \subseteq (\!(S)\!)$ et $Y_1 \subseteq (\!(S)\!)$ car X, Y et Z sont disjoints ; autrement dit : $X \subset S$ et $Y \subset S$. Enfin, avec le même raisonnement que ci-dessus, on obtient : $(\!(S')\!) = (\!(S)\!) \Cap (Z'_1 \cup X'_0) = (\!((S \setminus X') \cup Z)\!)$, d'où : $S' = (S \setminus X') \cup Z$. □

5.6 Bilan

Nous avons présenté dans ce chapitre plusieurs traductions :

- depuis les Frappes de Processus, vers le modèle de Thomas (section 5.2) et les réseaux de Petri (section 5.4) ;
- vers les Frappes de Processus, depuis les réseaux discrets asynchrones (section 5.1) et les systèmes d'équations biochimiques de la sémantique booléenne de Biocham (section 5.5).

Par ailleurs, nous avons donné une preuve d'équivalence entre les Frappes de Processus et les automates synchronisés (section 5.3) avec les traductions adéquates. Ces différentes traductions sont résumées au sein de la figure 1.1 en page 11.

Nous insistons ici sur le fait que si chacune des traductions exposées ici s'effectue depuis ou vers un formalisme particulier des Frappes de Processus, il est néanmoins possible de naviguer entre les différentes sémantiques de Frappes de Processus sans pertes d'expressivité, comme l'ont montré les différents résultats du chapitre 3. Ainsi, les résultats particuliers proposés dans le présent chapitre se généralisent à toutes les sémantiques de Frappes de Processus, ce qui permet non seulement d'élargir le cadre de ces traductions à ces différentes sémantiques, mais aussi de conclure quant aux rapports d'expressivité entre les Frappes de Processus et les autres types de modélisation abordés. Il est à noter qu'il n'est pas impropre de parler ici de « Frappes de Processus », sans préciser la sémantique, lorsqu'il est question d'expressivité, car les résultats du chapitre 3 ont bien montré leur équivalence au niveau des capacités de représentation — à l'exception notable des Frappes de Processus standards qui ont probablement moins d'expressivité que les autres sémantiques.

Chapitre 6

Applications sur des exemples de grande taille

Nous proposons dans ce chapitre d'appliquer les différents résultats vus au cours de cette thèse à des modèles de grande taille. Les deux principales applications sont les suivantes :

- différents raffinements successifs d'un modèle de récepteur des cellules de croissance épithéliale de 20 composants représenté en Frappes de Processus canoniques sont comparés à l'aide de leur traduction en modèle de Thomas, à l'aide des résultats du chapitre 5.
- la dynamique d'un modèle de récepteur de lymphocytes T de 94 composants est étudiée à l'aide des méthodes d'analyse statique développées au chapitre 4,

Ces approches ont été développées dans (Folschette, Paulevé, Inoue, Magnin & Roux, 2012*b*) et (Folschette, Paulevé, Magnin & Roux, 2013).

Nous avons présenté au chapitre 4 de cette thèse les Frappes de Processus canoniques qui permettent d'unifier les différents formalismes de Frappes de Processus à l'aide d'une classe de modèles particulière. Nous proposons ensuite dans ce même chapitre des méthodes permettant d'analyser la dynamique de cette classe de modèles et au chapitre 5 nous donnons notamment une traduction vers le modèle de Thomas de tout modèle décrit dans ce formalisme.

Afin de montrer l'efficacité des méthodes développées dans le cadre de cette thèse, nous proposons dans cette section d'en illustrer l'utilisation sur des modèles de grande taille. Nous considérons comme des modèles de grande taille les modèles comportant plusieurs dizaines ou centaines de composants — en somme, les modèles dont la taille ne permet actuellement pas d'en étudier le comportement à l'aide de *model checkers* classiques devant calculer l'intégralité de la dynamique. Les exemples que nous donnons ici montrent que nos méthodes peuvent être appliquées à des modèles de grande taille. De plus, nous détaillons la nature et l'utilisation de l'implémentation des différentes méthodes

mises en pratique dans la suite. Ces implémentations sont intégrées à la bibliothèque **Pint**[1] précédemment existante, principalement développée et maintenue par Loïc Paulevé et qui regroupe plusieurs utilitaires et outils propres aux Frappes de Processus. Les implémentations réalisées dans le cadre de cette thèse y ont été intégrées, et sont donc disponibles en ligne et sous une licence libre.

Nous appliquons à la section 6.1.1 notre méthode d'inférence du modèle de Thomas à partir de Frappes de Processus canoniques proposée section 5.2 en page 94 ; nous nous intéressons dans le détail à plusieurs modèles de récepteur de facteur de croissance épidermique contenant 20 composants, puis plus généralement aux résultats fournis sur différents modèles. Nous détaillons ensuite à la section 6.2 l'application de la méthode d'analyse statique présentée au chapitre 4 à un modèle du récepteur de lymphocyte T comprenant 94 composants.

6.1 Traductions vers le modèle de Thomas

6.1.1 Application au récepteur de facteur de croissance épidermique

Nous nous intéressons ici à l'étude d'un modèle de récepteur de facteur de croissant épidermique (Sahin, Frohlich, Lobke, Korf, Burmester, Majety, Mattern, Schupp, Chaouiya, Thieffry, Poustka, Wiemann, Beissbarth & Arlt, 2009). Ce modèle est représenté par un graphe des interactions contenant 20 composants et 52 arcs. Une protéine appelée EGF peut être considérée comme un composant d'entrée car elle ne comporte aucun régulateur, et une chaîne de réactions permet l'activation de la protéine pRB qui est responsable de la régulation de la division cellulaire. Celle-ci est donc essentielle pour la prévention du développement de cancers.

Nous créons trois modèles de Frappes de Processus canoniques à partir du graphe des interactions original, avec différents niveaux de précision dans la modélisation des coopérations entre composants.

- Le modèle (1) représente la dynamique généralisée du graphe des interactions, telle que théorisée par Paulevé et al. (2011*a*), c'est-à-dire sans aucune information concernant les règles booléennes des coopérations, et ne contient donc aucune sorte coopérative (ni aucune action primaire) ;
- Le modèle (2) est partiellement raffiné : il intègre certaines des règles booléennes en question en tant que sortes coopératives, choisies d'après des expériences de *knockdown* menées sur le système ;
- Le modèle (3) est le modèle totalement raffiné contenant toutes les coopérations.

Les modèles (2) et (3) consistent donc en des raffinements successifs du modèle (1) qui est le plus général au niveau de la dynamique. La constitution de chaque modèle est détaillée dans la suite, et les résultats de l'inférence du graphe des interactions et de l'inférence des paramètres sur ces trois modèles sont donnés dans la tableau 6.1 et discutés plus bas. Nous ne nous intéressons ici qu'à des paramètres entiers, autrement dit à des segments réduits à une seule valeur.

[1] **Pint** est une bibliothèque libre, disponible à `http://loicpauleve.name/pint/` accompagnée des modèles mentionnés dans ce chapitre ainsi que des implémentations réalisées au cours de cette thèse.

Modèle	$\|E\|$	$\|K\|$	Paramètres inférés	Modèles possibles	Points fixes
(1)	52	196	20	$2^{176} \simeq 9{,}6 \cdot 10^{52}$	0
(2)	51	192	98	$2^{94} \simeq 2{,}0 \cdot 10^{28}$	0
(3)	51	192	192	1	3

Table 6.1 – Résultats de l'inférence du graphe des interactions et des paramètres sur les trois modèles dérivés du récepteur EGF avec différentes précisions dans la définition des coopérations. Le modèle (1) ne contient aucune coopération entre les composants. Certaines coopérations ont été incluses dans le modèle (2) sous la forme de 14 sortes coopératives et le modèle (3) contient toutes les coopérations entre composants sous la forme de 22 sortes coopératives. La deuxième colonne donne le nombre d'arcs dans le graphe des interactions inféré (le nombre de nœuds étant toujours celui du modèle, c'est-à-dire 20). La troisième colonne donne le nombre total de paramètres à définir (calculé à partir du graphe des interactions), la quatrième colonne donne le nombre de paramètres entiers qui ont pu être inférés, et la cinquième colonne donne le nombre de modèles compatibles avec le modèle de Frappes de Processus canoniques étudié, qui dépend de façon exponentielle du nombre de paramètres n'ayant pu être inférés. Finalement, la dernière colonne donne le nombre de points fixes du modèle, calculé à l'aide du théorème 3.2 en page 47.

Le modèle (1) comprend uniquement des interactions individuelles entre composants, c'est-à-dire des activations ou inhibitions indépendantes d'un composant à l'autre, obtenues d'après les régulations contenues dans le graphe des interactions. Ainsi, le graphe des interactions inféré d'après le modèle (1) est exactement identique au graphe des interactions utilisé pour créer ce modèle, à l'exception d'une auto-activation sur l'entrée EGF, qui est due à son absence de régulateurs, comme expliqué à la section 5.2 en page 94, et se retrouve aussi pour les modèles suivants. Les seuls paramètres ayant pu être inférés sont ceux qui concernent les configurations extrêmes des ressources de chaque régulation, à savoir dans le cas où tous les activateurs sont présents et tous les inhibiteurs absents, et dans le cas inverse. Ce premier modèle abstrait donc un grand nombre de modèles de Thomas (de l'ordre de $9 \cdot 10^{52}$) étant donné que la quasi-totalité des paramètres restent indéterminés.

Afin de construire le modèle (2), 14 sortes coopératives ont été ajoutées dans le but de modéliser les fonctions booléennes de plusieurs composants (consistant en des portes logiques de type ET et OU). Pour ce faire, les composants suivants ont été retenus du fait de leur importance dans la chaîne de réactions : CDK4, CDK6, CycD1, ER-α and c-MYC. En effet, d'après les expériences par *knockdown* menées par Sahin et al. (2009), empêcher ces composants de s'exprimer menait à une réduction significative de la production de pRB. On peut en conclure que ces composants sont impliqués dans les fonctions booléennes d'autres composants d'une façon à ce que le *knockdown* des premiers empêche la production des seconds (ce qui est typique des portes de type ET). Afin de reproduire ce comportement, les fonctions booléennes des successeurs de ces composants ont été modélisées sous la forme de sortes coopératives le cas échéant, c'est-à-dire celles de CDK4, CDK6, prB, p21, p27, IGF1R, MYC, CycD1 et CycE1. En théorie, 9 sortes coopératives auraient suffi, mais la factorisation des sortes coopératives décrite en page 33 a été utilisée afin de réduire la taille de celles-ci — en nombre de processus — lorsque c'était possible. Les sortes coopératives ainsi ajoutées ont permis d'inférer environ la moitié des paramètres ; cependant, le nombre de modèles de Thomas compatibles avec ce modèle

reste très élevé du fait des nombreux paramètres qui restent indéterminés. De plus, ce graphe des interactions inféré comporte un arc de moins que le graphe des interactions d'origine. Cela est dû au fait que la fonction booléenne de pRB intégrée sous la forme d'une sorte coopérative pouvait en fait être simplifiée d'une façon telle que le composant CDK2 n'y apparaissait plus. Aucun arc n'a alors été inféré entre CDK2 et pRB par notre méthode, car il ne participait pas à modifier la dynamique.

Enfin, le modèle (3) a été construit en utilisant toutes les fonctions booléennes fournies par Sahin et al. (2009). Ces fonctions prennent la forme de 22 sortes coopératives qui permettent de retrouver le comportement attendu du système. Toutes ces coopérations étant correctement définies dans le modèle, tous les paramètres peuvent être inférés et un seul modèle de Thomas est compatible avec la dynamique des Frappes de Processus canoniques utilisées. Nous notons de plus que ce modèle est le seul à **contenir des points fixes**, calculés à l'aide du théorème 3.2 en page 47, et qui comprennent notamment les deux états stables correspondant à une propagation totale du signal pour les deux cas initiaux, c'est-à-dire dans le cas où EGF est initialement actif et dans le cas où il ne l'est pas. Les deux autres modèles ne contiennent pas de point fixe car les coopérations manquantes créent des oscillations qui sont la conséquence du comportement non-déterministe du formalisme lorsqu'on en reste à un stade de précision incomplet.

6.1.2 Temps d'exécution sur quelques modèles de grande taille

L'implémentation de la méthode proposée à la section 5.2 en page 94 permet de traiter des modèles de Frappes de Processus canoniques de grande taille et obtenus à partir de données trouvées dans la littérature. Ont notamment été traités :

- les modèles du récepteur de croissance épidermique présentés à la section 6.1.1 en page 122 (Sahin et al., 2009), qui contiennent 20 composants et jusqu'à 22 sortes coopératives,
- un modèle de récepteur de lymphocyte T (Klamt, Saez-Rodriguez, Lindquist, Simeoni & Gilles, 2006), contenant 40 composants et 14 sortes coopératives.

Pour chacun de ces modèles, les inférences du graphe des interactions et des paramètres sont effectuées en moins d'une seconde sur un ordinateur de bureau standard[2]

D'autres modèles de plus grande taille portant sur les mêmes systèmes ont aussi été testés :

- un modèle du récepteur de lymphocyte T contenant 94 composants, qui sera par ailleurs présenté à la section suivante (Saez-Rodriguez, Simeoni, Lindquist, Hemenway, Bommhardt, Arndt, Haus, Weismantel, Gilles, Klamt & Schraven, 2007),
- un modèle du récepteur de croissance épidermique avec 104 composants (Samaga, Saez-Rodriguez, Alexopoulos, Sorger & Klamt, 2009).

Ces deux modèles ont été obtenus par Paulevé et al. (2012) à l'aide d'une traduction automatique depuis le format CellNetAnalyzer (Klamt, Saez-Rodriguez & Gilles, 2007) vers le Frappes de Processus canoniques.

La composition de ces modèles et les résultats de l'inférence du modèle de Thomas sont résumés dans la tableau 6.2. Nous notons notamment que, du fait de la très grande taille

[2]Pour tous les exemples de cette section, le logiciel a été lancé sur une machine comportant un processeur de 3,4 Ghz avec 8 Go de mémoire vivre.

des paramétrisations des modèles de 94 et 104 composants, l'inférence des paramètres n'a pas pu être réalisée. Ce nombre anormal de paramètres est dû à la présence d'un très grand nombre de régulateurs pour un même composant dans ces deux modèles, faisant croître la taille de la paramétrisation de façon exponentielle. Il est à noter que ces régulations sont principalement dues à des contraintes de modélisation propres à la représentation d'origine au format CellNetAnalyzer ; notre méthode devrait néanmoins être en mesure de traiter des modèles d'aussi grande taille à condition que chaque composant ne possède qu'un nombre limité de régulateurs. Cela indique de plus que de tels modèles seraient plus efficacement étudiés en tant que Frappes de Processus qu'en tant que modèles de Thomas.

Modèle	$\lvert\boldsymbol{\Gamma}\rvert$	$\lvert\boldsymbol{\Delta}\rvert$	**Inférence GI**	**Inférence K**	$\lvert\mathbf{K}\rvert$
RFCE	20	22	< 1 s	< 1 s	192
RLT	40	14	< 1 s	< 1 s	143
RLT	94	39	10 s	—	$2{,}1 \cdot 10^9$
RFCE	104	89	3 min 20 s	—	$4{,}2 \cdot 10^6$

Table 6.2 – Temps d'exécution et informations relatives aux modèles utilisés dans l'inférence du graphe des interactions et de la paramétrisation de quatre modèles de réseaux de régulation biologique. Dans la première colonne, RFCE signifie « Récepteur de facteur de croissance épidermique » et RLT signifie « Récepteur de lymphocyte T ». Les références dont sont tirés ces modèles sont données à la page précédente. La deuxième colonne donne le nombre de composants de chaque modèle et la troisième donne le nombre de sortes coopératives utilisées pour modéliser des coopérations entre composants. La quatrième (resp. cinquième) colonne donne le temps de calcul de l'inférence du graphe des interactions (resp. de la paramétrisation) pour chaque modèle, et la dernière colonne renseigne sur le nombre de paramètres que contient chaque modèle. Du fait d'un trop grand nombre de paramètres, l'inférence de la paramétrisation n'a pas pu être effectuée sur les modèles de 94 et 104 composants par manque de mémoire.

Implémentation

La traduction de Frappes de Processus canoniques en modèle de Thomas développée à la section 5.2 en page 94 a été implémentée[3] et intégrée à Pint. L'implémentation consiste principalement en trois programmes écrits en programmation par ensembles de réponses (ou *Answer Set Programming*), un paradigme de programmation logique permettant de se concentrer sur la modélisation du problème plutôt que sur sa résolution (Baral, 2003). Ce paradigme est notamment pris en charge par le *grounder-solver* Clingo (Gebser et al., 2014), qui a déjà montré ses capacités dans la résolution de problèmes de complexité NP. Les trois programmes de cette implémentation se chargent respectivement de réaliser l'inférence du graphe des interactions, des paramètres et de tous les modèles de Thomas compatibles. Des compléments écrits en OCaml assurent de plus la traduction du modèle vers le langage de programmation par ensembles de réponses puis la récupération des résultats du *solver*. Cette traduction peut être appelée par l'outil `ph2thomas`, par exemple avec la ligne de commande suivante :

[3] Cette implémentation a été réalisée dans le cadre d'un stage doctoral en tant qu'invité dans l'équipe de Katsumi Inoue (National Institute of Informatics, Tokyo, Japon).

```
ph2thomas -i ERBB_G1-S.ph
```

pour effectuer l'inférence des paramètres sur le modèle de Frappes de Processus canoniques du fichier `ERBB_G1-S.ph`. Il est possible de plus d'exporter le graphe des interactions inféré dans un fichier `ERBB_G1-S.dot` à l'aide de l'option `--dot ERBB_G1-S.dot` ou de lancer l'énumération des modèles de Thomas compatibles avec `--full-enumerate` (ou `--enumerate` pour se restreindre à des paramètres entiers).

6.2 Analyse statique du récepteur de lymphocyte T

Nous souhaitons démontrer dans cette section la puissance et l'adaptabilité de l'analyse statique par sous-approximation développée à la section 4.3. Nous l'appliquons pour cela à un réseau booléen asynchrone comportant 94 composants, ce qui peut être considéré comme un grand modèle, et modélisant le récepteur de lymphocyte T (*T-cell receptor*). Ce modèle avait auparavant été étudié par une méthode proche dans le cadre des Frappes de Processus standards (Paulevé et al., 2012), qui ne permettaient cependant d'étudier qu'une approximation du modèle d'origine. Nous utilisons ici une représentation en Frappes de Processus standards en assignant une priorité haute à toutes les actions mettant à jour les sortes coopératives ajoutées pour représenter les coopérations entre composants. Nous constatons que **notre méthode d'analyse statique reste efficace malgré la taille du modèle** ; par ailleurs, nous toutes les analyses effectuées s'avèrent être conclusives.

Nous nous intéressons au modèle de récepteur de lymphocyte T proposé par Saez-Rodriguez et al. (2007), qui se présente sous la forme d'un réseau booléen asynchrone comportant 94 composants. On y distingue notamment des composants d'entrée, c'est-à-dire qui ne sont régulés par aucun autre composant, et dont la valeur de départ est donc fixe, et des composants de sortie qui à l'inverse ne régulent aucun autre composant. Dans l'état initial, les composants d'entrée sont généralement indépendamment choisis actifs ou inactifs selon ce qu'on souhaite observer, et tous les autres composants le sont en fonction des conditions initiales habituelles observées expérimentalement — la majorité étant inactifs. Cela permet d'observer la propagation du signal d'entrée caractérisée par une (dés)activation successive des autres composants du modèle.

Le but de cette étude est de tester la possibilité d'activer chaque composant de sortie en fonction de l'état initial (activé ou non) de chaque composant d'entrée. Cela permet notamment de vérifier que le modèle est fonctionnel, autrement dit de vérifier que tous les composants de sortie peuvent être activés si toutes les entrées le sont — et qu'elles ne peuvent pas l'être si aucune entrée n'est active. De plus, tous les autres comportements intermédiaires peuvent être analysés afin de prévoir le comportement du système réel dans ces situations.

Le modèle comporte trois composants d'entrée (appelés CD4, CD28 et TCRlig) et quatre composants de sortie (SRE, AP1, NFkB et NFAT). Il a été traduit automatiquement en Frappes de Processus canoniques[4] depuis le format CellNetAnalyzer (Klamt et al., 2007). Les états initiaux sont construits en définissant tous les composants comme étant inactifs sauf pour certains d'entre eux (à savoir : CD45, CARD11, Bcl10, Malt1, Rac1r, Lckr, cCblr et IkB) et pour les composants d'entrée choisis comme initialement actifs. Pour chacun des 8 états initiaux ς ainsi obtenus, et pour chaque composant x de sortie, nous vérifions la propriété P^x_ς qui s'exprime de la manière suivante : « Depuis l'état ς,

[4] tous les fichiers sont disponibles à `http://maxime.folschette.name/underapprox-tcrsig94.zip`.

existe-t-il une succession d'actions permettant d'atteindre un état où x_1 est actif ? » En d'autre termes, nous testons l'atteignabilité de chaque processus x_1 indépendamment. Nous utilisons pour cela les deux analyses statiques suivantes :

- la sous-approximation développée à la section 4.3 en page 80, et notamment le théorème 4.3, qui permet de répondre « Oui » ou « Non-conclusif » à une question d'atteignabilité locale,
- la sur-approximation proposée par Paulevé et al. (2012), appliquée au modèle fusionné d'après la définition 3.3 en page 46, qui permet de répondre « Non » ou « Non-conclusif ».

En d'autres termes, la première méthode, issue du travail présenté dans cette thèse, teste l'atteignabilité afin de répondre positivement ; son utilisation se justifie par le fait qu'elle est propre aux Frappes de Processus canoniques, qui sont utilisées pour représenter le modèle étudié. Dans les cas où cette méthode ne permet pas de répondre (c'est-à-dire lorsqu'elle répond « Non-conclusif »), la seconde méthode est utilisée en complément afin de tester l'atteignabilité et de répondre négativement. Dans le cas où les deux méthodes répondraient « Non-conclusif », aucune conclusion ne pourrait être formulée pour la question d'atteignabilité ; cependant, ce cas ne s'est pas présenté pour le modèle étudié ici.

Il s'avère finalement que cette méthode est conclusive pour tous les cas testés ; autrement dit, les deux méthodes ne répondent jamais « Non-conclusif » pour une même question d'atteignabilité. Sur les 32 cas de figure testés, nous avons notamment pu répondre « Oui » dans 12 cas grâce au théorème 4.3 en page 85. Lorsque tous les composants d'entrée sont inactifs, aucun composant de sortie ne peut être activé ; à l'inverse, lorsque tous les composants d'entrée sont actifs, SRE, AP1 et NFAT peuvent être activés mais pas NFkB, ce qui permet de corriger le modèle de façon à la rendre actif en ajoutant une action manquante. Les temps de calcul sont de l'ordre de quelques centièmes de seconde sur un ordinateur de bureau classique[5], ce qui permet d'effectuer un grand nombre de tests sur un même modèle. À titre de comparaison, les mêmes analyses ont été effectuées sur une traduction en réseau de Petri du modèle étudié, à l'aide du *model checker* symbolique libDDD (Thierry-Mieg, Poitrenaud, Hamez & Kordon, 2009), connu pour ses bonnes performances. Cependant, du fait de la taille du modèle, le programme s'arrête sur un dépassement de mémoire pour chacun des cas testés.

Pour terminer, nous testons en complément l'atteignabilité simultanée de l'activation des quatre composants de sortie (SRE, AP1, NFkB et NFAT). Ce résultat est utile car les propriétés précédemment étudiées consistaient uniquement en des atteignabilités indépendantes, et il demeure toujours la possibilité que l'une des atteignabilité n'en entrave une autre. Pour ce résultat supplémentaire, nous modifions le modèle comme expliqué à la section 4.3.3 en page 88, qui consiste à ajouter les éléments suivants :

- une sorte coopérative τ entre les quatre composants considérés,
- toutes les actions primaires nécessaires à la mise à jour de τ,
- une sorte ρ comprenant deux processus (ρ_0 et ρ_1),
- une action secondaire $\tau_{1111} \to \rho_0 \Rsh \rho_1$.

Nous pouvons alors calculer l'atteignabilité de ρ_1, qui permettra de déterminer si les quatre composants peuvent être simultanément actifs depuis l'état initial.

[5]Testé sur une machine comportant un processeur de 2,4 Ghz avec 2 Go de mémoire vivre.

L'implémentation répond alors « Oui », ce qui permet d'affirmer que les quatre composants peuvent être activés simultanément, ce qui permet de conclure définitivement que le modèle est fonctionnel. La réponse a été obtenue quelques centièmes de seconde, en ne considérant qu'un sous-ensemble des nœuds solution, selon la méthode expliquée par la remarque en page 85.

Nous avons démontré avec cet exemple la puissance de l'analyse statique que nous proposons à la section 4.3 en page 80, applicable aux réseaux booléens asynchrones. Bien que la dynamique soit approximée, tous les cas étudiés dans cet exemple restent conclusifs, ce qui permet d'étudier la dynamique d'un grand modèle de 94 composants, chose impossible avec les outils de *model checking* classiques.

Implémentation

Nous avons implémenté l'analyse statique développée à la section 4.3 en page 80 et l'avons intégré à la bibliothèque Pintsur la base de celle, déjà existante, portant sur les Frappes de Processus standards. L'outil en question s'appelle `ph-reach`, et il est nécessaire de spécifier l'option `--coop-priority` pour que les actions de mise à jour des sortes coopératives soient priorisées, et que le modèle soit interprété comme des Frappes de Processus canoniques. Voici un exemple de ligne de commande :

```
ph-reach --coop-priority -i tcrsig94.ph
    --initial-state "cd4 1, cd45 1, cd28 1, tcrlig 1" ap1 1
```

Cette ligne de commande utilise le modèle du fichier `tcrsig94.ph` ; elle définit l'état initial du modèle comme étant $s \Cap \{\mathrm{CD4}_1, \mathrm{CD45}_1, \mathrm{CD28}_1, \mathrm{TCRlig}_1\}$ (où s est défini préalablement dans le fichier `tcrsig94.ph` et $\Cap$ représente l'opérateur de recouvrement défini à la définition 2.11 en page 28) et calcule l'atteignabilité de $\mathrm{AP1}_1$ à partir de cet état. Après exécution, l'outil retourne `True`, ce qui signifie que l'atteignabilité testée à l'aide de la méthode de la section 4.3 en page 80 est possible. Les autres réponses sont `False`, calculé d'après les travaux précédents de Paulevé et al. (2012), et `Inconc`, qui indique qu'une des deux approximations n'a permis de conclure.

6.3 Bilan

Les applications présentées dans ce chapitre illustrent l'utilisation des méthodes développées dans cette thèse. Les différents résultats montrent que ces méthodes peuvent être appliquées à des modèles de grande taille et fournissent donc des résultats intéressants.

La traduction vers le modèle de Thomas permet de rendre compte des abstractions permises par les Frappes de Processus. En effet, un seul modèle de Frappes de Processus permet de représenter les dynamiques « superposées » d'un ensemble de modèles de Thomas. Cet ensemble peut être très grand, comme c'est le cas pour le modèle (1) étudié à la section 6.1 qui est compatible avec un très grand nombre de modèles. La dynamique d'une telle abstraction serait donc mieux étudiée à l'aide d'un modèle de Frappes de Processus canoniques, qu'avec un grand nombre de modèles de Thomas. Cependant, s'il est possible de raffiner un tel modèle suffisamment, la traduction s'avère intéressante pour pouvoir l'étudier à l'aide de nouvelles approches.

L'application à la section 6.2 de l'analyse statique développée au chapitre 4 montre que cette méthode est très efficace pour l'analyse d'un modèle comportant une centaine

de composants. Cependant, au vu des temps de calcul (n'excédant pas le dixième de seconde), il semble tout à fait envisageable de l'appliquer à des modèles de taille supérieure, allant jusqu'à plusieurs milliers de composants. Nous notons par ailleurs que l'analyse a toujours été conclusive, malgré l'approximation de la dynamique effectuée. Cette conclusivité s'explique notamment par la forme du modèle considéré, qui comporte des interactions « simples » entre composants : des activations et des inhibitions, parfois sous la forme de coopérations. Les approximations développées par Paulevé et al. (2012) puis par Folschette et al. (2013) prennent parti de cette forme particulière, propre aux réseaux de régulation biologique, et sont donc particulièrement adaptées à ce type de modèles.

Les applications proposées dans ce chapitre montrent en quoi les résultats que nous proposons ouvrent la voie à de nombreuses possibilités d'analyse. Il devient en effet possible d'analyser de grands modèles de réseaux de régulation biologique, ce qui permet de nombreuses applications. Nous en proposons quelques-unes :

- Le raffinement des coopérations d'un modèle peut être traité en ajoutant des sortes coopératives à des endroits clefs et en testant certaines propriétés dynamiques (présence de points fixes ou atteignabilité des composants de sortie), dans le prolongement de l'application proposée à la section 6.1.
- L'inférence des actions manquantes d'un modèle à partir de données expérimentales nouvelles : chaque comportement biologique peut être testé sur le modèle, et en cas d'échec, le graphe de causalité locale peut être analysé pour en déduire les comportements manquants.
- Grâce aux techniques puissantes d'analyse de la dynamique, il devient potentiellement possible d'énumérer un grand nombre de modèles et de les tester indépendamment.

Chapitre 7

Conclusion et perspectives

Nous revenons dans ce chapitre sur les apports de la présente thèse, en insistant notamment sur la façon dont nous avons répondu aux deux problématiques principales proposées initialement : renforcer la puissance de la modélisation d'une part, en permettant notamment l'intégration de données temporelles et de synchronisation, et les capacités d'analyse d'autre part, particulièrement en ce qui concerne les modèles de grande taille. Plusieurs nouvelles perspectives scientifiques sont par ailleurs évoquées.

L'étude des réseaux de régulation biologique peut s'effectuer sous de nombreuses formes, allant des systèmes d'équations différentielles à la programmation logique, en passant par des modèles probabilistes. L'utilisation de modèles discrets permet une abstraction puissante de la complexité inhérente de ces systèmes, en conservant certaines propriétés intéressantes. On peut considérer que l'utilisation de tels modèles a été initiée par Stuart A. Kauffman (1969), plus tard suivi par René Thomas (1973) qui y ajoute une dynamique asynchrone et par El Houssine Snoussi (1989) qui propose la notion de paramètres discrets pour définir totalement la dynamique. Ces modèles discrets ont été abondamment étudiés, comme nous l'avons vu au chapitre 2 : beaucoup de résultats permettent d'optimiser l'étude de leur dynamique, mais aussi de déduire la présence de certains comportements simplement à l'aide de la structure du modèle.

Cependant, la simplicité apparente de leur définition masque deux écueils importants. Tout d'abord, ces modèles possèdent une dynamique souvent très complexe, l'espace des états ayant en effet une taille exponentielle dans la taille du modèle. Les méthodes de compression ou de réduction des modèles ne permettent pas toujours de restreindre suffisamment cet espace des états pour en permettre une étude exhaustive. L'analyse dynamique des modèles de régulation reste donc souvent cantonnée à des modèles de petite ou moyenne taille — rarement plus d'une vingtaine de composants. De plus, certains paramètres peuvent être difficiles, voire impossibles à intégrer dans le modèle. En effet, le modèle de Thomas ne permet pas de faire figurer des notions de précédence, de préemption ou de synchronisme entre les différentes interactions. De même, son étude requiert d'avoir correctement défini tous les paramètres discrets propres au système étudié, paramètres qui

sont parfois inconnus à priori, alors que les comportements attendus le sont (même partiellement). Pour finir, il peut s'avérer difficile de raffiner un modèle lorsqu'il n'est pas possible d'étudier l'impact de chaque modification à cause des difficultés inhérentes à l'analyse des grands modèles.

Les Frappes de Processus, introduites par Paulevé, Magnin & Roux (2011*a*), offrent un outil alternatif puissant de représentation et d'analyse des réseaux de régulation biologique. Le formalisme proposé, totalement asynchrone, est basé sur une représentation atomique des interactions entre composants, où le changement d'état d'un composant ne peut être déclenché que par l'état d'au plus un autre composant. Cette forme particulière a notamment permis le développement de méthodes efficaces d'analyse de la dynamique, qui permettent notamment d'en réduire la complexité (Paulevé et al., 2012). Cependant, les Frappes de Processus ne permettent pas de représenter fidèlement un réseau de régulation biologique sous la forme d'un réseau discret asynchrone. De plus, certaines contraintes de modélisation ne peuvent pas toujours être introduites efficacement dans ces modèles standards de Frappes de Processus.

Le travail proposé dans cette thèse avait pour ambition de contribuer à répondre à cette double problématique de modélisation et d'analyse des grands modèles de réseaux de régulation biologique, en partant du formalisme Frappes de Processus précédemment proposé, et des différents travaux correspondants. Nous proposons un bref rappel des résultats obtenus sur ces deux points à la section 7.1 et nous ouvrons un certain nombre de perspectives dans la lignée de nos travaux à la section 7.2.

7.1 Retour sur les résultats de cette thèse

Nous mettons en perspective dans la suite l'apport de nos travaux aux deux questions que sont la modélisation et l'analyse d'un système.

Modélisation

La représentation d'un système réel (ou fictif) sous forme de modèle est fondamentale et doit être adaptée aux besoins de l'analyse.

Nous avons proposé au chapitre 3 plusieurs représentations alternatives permettant d'enrichir la modélisation d'un système en Frappes de Processus.

À la section 3.1, nous avons défini des **classes de priorités** sur les actions d'un modèle de Frappes de Processus. Ces classes de priorités permettent de spécifier des contraintes de préemption globales entre plusieurs ensembles d'actions — chaque action ne pouvant être jouée que si aucune action plus prioritaire ne l'est. Nous avons montré comment ces classes de priorités permettent de représenter de façon discrète des **contraintes temporelles** (éventuellement continues). Pour cela, nous avons pris comme exemple le cas de modèles de Frappes de Processus intégrant des paramètres stochastiques pour chaque action, paramètres qui peuvent se traduire en intervalles de tirs, et nous avons montré comment modéliser une dynamique semblable grâce à des classes de priorités.

L'introduction d'**arcs neutralisants** à la section 3.2 a en revanche permis de définir des contraintes de préemption locales entre les actions. Leur plus grande granularité permet d'intégrer plus finement certaines informations relatives entre les actions, comme des contraintes de tir ou des durées de réaction. Cette granularité est notamment nécessaire pour éviter des relations de préemption non désirées qui peuvent bloquer des parties du

modèle, par exemple en abstrayant certaines phénomènes d'accumulation, provoquant des **comportements Zénon**.

Enfin, les **actions plurielles** définies à la section 3.3 introduisent de la synchronicité entre les actions. Celle-ci est nécessaire pour modéliser certains phénomènes comme des réactions biochimiques, qui nécessitent de représenter la formation de plusieurs produits ou la consommation de plusieurs réactifs, voire les deux simultanément. Ce formalisme offre donc à fortiori la possibilité de représenter du **synchronisme** au niveau des pré-conditions de tir des actions, et par conséquent permet de compresser les modèles à l'aide d'une notion naturelle de coopération.

Nous avons de plus délimité au chapitre 4 une sous-classe des Frappes de Processus avec classes de priorités, dites **Frappes de Processus canoniques**, dont les modèles ne comportent que deux classes de priorités et possèdent une forme particulière pour les actions très prioritaires. Cette classe de modèles présente un intérêt particulier pour la représentation, car elle **corrige les problèmes de décalage temporel** présents dans les coopérations modélisées par des Frappes de Processus standards. Nous avons de plus mis en évidence le fait qu'elle est suffisante pour représenter tous les autres formalismes de Frappes de Processus alternatifs rappelés ci-dessus ; nous avons démontré par ailleurs grâce à cela que tous ces formalismes possèdent une **expressivité équivalente**, et nous utilisons les Frappes de Processus canoniques pour proposer les traductions adéquates.

Enfin, le chapitre 5 a été l'occasion de comparer l'expressivité des Frappes de Processus avec d'autres formalismes courants.

Nous avons notamment démontré à la section 5.1 que les Frappes de Processus canoniques permettent de représenter fidèlement et naturellement les **réseaux discrets asynchrones**. Cette équivalence est particulièrement utile pour pouvoir étudier ce type de modèles efficacement à l'aide des outils détaillés plus bas.

Nous nous sommes aussi penché sur le cas du **modèle de Thomas** à la section 5.2, qui présente un intérêt supplémentaire du fait de sa dynamique unitaire et de ses paramètres discrets. Nous avons donné les conditions sous lesquelles les Frappes de Processus canoniques peuvent être traduites en modèle de Thomas de façon plus ou moins complète, ce qui se manifeste par des arcs non-signés ou des paramètres impossibles à inférer. Cependant, en cas de réponse partielle, un ensemble de modèles compatibles peut être énuméré, et le modèle d'origine peut être itérativement raffiné afin d'obtenir le modèle de Thomas attendu.

Ces différents formalismes de Frappes de Processus peuvent directement bénéficier aux biologistes désirant vérifier le bon fonctionnement d'un réseau discret asynchrone ou d'un modèle de Thomas, ou effectuer des prédictions sur ces modèles. Ils peuvent naturellement tout aussi bien être utilisés à d'autres fins, comme l'inférence ou la correction de modèles, à l'aide des différentes équivalences proposées. Cependant, il est aussi envisageable pour des informaticiens ou bioinformaticiens d'en faire usage dans un cadre plus large, à la condition de pouvoir représenter le système informatique souhaité sous la forme de Frappes de Processus.

Analyse

Un formalisme adapté à la représentation d'une certaine classe de systèmes présente cependant un intérêt limité s'il n'est pas accompagné de méthodes d'analyse efficaces.

C'est pourquoi, en sus de la définition des Frappes de Processus canoniques, nous avons proposé au chapitre 4 une méthode efficace d'**analyse statique pour l'atteignabilité**, qui peut cependant aussi être appliquée aux autres formalismes de Frappes de Processus, moyennant les traductions adéquates. Elle s'inspire des résultats de Paulevé, Magnin & Roux (2012) qui avaient précédemment proposé une telle analyse sur les Frappes de Processus standards. De la même façon, notre analyse s'appuie sur une sous-approximation de la dynamique du modèle étudié, qui exploite la forme particulière des actions et de la définition des Frappes de Processus canoniques. L'apport de notre travail est donc, grâce aux traductions proposées au chapitre 5, de **pouvoir répondre de façon formelle à des questions d'atteignabilité locale sur des modèles de réseaux discrets asynchrones**, ce qui était impossible en utilisant les Frappes de Processus standards. De plus, elle reste efficace du fait de sa complexité polynomiale dans la taille du modèle étudié, évitant ainsi l'explosion combinatoire inhérente au calcul exhaustif de la dynamique. À l'instar de la méthode d'origine, elle permet de vérifier des questions d'atteignabilités individuelles successives ; les question d'**atteignabilités simultanées d'états locaux** peuvent cependant aussi être traitées, ce qui est propre à notre approche.

Enfin, le chapitre 6 nous a permis d'illustrer l'étendue des différents résultats proposés dans cette thèse par quelques applications à des modèles biologiques.

L'étude de comportements peut être abordée par des formalismes différents. Les différents modèles de Frappes de Processus traduits en modèle de Thomas à la section 6.1 ont illustré cela : chaque modèle était issu d'un **raffinement** différent d'un même système, et la traduction en modèle de Thomas a permis de déterminer les parties de modèle dont le comportement est déterministe. De plus, l'analyse efficace de la dynamique sur les Frappes de Processus peut s'avérer utile pour effectuer des raffinements successifs d'un même modèle avant de le traduire vers le formalisme souhaité. La conservation de la dynamique pour toutes les traductions présentées permet en effet de généraliser les résultats obtenus sur un modèle à tous les formalismes dans lesquels il peut être traduit.

Nous avons appliqué à la section 6.2 la méthode d'analyse des Frappes de Processus canoniques résumée ci-dessus à un réseau discret asynchrone de 94 composants et pouvant donc être considéré comme de grande taille. Cette application a permis de répondre avec succès à plusieurs questions d'atteignabilité avec **des temps de calcul de l'ordre de quelques centièmes de seconde**, ce qui à notre connaissance est inégalé.

Les possibilités offertes par l'utilisation conjointe des traductions présentées dans cette thèse avec les méthodes d'analyse développées rendent accessible l'étude de modèles de grande taille, voire de très grande taille. Ces capacités d'analyse devraient notamment être suffisantes pour pouvoir analyser en un temps raisonnable des propriétés dynamiques sur des bases de données métaboliques comme PID (Schaefer, Anthony, Krupa, Buchoff, Day, Hannay & Buetow, 2009) ou des regroupements de telles bases de données comme hiPathDB (Yu, Seo, Rho, Jang, Park, Kim & Lee, 2012).

7.2 Perspectives de travail

Les travaux présentés au cours de cette thèse ouvrent de nombreuses pistes de travail permettant d'étendre et d'exploiter les résultats proposés.

Raffinement de l'analyse statique

L'analyse d'atteignabilité présentée au chapitre 4 repose sur une approximation de la dynamique ; cette approximation fait chuter la complexité de l'analyse, au risque cependant de ne pas pouvoir conclure dans certains cas. Si le besoin s'en fait ressentir sur certains modèles, il pourrait naturellement être intéressant de **raffiner cette analyse statique** tant au niveau de la sous-approximation que nous proposons qu'au niveau de la sur-approximation définie par Paulevé et al. (2012), afin de pouvoir conclure dans un plus grand nombre de cas. Un tel raffinement serait basé soit sur la construction du graphe de causalité locale, soit sur les conditions nécessaires et suffisantes qui l'exploitent. Des **méthodes dérivées** peuvent aussi être envisagées, permettant par exemple de guider une analyse plus exhaustive, comme l'analyse des scénarios permis par le graphe de causalité locale.

De même, les méthodes d'analyse statique en question pourraient être adaptées à d'autres formalismes que les Frappes de Processus canoniques. Leur application directe aux Frappes de Processus avec classes de priorités ou aux Frappes de Processus avec actions plurielles, par exemple, permettrait d'éviter la traduction coûteuse d'un modèle vers les Frappes de Processus canoniques.

Par ailleurs, l'analyse d'atteignabilité ne permet actuellement de vérifier que des propriétés qui s'expriment en logique CTL sous la forme $\mathbf{EF}(P)$; il pourrait être intéressant de l'enrichir avec de **nouveaux types de propriétés** comme $\mathbf{AF}(P)$, assurant par exemple qu'un état local est toujours atteignable, ou encore $(P\ \mathbf{EU}\ Q)$, qui permettrait d'observer des commutations, c'est-à-dire des bifurcations de la dynamique impossibles à inverser. Par exemple, la formule suivante décrit un tel comportement, en exprimant le fait qu'il existe un chemin tel que les deux propriétés P et Q sont initialement atteignables, mais qu'un phénomène de commutation force l'atteignabilité de Q tout en empêchant celle de P :

$$(\mathbf{EF}(P) \wedge \mathbf{EF}(Q))\ \mathbf{EU}\ (\neg\mathbf{EF}(P) \wedge \mathbf{AF}(Q))$$

L'utilisation de logiques intégrant des **compteurs**, ou de **logiques non-régulières**, augmenterait aussi les capacités d'analyse.

Extensions de la modélisation et de l'analyse

Dans un cadre plus général, au niveau de la modélisation, la représentation de certains systèmes biologiques peut nécessiter de nouveaux ajouts au niveau du formalisme, comme des **classes de priorités dynamiques** ou des **actions gardées**. À l'instar des réseaux d'automates, l'ajout de telles notions crée une forte complexité, mais l'approche par interprétation abstraite pourrait à nouveau être plus efficace qu'une recherche naïvement exhaustive.

En ce qui concerne l'analyse, de nombreux autres résultats peuvent être recherchés sur un modèle de Frappes de Processus, comme la **prévision d'oscillations**, la recherche de **domaines pièges** et d'**attracteurs**, etc. Il est difficile de prévoir quels types d'analyse ces résultats peuvent requérir. Le cas le plus souhaitable est une simple observation de la structure du modèle (à l'instar des conjectures de Thomas) impliquant une complexité polynomiale, voire linéaire. De telles observations peuvent porter sur la forme et l'agencement des actions, sur la recherche d'éléments comme les $(n-k)$-cliques dans le graphe sans-frappe, etc. Le cas le plus coûteux en temps de calcul, en revanche, serait la nécessité d'effectuer une **analyse exhaustive** de la dynamique pour parvenir à un résultat. De telles analyses peuvent cependant bénéficier d'avancées notables comme l'utilisation d'un paradigme de programmation logique telle que la programmation par ensembles de réponse

ou *Answer Set Programming* (voir p. ex. Ben Abdallah, 2014). Il est de plus possible de trouver des résultats intermédiaires permettant de guider la recherche et de trouver plus rapidement une solution (comme l'utilisation du graphe de causalité locale évoquée plus haut).

Autres pistes d'application des capacités d'analyse

L'**inférence et la correction** de modèles pour les systèmes biologiques suscitent un engouement justifié, alors que les données d'expression deviennent toujours plus accessibles. L'utilisation de **données de puce à ADN** donne en effet accès une quantité inégalée de données d'expression et offre la possibilité de corriger des modèles existants. L'automatisation de ce procédé présente alors un intérêt certain, notamment pour les systèmes complexes, qui comportent beaucoup de mesures et qui sont représentés par des modèles de grande taille (voir p. ex. Guziolowski, Videla, Eduati, Thiele, Cokelaer, Siegel & Saez-Rodriguez, 2013). Les Frappes de Processus pourraient être avantageusement utilisées pour ce type de problème, étant données leur définition atomique et les analyses efficaces de la dynamique dont elles font l'objet.

Dans le même ordre d'idées, la question de la **résilience d'un système biologique**, c'est-à-dire sa capacité et sa propension à revenir vers un état acceptable après une perturbation, peut aussi être soulevée et abordée de plusieurs manières grâce aux Frappes de Processus. Tout d'abord, l'ajout de **comportements incontrôlables** (Schwind, Magnin & Inoue, 2013) poserait de nouvelles questions : le modèle court-il le risque d'être forcé à rentrer dans un état non acceptable ? Est-il capable de revenir à tout moment dans un état acceptable ? Et à quel prix ? Répondre à ces questions pourra nécessiter d'élargir le cadre des propriétés vérifiables de manière efficace. D'un autre point de vue, il serait intéressant d'observer les conséquences sur la dynamique de **perturbations ponctuelles aléatoires**, comme l'ajout ou la suppression d'actions dans un modèle. Cette approche pourrait bénéficier d'autres moyens détournés comme le calcul des *cut-sets* afin de détecter des parties sensibles des modèles, en révélant les parties du modèle les plus sensibles aux perturbations.

Bibliographie

Aracena, J. (2008), 'Maximum number of fixed points in regulatory boolean networks', *Bulletin of Mathematical Biology* **70**(5), 1398–1409.

Baldan, P., Busi, N., Corradini, A. & Michele Pinna, G. (2000), Functional concurrent semantics for petri nets with read and inhibitor arcs, *in* C. Palamidessi, ed., 'CONCUR 2000 — Concurrency Theory', Vol. 1877 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 442–457.

Baral, C. (2003), *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press.

Ben Abdallah, E. (2014), Exhaustive search of dynamical properties in process hitting using answer set programming, *in* 'Proceedings of the 11th Summer School on Modelling and Verification of Parallel Processes (MOVEP'14)', Nantes, pp. 82–88.

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., Richard, A. & Guespin, J. (2004), 'Application of formal methods to biological regulatory networks : extending thomas' asynchronous logical approach with temporal logic', *Journal of Theoretical Biology* **229**(3), 339–347.

Brand, D. & Zafiropulo, P. (1983), 'On communicating finite-state machines', *Journal of the ACM* **30**(2), 323–342.

Bryant, R. E. (1986), 'Graph-based algorithms for boolean function manipulation', *Computers, IEEE Transactions on* **C-35**(8), 677–691.

Chaouiya, C. (2007), 'Petri net modelling of biological networks', *Briefings in Bioinformatics* **8**(4), 210–219.

Clarke, E. M. & Emerson, E. A. (1982), Design and synthesis of synchronization skeletons using branching time temporal logic, *in* D. Kozen, ed., 'Logics of Programs', Vol. 131 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 52–71.

Comet, J.-P., Noual, M., Richard, A., Aracena, J., Calzone, L., Demongeot, J., Kaufman, M., Naldi, A., Snoussi, E. & Thieffry, D. (2013), 'On circuit functionality in boolean networks', *Bulletin of Mathematical Biology* **75**(6), 906–919.

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.

Fages, F., Soliman, S. & Chabrier-Rivier, N. (2004), 'Modelling and querying interaction networks in the biochemical abstract machine biocham', *Journal of Biological Physics and Chemistry* **4**, 64–73.

Folschette, M., Paulevé, L., Inoue, K., Magnin, M. & Roux, O. (2012*a*), Abducing biological regulatory networks from process hitting models, *in* 'Proceedings of the ECML-PKDD 2012 Workshop on Learning and Discovery in Symbolic Systems Biology (LDSSB'12)', Bristol, UK.

Folschette, M., Paulevé, L., Inoue, K., Magnin, M. & Roux, O. (2012*b*), Concretizing the process hitting into biological regulatory networks, *in* D. Gilbert & M. Heiner, eds, 'Computational Methods in Systems Biology X', Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 166–186.

Folschette, M., Paulevé, L., Magnin, M. & Roux, O. (2013), 'Under-approximation of reachability in multivalued asynchronous networks', *Electronic Notes in Theoretical Computer Science* **299**, 33–51. 4th International Workshop on Interactions between Computer Science and Biology (CS2Bio'13).

François, P., Hakim, V. & Siggia, E. D. (2007), 'Deriving structure from evolution : metazoan segmentation', *Molecular Systems Biology* **3**(1).

Gallet, E., Manceny, M., Le Gall, P. & Ballarini, P. (2014), Adapting LTL model checking for inferring biological parameters, *in* 'Actes des 13$^{\text{èmes}}$ journées sur les Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL 2014)', pp. 46–60.

Gebser, M., Kaminski, R., Kaufmann, B. & Schaub, T. (2014), '*Clingo* = ASP + control : Preliminary report'. Theory and Practice of Logic Programming, Online Supplement.

Gebser, M. & Schaub, T. (2013), 'Answer set solving in practice'. Tutorial at IJCAI'13.

Guziolowski, C., Videla, S., Eduati, F., Thiele, S., Cokelaer, T., Siegel, A. & Saez-Rodriguez, J. (2013), 'Exhaustively characterizing feasible logic models of a signaling network using answer set programming', *Bioinformatics* .

Hack, M. (1976), Petri net language, Technical report, Cambridge, MA, USA.

John, M., Lhoussaine, C., Niehren, J. & Uhrmacher, A. (2010), The attributed pi-calculus with priorities, *in* C. Priami, R. Breitling, D. Gilbert, M. Heiner & A. M. Uhrmacher, eds, 'Transactions on Computational Systems Biology XII', Vol. 5945 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 13–76.

Kauffman, S. A. (1969), 'Metabolic stability and epigenesis in randomly constructed genetic nets', *Journal of Theoretical Biology* **22**(3), 437–467.

Klamt, S., Saez-Rodriguez, J. & Gilles, E. D. (2007), 'Structural and functional analysis of cellular networks with cellnetanalyzer', *BMC Systems Biology* **1**(1).

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.

Marsan, M. A., Balbo, G., Chiola, G. & Conte, G. (1987), Generalized stochastic petri nets revisitied : Random switches and priorities, *in* 'The Proceedings of the Second International Workshop on Petri Nets and Performance Models', PNPM '87, IEEE Computer Society, pp. 44–53.

Milner, R. (1989), *Communication and Concurrency*, Prentice-Hall, Inc.

Naldi, A., Remy, E., Thieffry, D. & Chaouiya, C. (2009), A reduction of logical regulatory graphs preserving essential dynamical properties, *in* 'Computational Methods in Systems Biology', Vol. 5688 of *LNCS*, Springer, pp. 266–280.

Naldi, A., Thieffry, D. & Chaouiya, C. (2007), Decision diagrams for the representation and analysis of logical models of genetic networks, *in* 'Computational Methods in Systems Biology', Springer, pp. 233–247.

Paulevé, L. (2011), Modélisation, Simulation et Vérification des Grands Réseaux de Régulation Biologique, PhD thesis, École Centrale de Nantes.

Paulevé, L., Andrieux, G. & Koeppl, H. (2013), Under-approximating cut sets for reachability in large scale automata networks, *in* N. Sharygina & H. Veith, eds, 'Computer Aided Verification', Vol. 8044 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 69–84.

Paulevé, L., Chancellor, C., Folschette, M., Magnin, M. & Roux, O. (2014), *Analyzing Large Network Dynamics with Process Hitting*, ISTE Wiley. en cours d'impression.

Paulevé, L., Magnin, M. & Roux, O. (2011*a*), Refining dynamics of gene regulatory networks in a stochastic π-calculus framework, *in* 'Transactions on Computational Systems Biology XIII', Springer, pp. 171–191.

Paulevé, L., Magnin, M. & Roux, O. (2011*b*), 'Tuning temporal features within the stochastic π-calculus', *IEEE Transactions on Software Engineering* **37**(6), 858–871.

Paulevé, L., Magnin, M. & Roux, O. (2012), 'Static analysis of biological regulatory networks dynamics using abstract interpretation', *Mathematical Structures in Computer Science* **22**(04), 651–685.

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. & Richard, A. (2011), 'Static analysis of boolean networks based on interaction graphs : a survey', *Electronic Notes in Theoretical Computer Science* **284**, 93–104. Proceedings of The Second International Workshop on Static Analysis and Systems Biology (SASB 2011).

Peterson, J. L. (1977), 'Petri nets', *ACM Computing Surveys* **9**(3), 223–252.

Petri, C. A. (1962), Kommunikation mit automaten, PhD thesis, Universität Bonn.

Pnueli, A. (1977), The temporal logic of programs, *in* 'Foundations of Computer Science, 1977., 18th Annual Symposium on', pp. 46–57.

Priami, C. (1995), 'Stochastic π-calculus', *The Computer Journal* **38**(7), 578–589.

Remy, E., 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, PhD thesis, 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*, Springer Netherlands, chapter Formal Methods for Modeling Biological Regulatory Networks, pp. 83–122.

Roux, O. H. & Lime, D. (2004), Time petri nets with inhibitor hyperarcs. formal semantics and state space computation, *in* J. Cortadella & W. Reisig, eds, 'Applications and Theory of Petri Nets 2004', Vol. 3099 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 371–390.

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 Computational Biology* **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 Computational Biology* **5**(8), e1000438.

Schaefer, C. F., Anthony, K., Krupa, S., Buchoff, J., Day, M., Hannay, T. & Buetow, K. H. (2009), 'PID : the Pathway Interaction Database', *Nucleic Acids Research* **37**(suppl 1), D674–D679.

Schwind, N., Magnin, M. & Inoue, K. (2013), Resilience of event-driven dynamic systems, *in* 'the 27th Annual Conference on Japanese Society of Artificial Intelligence (JSAI 2013)'.

Snoussi, E. H. (1989), 'Qualitative dynamics of piecewise-linear differential equations : a discrete mapping approach', *Dynamics and Stability of Systems* **4**(3-4), 565–583.

Srinivasan, A., Ham, T., Malik, S. & Brayton, R. K. (1990), Algorithms for discrete function manipulation, *in* 'International Conference on Computer-Aided Design (ICCAD-90), Digest of Technical Papers', pp. 92–95.

Thierry-Mieg, Y., Poitrenaud, D., Hamez, A. & Kordon, F. (2009), Hierarchical set decision diagrams and regular models, *in* S. Kowalewski & A. Philippou, eds, 'Tools and Algorithms for the Construction and Analysis of Systems', Vol. 5505 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 1–15.

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* J. Della Dora, J. Demongeot & B. Lacolle, eds, 'Numerical Methods in the Study of Critical Phenomena', Vol. 9 of *Springer Series in Synergetics*, Springer Berlin Heidelberg, pp. 180–193.

Tyson, J. J. & Othmer, H. G. (1978), 'The dynamics of feedback control circuits in biochemical pathways', *Progress in Theorrtical Biology* **5**.

Yu, N., Seo, J., Rho, K., Jang, Y., Park, J., Kim, W. K. & Lee, S. (2012), 'hiPathDB : a human-integrated pathway database with facile visualization', *Nucleic Acids Research* **40**(D1), D797–D802.

Zhang, J., Johansson, K. H., Lygeros, J. & Sastry, S. (2001), 'Zeno hybrid systems', *International Journal of Robust and Nonlinear Control* **11**(5), 435–451.

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

Centrale Nantes

Thèse de Doctorat

Maxime FOLSCHETTE

Modélisation algébrique de la dynamique multi-échelles des réseaux de régulation biologique

Algebraic Modeling of the Dynamics of Multi-scale Biological Regulatory Networks

Résumé

La représentation et l'analyse des grands réseaux de régulation biologique sont les deux défis majeurs dans la compréhension des mécanismes du vivant. Le travail exposé dans cette thèse se concentre sur les modèles discrets, souvent représentés sous la forme de graphes et d'ensembles de paramètres. Il s'inspire notamment d'un formalisme préalablement développé, appelé Frappes de Processus, qui repose sur une représentation atomique d'un ensemble de composants et de leur dynamique. Nous proposons dans cette thèse plusieurs représentations alternatives à ce formalisme, qui possèdent une plus grande expressivité. Ces représentations sont adaptées à l'intégration de contraintes discrètes dans les modèles provenant de durées relatives ou de relations de synchronisme entre certaines réactions. Nous proposons par ailleurs une méthode d'analyse de la dynamique par interprétation abstraite qui permet de répondre à des questions d'atteignabilité. Cette méthode est spécifiquement adaptée à l'étude des modèles de grande taille, pouvant contenir plusieurs centaines de composants, et potentiellement davantage. Elle repose en effet sur une approximation de la dynamique qui évite ainsi l'explosion combinatoire inhérente à ce type d'analyse, permettant de répondre en quelques dixièmes de secondes au prix d'être parfois non conclusive. Enfin, nous traçons des liens formels entre les différents formalismes développés dans cette thèse, ainsi qu'avec plusieurs autres modélisations discrètes répandues. Nous permettons ainsi à un modèle de jouir des capacités de représentation et d'analyse de plusieurs formalismes à la fois.

Mots clés

réseaux de régulation biologique, frappes de processus, modèle de Thomas, formalisme asynchrone, analyse statique, atteignabilité

Abstract

Representing and analyzing large biological regulatory networks are the two main challenges of the understanding of the living machinery. The work we expose here focuses on discrete modeling, usually composed of graphs and sets of parameters, and especially on a previously developed framework called Process Hitting, which allows to give an atomistic representation of some components and their combined dynamics. In this thesis, we propose several new frameworks that consist of alternatives to the Process Hitting. Their higher expressivity permits to integrate discrete constraints into models based on the knowledge of reaction durations or synchronicity relationships. We also propose a new method to analyze the dynamics of such models by abstract interpretation which allows to answer reachability questions and is well-suited to large-scale models (made of hundreds of components, and potentially more). This method relies on an approximation of the dynamics that avoids the combinatorial explosion usually inherent to such analyses, thus answering in in tenths of a second at the price of being sometimes inconclusive. At last, we discuss the formal bonds between the different formalisms developed in this thesis and the link with some other widespread discrete modelings. We propose several translations from and to these other modelings, in order to benefit from the high power of modeling and analysis of these different frameworks.

Key Words

biological regulatory networks, process hitting, Thomas modeling asynchronous framework, static analysis, reachability