Université Paris 7 – Denis-Diderot
UFR d'Informatique

Thèse

pour l'obtention du diplôme de

Docteur de l'Université Paris 7, spécialité informatique

Étude d'un formalisme concurrent pour les phénomènes d'auto-organisation et la biologie moléculaire

présentée et soutenue publiquement par

Fabien Tarissan

le 13 décembre 2006

devant le jury composé de

Nadia Busi	rapporteuse
Alessandra Carbone	examinatrice
Guy Cousineau	président du jury
Vincent Danos	directeur de thèse
Jean-Louis Giavitto	rapporteur

Table des matières

Remerciements

Les remerciements constituent très certainement la partie de la thèse qui sera la plus lue, c'est donc avec grand plaisir que je me plie à la tradition. Je commence évidemment cette liste de personnes qui ont été présentes pendant ces trois années par Vincent Danos, mon directeur de thèse, qui a toujours porté un regard critique sur mon travail et m'a lancé dans ce milieu de la recherche. Je tiens tout particulièrement à remercier mes deux rapporteurs, Jean-Louis Giavitto et Nadia Busi qui ont accepté cette relecture attentive, ce qui a d'ailleurs constitué un retour très motivant sur mon travail. Je remercie également Guy Cousineau et Alessandra Carbone d'avoir accepté de faire partie de mon jury.

En second lieu, je remercie ma m'man, bien sûr, pour son soutien indéfectible et pour prendre soin du yuka qui occupe désormais à peu près la moitié de son salon ; Nany, tout particulièrement, pour tout ce qu'elle m'a apporté (apprentissage du ski à 3 ans et demi et concoction de la purée mousseline par inversion de quantité) et pour qui j'espère bien rester éternellement son « canard » ; Renée et Raymond pour avoir fait de Lanespède un lieu accueillant ; bref toutes les tarfouettes et autres kikis, pour leurs commentaires de tous les « dessins » de mes articles, pour m'avoir appris à distinguer les algues des choux-fleurs et qui peuvent témoigner que, s'il m'est arrivé de confondre une sardine avec un radis, cela n'a jamais empêché ma maturité intellectuelle de progresser. Ils ont par ailleurs joué le rôle important pour moi de ravitaillement en cuisine du sud-ouest et c'est peut-être ce qui me manque le plus en terme culinaire actuellement (j'dis ça, j'dis rien . . .)

De l'autre côté de la famille, je remercie Didi dont la systématicité à lire Le Monde a infléchi mes habitudes (en gros, je passe toutes mes matinées depuis trois ans à éplucher lemonde.fr) et qui a joué un rôle extrêmement apaisant dans le contexte de fin de rédaction de thèse en acceptant la mission encombrante, mais ô combien fondamentale, de re-re-lectrice orthographique du manuscrit en entier (je me décharge par conséquent de la responsabilité d'une quelconque erreur de ce type !) ; Nono qui a lui aussi contribué à apaiser mes inquiétudes les plus vives au sujet du stock de bois des Autels et qui m'a enseigné les techniques de bricolage à la pointe de la modernité quant à l'installation d'un porte-serviettes, et, plus généralement, aux membres de la famille Nollez-Goldbach qui ont choisi de ne pas écouter les avertissements proférés dès le début par Lolo qui les encourageait, à la vue de mon état de santé, à me remplacer par un modèle plus neuf.

Un grand merci à ma bande de potes, cette petite bande de 5 qui s'est constituée il y a pas mal d'années maintenant. Dans le cadre d'une thèse s'intéressant à la dynamique des réseaux, il est fascinant pour moi de constater que nos liens sont en constant mouvement et renouvellement et m'ont apporté

un inestimable soutien psychologique. Je remercie donc Pierrot, que certaines connaissent peut-être déjà sans le savoir sous le pseudonyme déconcertant de Myxomatose, alias Powlux, sur divers forum de rencontres (ça t'apprendra à m'envoyer des insultes par tchat ☺), pour m'avoir un jour sauvé d'une attaque de sangliers dans les Cévennes ; Julien, avec qui je suis fier d'avoir retrouvé le *King-Louis-Beat* et qui met, j'en suis sûr, autant d'ardeur à soigner les gens qu'à fixer des étagères ; Romain, dont la garde robe s'étend désormais du string léopard au tee-shirt moulant jaune en passant par son fameux tee-shirt Anarchie, aujourd'hui renié mais un jour porté fièrement le poing levé et que je remercie d'avoir agrémenté la Tombe Issoire de ses fameux bonhommes Jenlain et autres sucriers punks ; Mathieu, enfin, avec qui je garde le souvenir embrumé de nostalgie du *Ballet des Notes* et qui était particulièrement présent, sans se rendre compte de son importance, au moment crucial de mon choix pour l'informatique.

Autour de cette bande gravite aussi d'autres personnes comme Cécile avec qui j'ai notamment partagé la joie estudiantine de l'apprentissage de la photo argentique (sacré Tuong !) et du cha-cha-cha et que je remercie pour sa ponctualité, Sam qui a été à mes côtés en cette fin de thèse et qui a partagé certains des moments les plus angoissants (je parle du choix des liqueurs dans les cocktails), Phinou, avec qui j'ai fait mes premiers et uniques pas en 3DMax (Ahhh la joie enfantine de l'explosion de vaches et autres donuts ; je dois dire au passage que je suis assez déçu que ce concept n'ait pas repris pour la fin de son court-métrage *Steppin'In* [1]), Raphaël, chargé de superviser un stage de maîtrise qui s'est très vite transformé en apprentissage de la culture vietnamienne et avec qui j'ai adoré avoir tous ces échanges (de tennis comme de civilisation orientale) et que j'espère bien retrouver à HK avant qu'il ne revienne, Matthieu (Latapy) pour son ardeur méthodique à comparer tous les parfums de rhums arrangés, et, en vrac, car le temps presse au moment où j'écris ces lignes, Maureen, Assia, Germain, Daniel, Michel (ça, c'est fait !), Erwan pour sa dignité dans sa défaite au championnat de raclette, Julien, Géraldine, Milan and Co, Jean qui s'est révélé être un collaborateur en plus d'un pote, David dont j'attends avec fébrilité le prochain choix d'études, Séverine pour sa bonne humeur, Manu, Florianne, Ambroise, Vincent et Valent' pour leurs prestations à SingStar, Pasquale pour la recette du tiramisù, Christine pour avoir osé s'attaquer à la coupe de cheveux de Sam et avoir réglé le tout dernier détail administratif de ma thèse, Sylvain pour avoir veillé à mon orthographe lors de l'élaboration des feuilles de TD, et toute la petite bande de thésards [2] avec qui j'ai partagé quelques unes des luttes les plus revendicatives (je fais plus ici référence aux café-croissants pour le séminaire thésards qu'à la création de la cellule « Thésards en lutte »). Pour finir, il me faut bien avouer que cette thèse eut

[1] cf. `http://www.youtube.com/watch?v=B481HqXrSOc`

[2] Joachim, Samuel M, Nicolas, Grégoire, Marie, Laura, Pierre et tous les autres que j'oublie dans la précipitation

été bien ennuyeuse sans la diversité culinaire déployée par Raph (fondue bourguignonne au bœuf, fondue bourguignonne au poulet, fondue bourguignonne au steak haché, fondue bourguignonne au canard, ...).

Comme toujours, j'ai écrit ces lignes dans l'urgence d'une « deadline », terme souvent utilisé durant ces trois années pour justifier mes arrivées tardives et que j'invoque une dernière fois pour me blanchir face à ceux qui pourraient me blâmer pour la faute impardonnable de ne pas les mentionner dans cette page de remerciements.

Chapitre 1

Introduction

Théorie de la concurrence

C'est une situation courante en informatique de se retrouver confronté à l'opposition entre spécification et réalisation. Du point de vue d'un programmeur, la spécification désigne le comportement que doit avoir un programme dont il est en charge et la réalisation représente le programme lui-même. Si la spécification se décrit et se comprend assez simplement en général, la réalisation de cette spécification, en revanche, peut être arbitrairement compliquée en fonction des contraintes et des moyens qui sont mis à la disposition du programmeur. Face à cette complexité, il est naturel de se tourner vers la théorie pour y chercher des outils de type mathématique capables d'analyser les réalisations proposées, de prouver leur adéquation vis à vis de la spécification dont elles sont dépendantes, voire même de synthétiser automatiquement ces programmes. Dans le contexte actuel où les systèmes informatiques se caractérisent de plus en plus par la complexité des réseaux qui les relient, il faut souligner la particularité, extrêmement naturelle bien que très difficile à appréhender, qui fait que l'architecture des connexions qui forment ces réseaux est modifiée au cours de l'évolution des systèmes. On parle alors de systèmes *concurrents* [1] pour désigner le fait qu'ils s'exécutent de façon parallèle et de *mobilité* pour faire référence à la fois à la dynamique des connexions entre ces systèmes et à la possible migration des programmes s'exécutant alors dans des localités différentes.

L'objet de la théorie des *calculs de processus* est précisément de proposer des formalismes permettant l'analyse de ces systèmes. Les calculs proposés par Milner (CCS [Mil89b] et π-calcul [Mil99, SW01]), par exemple, mettent en avant la dissymétrie inhérente aux réseaux informatiques entre émetteur et récepteur (comme le modèle client/serveur de la plupart des applications web) et permettent une représentation simple mais suffisamment riche pour

[1] Le terme de concurrence est à prendre dans son acception anglaise et pourrait se traduire plus formellement par *mise en parallèle*.

beaucoup d'applications réparties. Chaque unité de calcul y est décrite par un *processus* capable d'interagir avec son environnement par le biais de synchronisations sur des *canaux de communication*. Outre la synchronisation, la communication entre processus permet l'échange de nouveaux noms de canaux explicitant ainsi le caractère dynamique des capacités d'interaction de chaque processus. Le calcul des Ambiants Mobiles dû à Cardelli et Gordon [CG00] donne en revanche une notion de *compartimentation* plus à même de mettre en valeur la notion de domaines dynamiques dans lesquels ont lieu les échanges d'information. Dans le présent travail de thèse, nous allons explorer des formalismes similaires, que l'on peut classer dans la catégorie des algèbres de processus et qui empruntent les mêmes concepts. La différence se situe dans le domaine applicatif qui est visé, à savoir l'auto-assemblage et la biologie moléculaire.

Dans de telles applications, la dissymétrie assez naturelle en programmation entre émetteur et récepteur n'est plus adaptée. Les mécanismes mis en jeu se placent dans un espace réel et s'appréhendent plus facilement comme des phénomènes de collisions. Le langage que nous développons ici repose fortement sur cette intuition, déjà développée dans des travaux antérieurs dus à Danos et Laneve [DL03, DL04]. Ceux-ci, poursuivant des idées énoncées par Fontana [FB96] et Regev [RSS01], proposaient un langage, le κ-calcul, dédié à la représentation formelle des protéines. Les liaisons entre protéines au sein d'un complexe y étaient représentées par une communication et la perpétuité de ces liaisons par le partage d'un nom de canal. Le langage présenté dans ce travail s'inspire largement de ces travaux et peut se voir, dans un premier temps, comme une généralisation du κ-calcul. Afin de souligner cette idée, nous nommerons ce langage $g\kappa$-calcul (pour « général κ-calcul »).

L'un des premiers apports du présent travail sera d'étudier le rapport entre le π-calcul et ces types de formalismes. Une partie importante de la thèse se concentrera ensuite sur les phénomènes d'auto-organisation, qui se ramènent, de façon plus générique, à la question de l'émergence d'une forme, d'une structure de connexion abstraite, ou plus généralement d'un phénomène collectif à partir de multiples interactions locales entre composants élémentaires. Telle qu'elle est énoncée, la question de l'auto-assemblage dépasse le strict cadre de l'informatique et se retrouve de fait dans des domaines aussi divers que la biologie moléculaire [HMC02], les calculs amorphes [Nag02], les nanotechnologies [DS03] ou encore la robotique distribuée [Kla02]. Comme nous le verrons, l'expressivité du langage permet de décrire ces transferts d'information dans des systèmes à base de réécriture de graphes, éventuellement structurés hiérarchiquement dans l'optique d'une application à la biologie moléculaire.

Deux approches opposées

Un des aspects intéressants de cette approche consiste en ce que la spécification (le phénomène collectif proprement dit), la réalisation (présentée sous forme d'algorithme), ainsi que la preuve de la bonne correspondance entre les deux, s'établissent dans un cadre théorique unique fourni par le langage. Les approches plus traditionnelles nécessitent en général deux cadres différents, l'un pour la spécification et l'autre pour la réalisation.

Cette approche correspond à celle dite de *haut en bas* (ou « top-down »), le haut faisant référence à la spécification et le bas à sa réalisation. Une autre façon de la présenter, et qui nous rapproche déjà plus de la biologie, consiste à dire que la spécification et la réalisation modélisent le même système mais à un niveau de détail différent. Ainsi, de façon totalement opposée, la biologie moléculaire présente le problème dans l'autre sens, de *bas en haut* (ou « bottom-up »). Ici la question est d'extraire, à partir d'une connaissance d'un système biologique décrit au niveau moléculaire et obtenue par le biais de technologies de plus en plus sophistiquées (puces à ADN, spectrographie de masse, etc...) et fournissant de très nombreux détails, une compréhension de ce système en terme de comportement. Le but est donc de s'abstraire des données concrètes fournies et de tenter d'en retirer une signification en termes *fonctionnels*, permettant de mettre en évidence les phénomènes étudiés. Autrement dit, dans le cadre de la biologie moléculaire, la réalisation *existe déjà* et c'est sa spécification que l'on tente de cerner.

Dans les deux approches – synthèse de la réalisation d'une spécification donnée et extraction d'un modèle abstrait à partir d'une description détaillée – c'est le même outil mathématique, la *bisimulation*, introduite dans [Par81], qui va servir à établir la relation entre les deux niveaux de description. Dans le premier cas, la bisimulation sert à prouver la correction des théorèmes proposés. Dans le second cas, elle permet d'approcher une idée d'équivalence observationnelle en biologie.

Ces deux approches se retrouvent clairement séparées dans le présent travail et nous présentons donc des résultats de types différents. Dans la première partie de la thèse, nous fournissons des théorèmes qui précompilent des systèmes réalisant des spécifications décrites sous forme d'arbres ou de graphes. Dans la deuxième partie, qui est relative à la biologie, la spécification n'étant pas connue, nous nous contentons de définir des formalismes, de montrer leur valeur descriptive sur quelques exemples et d'explorer des définitions pertinentes de bisimulations. Il est à noter par ailleurs que la biologie décrite ici est une version simplifiée de la biologie moléculaire purement qualitative (bien que l'on puisse en donner une version quantitative[2]).

[2]en se basant notamment sur les travaux de Gillespie [Gil76, Gil77] et de Priami [Pri95, PRSS01].

Une des principales contributions de cette thèse consiste donc, à partir d'une spécification d'une certaine nature exprimée sous forme de réécriture d'arbres ou de graphes, à fournir un système qui la réalise de manière hautement décentralisée, en ce sens que tous les processus ont exactement le même comportement initial et que toutes les interactions ne font intervenir que deux processus au plus. On s'approche ici du thème de l'algorithmique distribuée. Que ce soit dans le cas des arbres ou dans celui des graphes, des impasses inhérentes au traitement décentralisé des assemblages conduisent à l'étude des stratégies qu'adoptent les systèmes pour y échapper. Ceci se traduit, dans une approche générale, par l'étude du choix des règles régissant leur comportement, que l'on rend réversibles. Dans le cas des arbres, plus simple, on est à même de décrire directement le comportement réversible du système afin qu'il sorte de ces impasses. Pour le cas des graphes, un traitement plus drastique est effectué puisque les composantes partiellement assemblées qui sont bloquées dans leur évolution se voient totalement désassemblées et contraintes de recommencer depuis l'état initial. L'étude plus générale des algèbres de processus réversibles ne sera évoquée dans le présent travail qu'à titre prospectif.

Nous fournissons par ailleurs une implémentation en Caml qui simule cet algorithme pour les graphes. Cela amène une ouverture intéressante, sur laquelle nous ne nous attarderons pas dans cette thèse, qui permet d'introduire une véritable notion d'espace qui ne soit pas simplement un espace abstrait de connexions. Cette question de la localisation des connexions dans un espace réel trouve par ailleurs un écho naturel dans le domaine de la biologie.

Plan de la thèse

Le chapitre 2 présente le langage qui va être utilisé tout au long de cette étude, le $g\kappa$-calcul. Nous y définissons la syntaxe et la sémantique opérationnelle associée. Notre langage étant fortement inspiré des langages provenant de la famille des algèbres de processus, nous nous posons tout de suite la question de son rapport avec le π-calcul et nous montrons comment ce dernier peut être interprété comme un système de $g\kappa$-calcul. Les chapitres 3 et 4 se concentrent ensuite sur la question de l'auto-assemblage, le premier concernant celui des arbres et le second celui des graphes. Dans les deux cas, la spécification du problème ainsi que la solution proposée sont décrites dans $g\kappa$-calcul. Les deux algorithmes utilisés pour réaliser les assemblages distinguent clairement le comportement du système qui est propre à l'assemblage de celui défini pour gérer les situations d'impasse. Ceci amène notamment à discuter la possibilité d'intégrer une notion de *comportement arrière* dans le langage même pour éviter d'encombrer la définition des algorithmes avec de telles situations. Le chapitre 5 propose ensuite une implémentation de l'auto-assemblage de graphes établi au chapitre 4, en mettant en avant les choix, en termes de représentation, qui distinguent l'implémentation de l'algorithme. On trouvera là en particulier une

ouverture quant à la modélisation de l'algorithme dans un véritable espace euclidien dans lequel se placent les processus. Pour finir, le chapitre 6 présente une extension du langage décrit dans le chapitre 2 permettant la description et la simulation d'une partie importante de la biologie moléculaire. Cette extension permet de modéliser les interactions entre protéines et membranes ainsi que des mécanismes modifiant la structure interne des solutions (fusions de membranes, translocation, phagocytose, etc...). Nous donnons notamment la représentation de deux systèmes biologiques importants et bien connus – une voie de signalisation et une infection virale – qui nous servent de référents pour appréhender notre extension. Nous discutons par ailleurs différentes définitions de la bisimulation, celle-ci pouvant se révéler adaptée à l'étude de tels systèmes. Nous terminerons alors cette thèse par des conclusions et quelques remarques prospectives concernant des travaux futurs.

Chapitre 2

Un langage de collision

Sommaire

Ce premier chapitre se consacre essentiellement à la présentation du langage qui va être utilisé dans les chapitres 3 et 4 pour résoudre des problèmes d'auto-assemblage et qui sera étendu, dans le chapitre 6, pour tenir compte des caractéristiques propres à la biologie moléculaire qui nous intéresse. La section 2.1 présente donc la syntaxe et la sémantique de ce langage, nommé dans la suite $g\kappa$-calcul. Puis, la section 2.2 pose la question du rapport entre le $g\kappa$-calcul et le π-calcul et nous y montrons que tout processus peut-être associé à un système de $g\kappa$-calcul qui lui est équivalent. Ce plongement du π-calcul dans le $g\kappa$-calcul repose cependant sur un trait de construction fort du langage qui permet d'utiliser le mécanisme de substitution de nom comme une opération extérieure au langage. La section 2.3 explore alors la possibilité de donner une traduction implémentant directement ce mécanisme de substitution dans $g\kappa$-calcul.

2.1 Le langage

À la différence du π-calcul pour lequel la structure même des termes définit leur capacité d'interaction avec l'environnement, l'approche choisie dans le présent travail a été de découpler la définition des termes du langage de leur capacité d'interaction avec leur environnement. La structure d'un système devient donc indépendante de ses capacités d'interaction. Celles-ci sont paramétrées par un ensemble de règles différentes pour chaque système mais respectant un certain schéma.

2.1.1 Une syntaxe pour les agents

Pour garder l'analogie avec les processus du π-calcul décrits dans l'introduction, c'est-à-dire des unités de calculs autonomes capables d'interagir avec leur environnement, mais afin d'éviter toute confusion, nous utiliserons désormais le terme d'*agent* pour désigner ces unités de calcul et réserverons celui de processus pour le π-calcul exclusivement. Dans le même ordre d'idées, le mot de *communication* sera réservé aux réductions dans le π-calcul tandis que nous emploierons celui d'*interactions* lorsque nous nous placerons dans le $g\kappa$-calcul (terme faisant référence à l'aspect collision des applications visées). Les caractéristiques propres d'un agent sont données par son *type* (encore appelé son nom), et son *interface*. Soit $\mathcal{T}$ un ensemble dénombrable. Nous utiliserons les lettres $\mathtt{t}$, $\mathtt{u}$, ..., pour représenter le type d'un agent et les lettres ϕ, ψ, ... pour représenter les interfaces. L'interface d'un agent correspond à l'état interne de ce dernier à un moment donné de son évolution. À chaque type $\mathtt{t}$ d'agent correspond un type d'interface caractérisé par le nombre de *sites*, $\mathtt{n}(\mathtt{t})$, qui la composent et le type d'informations que peut contenir chacun de ces sites. Ces informations peuvent être des valeurs (des entiers par exemple) ou bien des noms de canaux (dénotant alors une connexion avec un autre agent). Soit $\mathcal{V}$ un ensemble dénombrable de valeurs notées par la suite U, V, ...et soit $\mathcal{C}$ un ensemble dénombrable de noms de canaux. Traditionnellement, nous désignerons ces canaux par les lettres x, y, z, etc... Un site se définit donc comme un ensemble prenant ces éléments soit dans l'ensemble de valeur $\mathcal{V}$ soit dans l'ensemble des canaux $\mathcal{C}$.

Définition 1 (Agents) *Soit $\mathcal{T}$ un ensemble de types associé à la fonction de type $\mathtt{n}(\cdot) : \mathcal{T} \mapsto \mathbb{N}$. Soit $\mathcal{V}$ un ensemble de valeurs et soit $\mathcal{C}$ un ensemble de noms de canaux. Un agent est formellement défini par une paire $(\mathtt{t}, \phi)$, encore écrite $\mathtt{t}\langle\phi\rangle$, vérifiant les conditions suivantes :*

- $\mathtt{t} \in \mathcal{T}$
- $\phi \in (\mathscr{P}(\mathcal{V}) \cup \mathscr{P}(\mathcal{C}))^{\mathtt{n}(\mathtt{t})}$

Pour prendre un exemple sommaire, imaginons une application répartie composée d'un serveur proposant un service simple comme celui du calcul

d'une somme, effectuée à partir d'entiers fournis par un client. Le serveur et le client vont naturellement avoir des comportements différents, ce qui se traduit dans notre langage par les types différents `serveur` et `client`. Le serveur étant a priori destiné à communiquer avec différents clients, il est nécessaire de distinguer les connexions établies avec ces derniers. Comme il est de coutume dans la tradition des algèbres de processus de type π-calcul, nous représentons ces connexions par des noms de canaux. Ainsi, chacune des interfaces de ces agents va requérir l'utilisation d'un site dans l'interface (choisissons arbitrairement le premier) pour mémoriser ce nom. Le reste de l'interface va servir à mémoriser les éléments transmis, à savoir le résultat de la somme en cours du côté serveur et l'ensemble des entiers qu'il reste à transmettre du côté client. Ainsi si nous prenons l'ensemble de valeurs $\mathcal{V} = \mathbb{N}$ et l'ensemble de types $\mathcal{T} = \{\texttt{client}, \texttt{serveur}\}$ avec $\mathfrak{n}(\texttt{client}) = \mathfrak{n}(\texttt{serveur}) = 2$, les agents $a = \texttt{client}\langle\{\mathrm{x}\}, \{1, 2, 6\}\rangle$ et $b = \texttt{serveur}\langle\{\mathrm{x}\}, \{9\}\rangle$ sont correctement définis, tandis que les agents $\texttt{client}\langle\{\mathrm{x}, 1\}, \{2, 6\}\rangle$ et $\texttt{client}\langle\{\mathrm{x}\}, \{1\}, \{2\}, \{6\}\rangle$ ne le sont pas. Le premier, qui n'a par ailleurs pas vraiment de sens dans l'exemple considéré, n'est pas autorisé dans notre syntaxe puisqu'il mélange, pour un même site, des valeurs et des noms[1]. Le second est, quant à lui, incorrect au regard du nombre de sites supérieur à la taille de l'interface.

Par souci de lisibilité, nous nous autorisons à n'écrire que l'information portée par un site lorsque celle-ci est un ensemble composé d'un seul élément. L'agent b peut donc aussi s'écrire sous la forme $\texttt{serveur}\langle \mathrm{x}, 9\rangle$. Par ailleurs, nous aurons parfois besoin de décrire les interfaces partielles des agents et nous utiliserons par abus de notation les même lettres ϕ, ψ, ... que pour les interfaces complètes et nous noterons $|\,\phi\,|$ le nombre de sites composant l'interface (partielle) ϕ.

Ainsi, à partir de la définition des agents, nous pouvons définir un système d'agents basé sur la grammaire suivante :

Définition 2 (Syntaxe du $g\kappa$-calcul) *Un* système d'agents *(statique) S est défini par la syntaxe donnée dans la figure 2.1.*

$$
\begin{array}{rcll}
S & := & \mathbf{0} & \textbf{(système vide)} \\
 & & \mathtt{t}\langle\phi\rangle & \textbf{(agent)} \\
 & & S\,,\,T & \textbf{(groupement)} \\
 & & (\nu\,\mathrm{x})\,S & \textbf{(restriction)}
\end{array}
$$

FIG. 2.1 – Syntaxe du $g\kappa$-calcul

[1]Une extension permettant un tel mélange ne changerait aucune des propriétés fondamentales du langage mais n'est pas nécessaire dans les applications considérées dans cette thèse.

Un système est soit un système vide $\emptyset$, soit un simple agent $\mathtt{t}\langle\phi\rangle$, soit un groupement d'agents $S\,,\,T$ dénotant la composition parallèle des systèmes S et T, soit un système S préfixé par l'opérateur $(\nu\,\mathrm{x})$ signifiant que la portée du nom de canal x est limitée au système S. La notion de restriction permet de définir l'ensemble des noms qui ont une portée restreinte de ceux qui ont une portée globale, aussi appelés *noms libres*. Dans le cas où un système d'agents S ne comporte aucune restriction on qualifiera ce système de *libre*. Par la suite, nous désignerons par $\mathcal{A}$ l'ensemble des systèmes d'agents définis par la syntaxe de la définition 2.

Définition 3 (Noms libres) *On définit l'ensemble des noms libres d'un système S, noté $\mathsf{nl}(S)$ par induction sur sa structure :*

$$\begin{aligned}
\mathsf{nl}(\mathbf{0}) &= \emptyset \\
\mathsf{nl}(\mathtt{t}\langle I_1,\ldots,I_n\rangle) &= (\cup_i I_i)\cap\mathcal{C} \\
\mathsf{nl}(S\,,\,T) &= \mathsf{nl}(S)\cup\mathsf{nl}(T) \\
\mathsf{nl}((\nu\,\mathrm{x})\,S) &= \mathsf{nl}(S)\setminus\{\mathrm{x}\}
\end{aligned}$$

Cette grammaire permet de définir de plusieurs manières un même système d'agents. Par exemple, le système $(\nu\,\mathrm{x})\,(\mathtt{a_1}\langle\mathrm{x}\rangle\ ,\ \mathtt{a_2}\langle\mathrm{x}\rangle)$ désigne un système très simple dans lequel deux agents possèdent un nom privé représenté ici par x. Bien entendu, le nom utilisé est totalement arbitraire et ce système pourrait très bien être représenté de façon équivalente par $(\nu\,\mathrm{y})\,(\mathtt{a_1}\langle\mathrm{y}\rangle\ ,\ \mathtt{a_2}\langle\mathrm{y}\rangle)$. Afin de prendre en compte cette redondance et de ne plus s'en préoccuper par la suite, nous définissons une congruence structurelle sur ces termes.

Définition 4 (Congruence structurelle) *La congruence structurelle, notée $\equiv$, est la plus petite relation sur $\mathcal{A}$, close par α-conversion et vérifiant les propriétés suivantes :*

1. *$(\mathcal{A}/\!\equiv,\ ',\ ',\ \emptyset)$ est un monoïde symétrique*
2. *$(\nu\,\mathrm{x})\,(\nu\,\mathrm{y})\,S\equiv(\nu\,\mathrm{y})\,(\nu\,\mathrm{x})\,S$*
3. *$(\nu\,\mathrm{x})\,S\equiv S$ si $\mathrm{x}\notin\mathsf{nl}(S)$*
4. *$(\nu\,\mathrm{x})\,S\,,\,T\equiv(\nu\,\mathrm{x})\,(S\,,\,T)$ si $\mathrm{x}\notin\mathsf{nl}(T)$*

La congruence structurelle permet alors d'utiliser une nouvelle notation pour les noms restreints et nous représenterons à la fois $(\nu\,\mathrm{x})\,(\nu\,\mathrm{y})$ et $(\nu\,\mathrm{y})\,(\nu\,\mathrm{x})$ par la notation $(\nu\,\mathrm{xy})$. Plus généralement, nous parlerons d'un ensemble de noms $\tilde{x}$ restreint au système S et nous désignerons ce système par $(\tilde{\mathrm{x}})\,S$. Par ailleurs, nous abrégerons le groupe $\mathtt{a_1}\langle\sigma_1\rangle\ ,\ \cdots\ ,\ \mathtt{a_n}\langle\sigma_n\rangle$ par $\prod_{i\in 1..n}\mathtt{a_i}\langle\sigma_i\rangle$.

2.1.2 La sémantique

Parallèlement à la définition statique du système donnée par la grammaire de $\mathcal{A}$, il nous faut définir quelles sont les interactions autorisées, entre quels types d'agents et quel va en être l'impact sur l'interface des agents. On définit donc un ensemble de règles, différent pour chaque système que l'on veut modéliser. Une règle d'interaction se présente comme une paire (S_G, S_D) où S_G représente une partie recherchée de l'état actuel du système et S_D l'état de cette partie résultant de l'application de la règle.

Définition 5 (Règle d'interaction) *Soit S_G et S_D deux systèmes d'agents libres. La paire $(S_G, (\tilde{\mathrm{x}})\, S_D)$, aussi notée $S_G \longrightarrow (\tilde{\mathrm{x}})\, S_D$ est une* règle d'interaction *si la condition $\mathsf{nl}((\tilde{\mathrm{x}})\, S_D) \subseteq \mathsf{nl}(S_G)$ est vérifiée.*

Notons que le fait d'utiliser des systèmes libres contraint un peu les règles puisque tous les noms créés se retrouvent en tête du membre droit. Cette restriction n'est cependant que de façade puisque, par α-conversion, tout système S peut être associé à un système $(\tilde{\mathrm{x}})\, \mathrm{S'}$ congruent dans lequel S' est un système libre. D'autre part, du point du vue des noms apparaissant dans l'interface des agents concernés, une règle d'interaction permet de les séparer en trois ensembles distincts :

- l'ensemble des noms créés par la règle : $\{\tilde{x}\}$
- l'ensemble des noms préservés par la règle : $\mathsf{nl}(S_G) \cap \mathsf{nl}((\tilde{\mathrm{x}})\, S_D)$
- l'ensemble des noms supprimés par la règle : $\mathsf{nl}(S_G) \setminus \mathsf{nl}((\tilde{\mathrm{x}})\, S_D)$

La condition posée sur les noms libres dans la définition 5 s'interprète alors comme le seul fait de garantir que les noms apparaissant dans le membre droit de la règle sont soit préservés, soit créés par la règle.

Pour appliquer une règle d'interaction $S_G \longrightarrow (\tilde{\mathrm{x}})\, S_D$ à un système S et le faire ainsi évoluer, il suffit de réussir à extraire une partie de S « correspondant » au membre gauche de la règle et de remplacer cette partie par le membre droit. Pour cela, il faut s'autoriser une certaine souplesse sur la notion de correspondance. En particulier, les noms de canaux ayant avant tout un sens de connexion entre les agents, il paraît naturel de s'abstraire du nom proprement dit pour n'en garder que cette notion de liaison. C'est pourquoi on se permet un *renommage* de ces noms au moment de l'extraction de la partie du système correspondante au schéma défini par S_G.

Définition 6 (Renommage) *Une fonction de renommage est une fonction partielle injective sur $\mathcal{C}$.*

Le renommage des noms de canaux devient alors une condition de filtrage sur un système pour savoir si on peut lui appliquer la règle ou non.

Définition 7 (Filtrage) *Soit* $\mathfrak{r}$ *la règle d'interaction*

$$\mathtt{t}_1\langle\phi_1\rangle\ ,\ \ldots\ ,\ \mathtt{t}_n\langle\phi_n\rangle \longrightarrow (\tilde{\mathrm{x}})\,(\mathtt{t}'_1\langle\phi'_1\rangle\ ,\ \ldots\ ,\ \mathtt{t}'_m\langle\phi'_m\rangle)$$

La paire de systèmes (S,T) *est dite* filtrée *par* $\mathfrak{r}$*, aussi noté* $S \xrightarrow{\mathfrak{r}} T$ *s'il existe un renommage* $\rho : \mathcal{C} \mapsto \mathcal{C}$ *tel que :*

- $S = \mathtt{t}_1\langle\rho(\phi_1)\rangle\ ,\ \ldots\ ,\ \mathtt{t}_n\langle\rho(\phi_n)\rangle$
- $T = (\rho(\tilde{x}))\,(\mathtt{t}'_1\langle\rho(\phi'_1)\rangle\ ,\ \ldots\ ,\ \mathtt{t}'_m\langle\rho(\phi'_m)\rangle)$

Ce filtrage direct est ensuite clos par les constructeurs du langage et permet de définir la sémantique du langage à l'aide d'un système de transition.

Définition 8 (Sémantique du $g\kappa$-calcul) *Soit* $\mathfrak{R}$ *un ensemble de règles d'interaction. Les règles données par la figure 2.2 définissent une relation binaire* $\rightarrow_{\mathfrak{R}}$*, appelée* relation de transition*, sur* $\mathcal{A}$*.*

$$(\mathsf{DIR})\ \frac{S \xrightarrow{\mathfrak{r}} T \quad \mathfrak{r} \in \mathfrak{R}}{S \rightarrow_{\mathfrak{R}} T} \qquad \frac{S \rightarrow_{\mathfrak{R}} T}{S\,,\,U \rightarrow_{\mathfrak{R}} T\,,\,U}\ (\mathsf{GROUP})$$

$$(\mathsf{NEW})\ \frac{S \rightarrow_{\mathfrak{R}} T}{(\nu\,\mathrm{x})\,S \rightarrow_{\mathfrak{R}} (\nu\,\mathrm{x})\,T} \qquad \frac{S \equiv S' \quad S' \rightarrow_{\mathfrak{R}} T' \quad T' \equiv T}{S \rightarrow_{\mathfrak{R}} T}\ (\mathsf{STRUCT})$$

FIG. 2.2 – Règles de réduction du $g\kappa$-calcul

Nous écrirons simplement $\rightarrow$ au lieu de $\rightarrow_{\mathfrak{R}}$ lorsqu'il n'y aura pas de confusion possible.

Enfin, nous sommes maintenant capables de définir complètement les caractéristiques propres à un *système dynamique d'agents.*

Définition 9 (Les systèmes dynamiques d'agents) *Un système (dynamique)* $\mathcal{S}$ *du g*κ*-calcul est défini par le n-uplet*

$$\mathcal{S} = (\mathcal{T}, \mathfrak{n}(\cdot), \mathcal{V}, \mathcal{C}, \mathfrak{R}, S_0)$$

où $\mathcal{T}$ *est un ensemble de types,* $\mathfrak{n}(\cdot)$ *une fonction de type,* $\mathcal{V}$ *un ensemble de valeurs,* $\mathcal{C}$ *un ensemble de noms de canaux,* $\mathfrak{R}$ *un ensemble de règles d'interaction et* S_0 *un système d'agents correspondant à l'état initial du système dynamique.*

Le terme de « système dynamique d'agents » s'oppose au terme de « système statique d'agents » de la définition 2. Un système statique d'agents doit être entendu en réalité comme l'espace des états d'un système dynamique. Par abus de langage nous emploierons le même terme de « système » pour désigner les

deux notions. Le contexte se chargera de préciser quelle est la notion formelle véritablement utilisée.

Pour finir, la figure 2.3 permet de lister les typographies qui vont être utilisées dans le présent manuscrit. Dans le soucis de ne pas surcharger les notations, seules celles qui seront utilisées par la suite sont présentées.

	Élements	Ensemble
Types d'un agent	$\mathtt{t}$, $\mathtt{u}$	$\mathcal{T}$
Valeurs	U, V	$\mathcal{V}$
Noms de canaux	x, y	$\mathcal{C}$
Sites d'une interface	s, t	
Interfaces (partielles)	ϕ, ψ	
Renommage	ρ, σ	
Agents	a, b	
Systèmes d'agents	S, T	$\mathcal{A}$
Systèmes dynamiques	$\mathcal{S}$, $\mathcal{T}$	$\mathcal{D}$
Règles d'interaction	$\mathfrak{r}$, $\mathfrak{s}$	$\mathfrak{R}$

FIG. 2.3 – Les différentes typographies utilisées dans la thèse

2.2 Ré-interprétation du π-calcul

En tant que premier exemple complet de système dynamique décrit dans le $g\kappa$-calcul, nous pouvons tenter de répondre à une question simple : est-il possible de décrire le π-calcul dans notre langage ? On apporte ainsi un premier résultat sur l'expressivité du $g\kappa$-calcul et on précise les intuitions relatives au découpage syntaxe/règles.

2.2.1 Présentation du π-calcul

Comme nous l'avons dit en introduction, le π-calcul est le formalisme de référence en ce qui concerne la modélisation des systèmes concurrents. L'une de ses particularités les plus pertinentes consiste en la distinction qui est faite, lors d'une communication, entre l'émetteur et le récepteur. Cette distinction se caractérise syntaxiquement par l'utilisation de deux constructeurs complémentaires dans le langage. Les processus du π-calcul se définissent formellement par la syntaxe présentée dans la figure 2.4.x

$$\begin{array}{lrcl} \text{Capacités} & \alpha & := & \bar{\mathrm{x}}\mathrm{y} \mid \mathrm{x}(\mathrm{y}) \mid \tau \\ \text{Processus} & p & := & \mathbf{0} \mid \sum_i \alpha_i.p_i \mid (p \mid p) \mid D(\tilde{\mathrm{x}}) \mid (\tilde{\mathrm{x}})\, p \end{array}$$

FIG. 2.4 – Syntaxe du π-calcul

Les processus se voient dotés de *capacités*, dénotées par les α_i (aussi appelées *actions*), précisant si la communication offerte par ce processus correspond à l'émission d'un nom y sur un canal x (construction $\bar{\mathrm{x}}\mathrm{y}$), à la réception d'un nom z sur un canal x (construction $\mathrm{x}(\mathrm{z})$), ou bien si cette capacité est *silencieuse* (τ), en ce sens qu'elle permet au processus d'évoluer sans interagir avec le contexte. Le processus complet se définit à partir de cette notion de capacité en ajoutant la possibilité de proposer un choix exclusif sur les capacités de communication d'un processus $(p + q)$, de définir la composition parallèle $(p \mid q)$, la restriction $((\nu\, \mathrm{x})\, p)$, ainsi que la définition récursive d'un processus paramétré par un ensemble de noms libres ($D(\tilde{\mathrm{x}})$). Plus précisément, le constructeur qui va requérir toute notre attention lors de la traduction du π-calcul dans le $g\kappa$-calcul est celui de la somme. La somme (finitaire) $\sum_i \alpha_i.p_i$ exprime le fait que ce processus présente un certain nombre de choix de communications (potentiellement silencieuses). Si l'un de ces choix α_i est effectué, alors les autres possibilités sont oubliées et seule la continuation p_i est conservée. Par la suite, nous désignerons par Π l'ensemble des termes du π-calcul.

Pour conclure cette présentation, il nous reste à définir la sémantique du π-calcul. Le point crucial tient en la définition de la communication entre deux processus. Celle-ci a lieu lorsque deux processus présentent des capacités complémentaires, c'est-à-dire une émission $\bar{\mathrm{x}}\mathrm{y}.p$ sur un canal x et une réception $\mathrm{x}(\mathrm{z}).q$ sur ce même canal. Dans ce cas, le nom z du processus $\mathrm{x}(\mathrm{z}).q$ doit se comprendre comme une variable dont toutes les occurrences présentes dans q vont être instanciées par le nom y reçu. Ceci induit naturellement une notion de noms libres sur les termes du π-calcul, proche de celle de la définition 3, comme le montre la figure 2.5.

Par ailleurs, nous définissons la congruence structurelle $\equiv_\pi$ comme la plus petite relation d'équivalence sur les termes du π-calcul qui soit close par la somme, le produit et la restriction et telle que la somme et le produit soient associatifs, commutatifs et prennent $\mathbf{0}$ pour élément neutre. Nous ajoutons également la règle $D(\tilde{x}) \equiv_\pi p$ si $D(\tilde{x}) = p$, qui permet de déplier la définition du processus récursif lorsque nous en avons besoin.

La communication entre deux processus peut maintenant être définie syntaxiquement à l'aide de la substitution du canal z par le canal y dans la continuation q du processus récepteur $\mathrm{x}(\mathrm{z}).q$. Les autres règles de communication sont très semblables à celles données par la définition 8. La figure 2.6 reprend l'ensemble des règles de réduction définissant la sémantique opérationnelle du

$$
\begin{array}{lcl}
\mathsf{nl}(\bar{x}y.p) & = & \mathsf{nl}(p) \cup \{x, y\} \\
\mathsf{nl}(x(y).p) & = & \mathsf{nl}(p) \cup \{x\} - \{y\} \\
\mathsf{nl}(\tau.p) & = & \mathsf{nl}(p) \\
\mathsf{nl}(D(\tilde{x})) & = & \tilde{x} \\
\mathsf{nl}(p \mid q) & = & \mathsf{nl}(p) \cup \mathsf{nl}(q) \\
\mathsf{nl}(p + q) & = & \mathsf{nl}(p) \cup \mathsf{nl}(q) \\
\mathsf{nl}((\nu x)p) & = & \mathsf{nl}(p) \setminus \{x\}
\end{array}
$$

FIG. 2.5 – Définition des noms libres dans le π-calcul

π-calcul.

$$
\overline{(\bar{x}z.p + p') \mid (x(y).q + q') \to p \mid q\,\{z/y\}} \qquad \overline{\tau.p \to p}
$$

$$
\frac{p \to p'}{p \mid q \to p' \mid q} \qquad \frac{p \equiv_\pi p' \to q' \equiv_\pi q}{p \to q} \qquad \frac{p \to p'}{(\nu x)p \to (\nu x)p'}
$$

FIG. 2.6 – Les règles de réduction du π-calcul

2.2.2 Un codage syntaxique

Nous cherchons maintenant à interpréter les termes du π-calcul dans notre langage, c'est-à-dire définir $\mathcal{T}$ et $\mathcal{V}$ ainsi qu'un ensemble de règles d'interaction à même de simuler n'importe quel processus. Étant donné que les processus du π-calcul n'ont pas de nom, et que chaque processus interagit de la même façon avec son environnement, il paraît naturel de n'avoir qu'un type unique, `proc`, pour les représenter dans notre langage.

Les processus. Concentrons nous tout d'abord sur un terme simple de la forme $\sum_i \alpha_i.p_i$. Ce terme représente un processus offrant un certain nombre de noms de canaux – les α_is – et permettant des interactions avec son environnement. Ces interactions précisent plusieurs éléments : le nom du canal sur lequel le processus effectue une communication (ou τ dans le cas d'un choix interne), le sens de transmission (émission ou réception) ainsi que le nom de canal transmis (ou reçu) le cas échéant. Chacune de ces communications révèle alors de nouvelles possibilités, définies par la continuation p_i correspondante. L'idée naturelle de la traduction est de représenter un tel processus dans $g\kappa$-calcul

par un unique agent dont l'interface va mémoriser chacune de ces informations. Chacune des possibilités offertes par un processus $\alpha_i.p_i$ va requérir 4 sites pour mémoriser ces informations. Nous utiliserons un premier site pour représenter le nom de canal utilisé pour communiquer, un second site pour mémoriser le sens de communication, un troisième site pour représenter le nom de canal transmis et un quatrième site pour représenter la continuation du processus dans le cas où cette communication est choisie.

Plus formellement, on se donne l'ensemble de valeurs $\mathcal{V}_1 = \{\text{E}, \text{R}, \text{T}\}$ pour représenter respectivement les émissions, les réceptions et les τ-capacités. Étant donné une capacité α, on définit alors la fonction $\mathsf{n}(\alpha)$ (resp. $\mathsf{d}(\alpha)$ et val) à valeurs dans $\mathcal{C}$ (resp. $\mathcal{V}_1$ et $\mathcal{C}$) permettant de connaître le nom du canal (resp. le sens de communication et le nom de canal transmis) dénoté par α, c'est-à-dire tels que :

$$\mathsf{n}(\alpha) \quad := \quad \begin{cases} \{\text{x}\} & \text{si } \alpha \text{ est de la forme } \bar{\text{x}}\text{y ou x(y)} \\ \emptyset & \text{si } \alpha = \tau \end{cases}$$

$$\mathsf{d}(\alpha) \quad := \quad \begin{cases} \{\text{E}\} & \text{si } \alpha \text{ est de la forme } \bar{\text{x}}\text{y} \\ \{\text{R}\} & \text{si } \alpha \text{ est de la forme x(y)} \\ \{\text{T}\} & \text{si } \alpha = \tau \end{cases}$$

$$\mathsf{val}(\alpha) \quad := \quad \begin{cases} \{\text{y}\} & \text{si } \alpha \text{ est de la forme } \bar{\text{x}}\text{y ou x(y)} \\ \emptyset & \text{si } \alpha = \tau \end{cases}$$

D'autre part, puisque nous voulons mémoriser dans le quatrième site l'état du processus après une communication, il faut aussi se donner un ensemble de valeurs pour représenter les termes du π-calcul. Afin de conserver un peu de structure pour déchiffrer cet état lorsque le choix de communication aura été fait, il paraît naturel de représenter ces termes par des arbres étiquetés par les constructeurs du langage ('|', '+', $(\nu\,\text{x})$) et dont les feuilles sont les processus de la forme $\sum_i \alpha_i.p_i$ et $\emptyset$. Soit $\mathcal{V}_\pi$ l'ensemble de ces arbres, on peut alors redéfinir sur ces éléments l'α-conversion et la substitution comme pour le $g\kappa$-calcul ou le π-calcul. Afin d'alléger le codage qui va suivre, nous utiliserons ici les mêmes notations pour les termes du π-calcul que pour l'ensemble des valeurs qui les représentent[2].

Les trois fonctions précédemment définies permettent de structurer l'interface de l'agent représentant le processus $\sum_i \alpha_i.p_i$. Soit p un tel processus, on

[2]De même, pour être tout à fait correct, il faudrait distinguer les noms de canaux du π-calcul de ceux de $g\kappa$-calcul et établir une bijection entre les deux. On s'autorise ici à utiliser le même alphabet.

définit son interface $\mathsf{i}_\pi(p)$ par :

$$\mathsf{i}_\pi(\textstyle\sum_1^n \alpha_i.p_i) := \langle s_1, \ldots, s_{4n+4} \rangle$$

avec pour i de 1 à n :
— $s_{4i+1} = \mathsf{n}(\alpha_i)$,
— $s_{4i+2} = \mathsf{d}(\alpha_i)$,
— $s_{4i+3} = \mathsf{val}(\alpha_i)$.
— $s_{4i+4} = p_i$.

Il reste une dernière considération à prendre en compte, imposée par une contrainte sur la déclaration des règles d'interaction : les noms libres du membre droit d'une règle doivent apparaître dans le membre gauche. Or on représente temporairement les processus du π-calcul par des valeurs et donc en particulier les canaux de communication. Il est donc nécessaire de conserver en tant qu'éléments de $\mathcal{C}$ les noms de canaux qui vont être figés dans la valeur représentant le processus. On utilise pour cela un dernier site. Il conservera l'ensemble des noms de canaux connus (et donc utilisables plus tard) par un agent. Il est à noter que cette considération n'est pertinente que dans le cadre théorique de la vérification de la cohérence du système défini et se traduit formellement par le lemme 2, indispensable à la bonne formation des règles dans le $g\kappa$-calcul telles que présentées dans la définition 5. Il est évident que du point de vue plus intuitif de la connaissance par l'agent des noms de qui le lient aux autres agents, cette information est redondante puisque déjà présente (temporairement sous forme de valeur) dans les autres sites de l'agent.

Comme on l'a vu, la taille de l'interface d'un agent devant représenter le processus $\sum_i^n \alpha_i.p_i$ dépend de n. Comme pour un type donné une seule sorte d'interface est autorisée, il faut associer à chaque n un type différent $\mathtt{proc_n}$ tel que $\mathsf{n}(\mathtt{proc_n}) = 4n + 5$ ($4n + 4$ sites pour représenter les n choix offerts par le processus et 1 site pour mémoriser l'ensemble des noms connus). Chaque agent de type $\mathtt{proc_n}$ est donc prêt à accueillir l'interface définie par i_π.

Soit $[\![\cdot]\!]_\pi : \Pi \mapsto g\kappa$-calcul la fonction traduisant les termes du π-calcul dans la syntaxe du $g\kappa$-calcul. Soit p un processus de la forme $\sum_{i=1}^n \alpha.p_i$, on définit $[\![\sum_{i=1}^n \alpha_i.p_i]\!]_\pi$ par

$$[\![p]\!]_\pi := \mathtt{proc_n}\langle \mathsf{i}_\pi(p)\,, \mathsf{nl}(p) \rangle$$

Le processus $\bar{\mathrm{x}}\mathrm{y}.\mathrm{y}().\emptyset + \bar{\mathrm{x}}\mathrm{z}.\mathrm{z}().\emptyset$ devient donc, suivant cette représentation, l'agent $\mathtt{proc_2}\langle \mathrm{x}, \mathrm{E}, \mathrm{y}, \mathrm{y}().\emptyset, \mathrm{x}, \mathrm{E}, \mathrm{z}, \mathrm{z}().\emptyset, \{\mathrm{x}, \mathrm{y}, \mathrm{z}\} \rangle$.

Étant donné que l'interface $\mathsf{i}_\pi(p)$ cache une grande partie du processus en tant que valeur, ne laissant visibles que les noms apparaissant dans les α_i du processus, on en déduit immédiatement que les noms libres de cette interface sont tous contenus dans l'ensemble des noms libres du processus en entier. Ce qui se traduit par la propriété suivante :

Lemme 1 (Noms libres des agents stabilisés) *Pour tout processus p de la forme $\sum_{i=1}^{n} \alpha_i.p_i$ on a* $\mathsf{nl}(\mathtt{proc}_{\mathtt{n}}\langle \mathsf{i}_\pi(p)\,,\mathsf{nl}(p)\rangle) \subseteq \mathsf{nl}(p)$

Les autres termes du π-calcul se traduisent naturellement dans $g\kappa$-calcul en s'appuyant sur la similarité entre les opérateurs des deux langages.

$$\begin{aligned}
[\![p \mid q]\!]_\pi &:= [\![p]\!]_\pi \;,\; [\![q]\!]_\pi \\
[\![D(\tilde{x})]\!]_\pi &:= [\![p]\!]_\pi \text{ si } D(\tilde{x}) = p \\
[\![(\nu\tilde{x})\,p]\!]_\pi &:= (\nu\tilde{x})\,[\![p]\!]_\pi
\end{aligned}$$

Les règles d'interaction pour le π-calcul. Il reste maintenant à établir les règles qui vont permettre aux agents ainsi définis d'interagir. Ces règles vont en réalité simuler les communications binaires de π-calcul en travaillant en deux temps. Tout d'abord, il faut trouver deux agents présentant une interface complémentaire, c'est-à-dire présentant un nom de canal commun mais avec une direction opposée. C'est le rôle des interfaces de type s_{4i+1} et s_{4i+2}. L'effet de l'interaction est de révéler les futures capacités d'interaction du processus. C'est ce qui est mémorisé dans l'interface s_{4i+4}.

Cependant, cette étape présente une difficulté formelle, qui vient du fait que le devenir du processus est pour l'instant une valeur du langage et non pas encore une interface structurée. Il y a donc une étape intermédiaire qui consiste à déchiffrer cette valeur pour la rendre opérationnelle en tant qu'agent interagissant. Pour cela, on définit un nouveau type `tmp` correspondant à un agent ayant deux sites ($\mathtt{n}(\mathtt{tmp}) = 2$), le premier mémorisant le « futur » de l'agent et le second l'ensemble de noms connus par cet agent.

Pour comprendre un peu mieux l'agencement de tous ces renseignements, regardons comment se traduit la principale règle de communication du π-calcul, à savoir

$$\bar{\mathrm{x}}\mathrm{y}.p' + p'' \mid \mathrm{x}(\mathrm{z}).q' + q'' \longrightarrow p' \mid q'\{\mathrm{y}/\mathrm{z}\}$$

La seule condition posée sur la communication est de posséder un nom de canal commun mais dual. Pour tout processus $p = \sum_{i=1}^{n_1} \alpha_i.p_i$ et $q = \sum_{j=1}^{n_1} \beta_j.q_j$ tel que $\mathsf{i}_\pi(p) = \langle \phi, \mathrm{x}, \mathrm{E}, \mathrm{y}, p', \phi' \rangle$ et $\mathsf{i}_\pi(q) = \langle \psi, \mathrm{x}, \mathrm{R}, \mathrm{z}, q', \psi' \rangle$, on définit la règle :

$$(\mathsf{com})\ \frac{\mathtt{proc}_{\mathtt{n}_1}\langle \phi, \mathrm{x}, \mathrm{E}, \mathrm{y}, p', \phi', \mathsf{nl}(p)\rangle \;,\; \mathtt{proc}_{\mathtt{n}_2}\langle \psi, \mathrm{x}, \mathrm{R}, \mathrm{z}, q', \psi', \mathsf{nl}(q)\rangle}{\mathtt{tmp}\langle p', \mathsf{nl}(p')\rangle \;,\; \mathtt{tmp}\langle q'\{\mathrm{y}/\mathrm{z}\}\,, \mathsf{nl}(q'\{\mathrm{y}/\mathrm{z}\})\rangle}$$

Il convient de préciser qu'on définit ici une infinité de règles de cette forme pour chaque couple d'entiers (n_1, n_2). Ceci vient d'une des contraintes que l'on a posées sur la syntaxe des termes dans $g\kappa$-calcul, à savoir que la taille de l'interface des agents est définie une fois pour toutes lors de la description du système.

L'autre règle de communication du π-calcul $\tau.q + r \longrightarrow q$ se décline sur le même mode. Pour tout processus $p = \sum_{i=1}^{n} \alpha_i.p_i$ tel que $\mathtt{i}_\pi(p) = \langle \phi, \emptyset, \mathrm{T}, \emptyset, q, \phi' \rangle$, on ajoute la règle :

$$(\mathsf{tau})\ \frac{\mathtt{proc_n}\langle \phi, \emptyset, \mathrm{T}, \emptyset, q, \phi', \mathsf{nl}(p) \rangle}{\mathtt{tmp}\langle q, \mathsf{nl}(q) \rangle}$$

Reste maintenant à traduire l'état interne d'un agent de type `tmp` afin de rendre actif la suite du processus mémorisé en tant que valeur dans le seul site de son interface. Une manière assez naturelle de procéder serait d'avoir la règle

$$\frac{\mathtt{tmp}\langle p, \mathsf{nl}(p) \rangle}{[\![p]\!]_\pi}$$

dans notre système puisque c'est exactement l'intuition que renferme $\mathtt{tmp}\langle p, \mathsf{nl}(p) \rangle$. Mais, là encore, cela est impossible à cause de la structure imposée sur les règles. Le membre droit d'une règle d'interaction doit impérativement être de la forme $(\tilde{\mathrm{x}})\, S$ où S est un système libre. Or, évidemment, le terme $[\![p]\!]_\pi$ n'est pas forcément structuré de cette manière là. On découpe donc cette règle incorrecte en autant de règles qu'il y a de constructeurs possibles pour le processus p[3]. L'opérateur $p \,|\, q$ donne donc la règle :

$$(\mathsf{par})\ \frac{\mathtt{tmp}\langle p \,|\, q, \mathsf{nl}(p \,|\, q) \rangle}{\mathtt{tmp}\langle p, \mathsf{nl}(p) \rangle\ ,\ \mathtt{tmp}\langle q, \mathsf{nl}(q) \rangle}$$

Pour chaque définition récursive dans le π-calcul, on rajoute la règle

$$(\mathsf{def})\ \frac{\mathtt{tmp}\langle D(\tilde{x}), \mathsf{nl}(p) \rangle \qquad \text{si } D(\tilde{x}) = p}{\mathtt{tmp}\langle p, \mathsf{nl}(p) \rangle}$$

Enfin la règle de restriction peut maintenant correctement s'écrire en

$$(\mathsf{restric})\ \frac{\mathtt{tmp}\langle (\tilde{\mathrm{x}})\, p, \mathsf{nl}((\tilde{\mathrm{x}})\, p) \rangle}{(\tilde{\mathrm{x}})\, \mathtt{tmp}\langle p, \mathsf{nl}(p) \rangle}$$

Reste la règle permettant de rendre l'agent actif à nouveau. Celle-ci intervient lorsque la valeur mémorisée dans l'interface de l'agent `tmp` est de la

[3]Une autre possibilité serait d'effectuer un travail sur la valeur p afin de déterminer la valeur d'un processus équivalent mais dont la structure serait admise dans le membre droit d'une règle.

forme $\sum_i \alpha_i.p_i$. On se sert à nouveau de la fonction $\mathfrak{i}_\pi$. Pour chaque processus $p = \sum_i^n \alpha_i.p_i$, on ajoute la règle

$$(\mathsf{act})\ \frac{\mathtt{tmp}\langle p, \mathsf{nl}(p)\rangle}{\mathtt{proc_n}\langle \mathfrak{i}_\pi(p)\,, \mathsf{nl}(p)\rangle}$$

Le système. On est désormais à même de définir formellement le système $g\kappa$-calcul capable de représenter n'importe quel processus p du π-calcul par le n-uplet $(\mathcal{T}, \mathfrak{n}(\cdot), \mathcal{V}, \mathcal{C}, \mathfrak{R}, [\![p]\!]_\pi)$ tel que :

- $\mathcal{T} = \{\mathtt{proc_n} \mid n \in \mathbb{N}\} \cup \{\mathtt{tmp}\}$
- $\mathfrak{n}(\mathtt{t}) = \begin{cases} n & \text{si } \mathtt{t} = \mathtt{proc_n} \\ 2 & \text{si } \mathtt{t} = \mathtt{tmp} \end{cases}$
- $\mathcal{V} = \{\mathrm{E}, \mathrm{R}, \mathrm{T}\} \cup \mathcal{V}_\pi$
- $\mathfrak{R} = \{(\mathsf{com}), (\mathsf{tau}), (\mathsf{par}), (\mathsf{def}), (\mathsf{restric}), (\mathsf{act})\}$

Il est à noter que les règles sont bien autorisées par le langage. En particulier l'ensemble des noms libres apparaissant dans le membre droit d'une règle est bien inclus dans celui du membre gauche.

Lemme 2 (Bonne formation des règles) $\forall \mathfrak{r} : S_G \to S_D \in \mathfrak{R}$, $\mathsf{nl}(S_D) \subseteq \mathsf{nl}(S_G)$

Démonstration : Remarquons tout d'abord que pour toutes les règles, excepté (act), le membre droit contient exclusivement des agents de type $\mathtt{tmp}$. Comme le premier site ne contient que des valeurs, seul le second site détermine l'ensemble des noms libres de l'agent. La preuve se fait par cas sur les règles de $\mathfrak{R}$.

Cas (com) : Soit a, a', b et b' les quatre agents impliqués dans la règle :

$$\begin{aligned} a &= \mathtt{proc_{n_1}}\langle \phi, \mathrm{x}, \mathrm{E}, \mathrm{y}, p', \phi', \mathsf{nl}(p)\rangle \\ b &= \mathtt{proc_{n_2}}\langle \psi, \mathrm{x}, \mathrm{R}, \mathrm{z}, q', \psi', \mathsf{nl}(q)\rangle \\ a' &= \mathtt{tmp}\langle p', \mathsf{nl}(p')\rangle \\ b' &= \mathtt{tmp}\langle q'\{\mathrm{y}/\mathrm{z}\}\,, \mathsf{nl}(q'\{\mathrm{y}/\mathrm{z}\})\rangle \end{aligned}$$

Par définition, l'ensemble des noms libres de a , b est l'union de l'ensemble des noms libres de a et de b. Or l'hypothèse posée sur l'interface de a assure que $\langle \phi, \mathrm{x}, \mathrm{E}, \mathrm{y}, p', \phi'\rangle = \mathfrak{i}_\pi(p)$. Par application du lemme 1 il vient que $\mathsf{nl}(a) = \mathsf{nl}(p)$. Par ailleurs, étant donné que l'agent a' est de type $\mathtt{tmp}$, on a $\mathsf{nl}(a') = \mathsf{nl}(p')$. Or la définition de $\mathfrak{i}_\pi$ implique en particulier qu'il existe i tel $p' = p_i$, ce qui entraîne la relation d'inclusion $\mathsf{nl}(p') \subseteq \mathsf{nl}(p)$ et par suite $\mathsf{nl}(a') \subseteq \mathsf{nl}(a)$.

Par un raisonnement similaire, on établit aisément que $\mathsf{nl}(b') = \mathsf{nl}(q'\{\mathrm{y/z}\})$ et $\mathsf{nl}(q') \subseteq \mathsf{nl}(q)$, ce qui entraîne en particulier que $\mathsf{nl}(b') \subseteq \mathsf{nl}(q') \cup \{\mathrm{y}\} \subseteq \mathsf{nl}(q) \cup \{\mathrm{y}\} = \mathsf{nl}(b) \cup \{\mathrm{y}\}$. Or le nom potentiellement nouveau 'y' apparaît comme nom libre dans le membre gauche puisqu'il est visible dans l'interface de l'agent a. On en déduit donc $\mathsf{nl}(b') \subseteq \mathsf{nl}(b) \cup \mathsf{nl}(a)$ et on peut conclure que $\mathsf{nl}(a') \cup \mathsf{nl}(b') \subseteq \mathsf{nl}(a) \cup \mathsf{nl}(b)$.

Cas (tau) et (act) : Arguments similaires au cas précédent.

Cas (def) : Immédiat.

Cas (par) et (restric) : D'après la définition 3 de nl.

2.2.3 La correction

Dans la suite, nous appellerons *agent standard* tout agent de la forme $[\![\sum_i^n \alpha_i.p_i]\!]_\pi$ et par extension un *système standardisé* tout système $[\![p]\!]_\pi$. Il nous reste à prouver que la traduction proposée dans la section précédente est correcte. Pour ce faire, la notion usuelle utilisée, lorsque l'on compare des systèmes concurrents, est la bisimulation.

Définition 10 (Bisimulation faible) *Une bisimulation faible est une relation binaire R tel que pour tout couple $(P, Q) \in R$ les deux propriétés suivantes sont vérifiées :*

- *si $S \rightarrow_{\mathcal{S}} T$, alors il existe T' tel que $S' \rightarrow^*_{\mathcal{S}'} T'$ et $T \approx T'$*
- *si $S' \rightarrow_{\mathcal{S}'} T'$, alors il existe T tel que $S \rightarrow^*_{\mathcal{S}} T$ et $T \approx T'$*

Deux systèmes de transition $\mathcal{S} = (S_0, \rightarrow_{\mathcal{S}})$ et $\mathcal{S}' = (S'_0, \rightarrow_{\mathcal{S}'})$ sont faiblement bisimilaires, ce qu'on notera $\mathcal{S} \approx \mathcal{S}'$, s'il existe une bisimulation faible contenant (S_0, S'_0).

Comme nous l'avons vu dans la section précédente, l'ensemble des règles proposées pour modéliser le π-calcul dans notre système se répartit en deux groupes. Les règles (com) et (tau) qui simulent les primitives d'interactions du π-calcul d'une part – (com) pour une transaction complémentaire et (tau) pour une τ-capacité – et, d'autre part, les règles (par), (restric), (def) et (act) qui ne sont là que pour révéler les nouvelles capacités offertes par le résultat d'une interaction entre deux agents de type `proc`. Soit $\mathfrak{S}$ l'ensemble de ces règles, deux relations en particulier vont nous servir à établir la bisimulation entre le π-calcul et la traduction proposée :

Définition 11 *Soient $\mathcal{R}$ et $\mathcal{R}_{\mathfrak{S}}$ deux relations binaires entre le Π et $g\kappa$-calcul définies par*

$$\begin{aligned} \mathcal{R} &= \{(p, [\![q]\!]_\pi) \mid q \equiv_\pi p\} \\ \mathcal{R}_{\mathfrak{S}} &= \{(p, S) \mid S \rightarrow^*_{\mathfrak{S}} S' \quad (p, S') \in \mathcal{R}\} \end{aligned}$$

On le voit, $\mathfrak{S}$ joue un rôle particulier dans la vérification de la correction de la traduction puisque, comme le montrera le théorème 1 en fin de section, $\mathcal{R}_{\mathfrak{S}}$ *est* la bisimulation $\approx$ de la définition 10 mettant en relation les termes du π-calcul et ceux de la traduction. Cet ensemble présente en effet quelques propriétés qui vont nous aider à découper la preuve en éléments plus simples. On peut par exemple remarquer que toutes ces règles ont comme prémisse un agent $\mathtt{tmp}\langle p, \mathsf{nl}(p)\rangle$ et dépendent en fait fortement de la structure de p. Pour chaque construction, une seule règle est applicable :

Remarque 1 *Si une règle de $\mathfrak{S}$ s'applique, elle ne concerne qu'un seul agent de type* `tmp` *et elle seule peut s'appliquer à cet agent.*

Par ailleurs, pour toutes ces règles, aucune interaction avec l'environnement de l'agent n'est effectuée. Il est donc toujours possible de standardiser un agent de type `tmp`.

Lemme 3 (Standardisation) *Pour tout processus p du π-calcul on a*

$$\mathtt{tmp}\langle p, \mathsf{nl}(p)\rangle \rightarrow^*_{\mathfrak{S}} [\![p]\!]_\pi$$

Démonstration : Par induction sur la structure de p.

La preuve de correction de la traduction va alors se découper en deux étapes[4].

Tout d'abord une propriété assurant que si un système d'agents est stabilisé, alors il est capable de simuler n'importe quelle évolution du processus qu'il modélise, en ce sens que les éventuels agents de type `tmp` résultant d'une interaction peuvent se réduire en la version stabilisée du processus qu'ils mémorisent.

Proposition 1 *Soient p et S tel que $(p, S) \in \mathcal{R}$.*
- *Si $p \rightarrow p'$ alors il existe S' tel que $S \rightarrow S'$ et $(p', S') \in \mathcal{R}_{\mathfrak{S}}$*
- *Si $S \rightarrow S'$ alors il existe p' tel que $p \rightarrow p'$ et $(p', S') \in \mathcal{R}_{\mathfrak{S}}$*

Démonstration : Par définition $S = [\![q]\!]_\pi$ avec $q \equiv_\pi p$. Supposons $p \rightarrow p'$, on travaille par induction sur la preuve de $p \rightarrow p'$:

Si ($p = \bar{\mathrm{x}}\mathrm{y}.p_1 + q_1 \,|\, \mathrm{x}(\mathrm{z}).p_2 + q_2$) et ($p' = p_1 \,|\, p_2\{y/z\}$) : Puisque $q \equiv_\pi p$, par définition de $[\![\cdot]\!]_\pi$, il existe des interfaces partielles ϕ_1, ϕ_2, ψ_1 ,ψ_2 telles que, $[\![q]\!]_\pi = a_1 \,,\, a_2$ avec

$$\begin{aligned} a_1 &= \mathtt{proc}_{\mathtt{n}_1}\langle \phi_1, \mathrm{x}, \mathrm{E}, \mathrm{y}, p_1, \psi_1, \mathsf{nl}(\bar{\mathrm{x}}\mathrm{y}.p_1 + q_1)\rangle \\ a_2 &= \mathtt{proc}_{\mathtt{n}_2}\langle \phi_2, \mathrm{x}, \mathrm{R}, \mathrm{z}, p_2, \psi_2, \mathsf{nl}(\mathrm{x}(\mathrm{z}).p_2 + q_2)\rangle \end{aligned}$$

[4]Cette technique de preuve est à rapprocher de la bisimulation modulo introduite dans [Mil89a] et étudiée entre autres dans [San98]

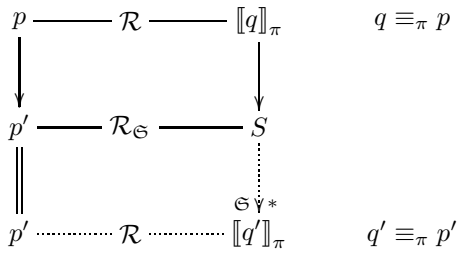


FIG. 2.7 – Schéma représentant la relation entre $\mathcal{R}$ et $\mathcal{R}_{\mathfrak{S}}$

La règle (com) peut donc s'appliquer : $[\![q]\!]_\pi \longrightarrow S' = a'_1 \,,\, a'_2$ avec

$$\begin{aligned} a'_1 &= \mathtt{tmp}\langle p_1, \mathsf{nl}(p_1)\rangle \\ a'_2 &= \mathtt{tmp}\langle p_2\{\mathrm{y/z}\} \,,\, \mathsf{nl}(p_2\{\mathrm{y/z}\})\rangle \end{aligned}$$

Par application du lemme 3, il vient que $a'_1 \,,\, a'_2 \rightarrow^*_{\mathfrak{S}} [\![p_1]\!]_\pi \,,\, [\![p_2\{\mathrm{y/z}\}]\!]_\pi = [\![p_1 \,|\, p_2\{\mathrm{y/z}\}]\!]_\pi = [\![p']\!]_\pi$ qui est bien de la forme $[\![q']\!]_\pi$ avec $q' \equiv_\pi p'$ ce qui prouve $(p', S') \in \mathcal{R}_{\mathfrak{S}}$.

Si $(p = \tau.p_1 + p_2)$ et $(p' = p_1)$: Arguments similaires.

Si $(p = p_1 \,|\, p_2)$ et $(p' = p'_1 \,|\, p_2)$ avec $(p_1 \rightarrow p'_1)$: Alors il existe q_1, q_2 tel que $[\![q]\!]_\pi = [\![q_1 \,|\, q_2]\!]_\pi = [\![q_1]\!]_\pi \,,\, [\![q_2]\!]_\pi$ avec $q_1 \equiv_\pi p_1$ et $q_2 \equiv_\pi p_2$. Par induction, il vient $[\![q_1]\!]_\pi \longrightarrow_{\mathfrak{R}} S'_1$ tel que $(p'_1, S'_1) \in \mathcal{R}_{\mathfrak{S}}$, c'est-à-dire qu'il existe $q'_1 \equiv_\pi p'_1$ tel que $S' \longrightarrow^*_{\mathfrak{S}} [\![q'_1]\!]_\pi$. Par application de (par), on a donc $S' \,,\, [\![q_2]\!]_\pi \longrightarrow^*_{\mathfrak{S}} [\![q'_1]\!]_\pi \,,\, [\![q_2]\!]_\pi = [\![q'_1 \,|\, q_2]\!]_\pi$ avec $q'_1 \,|\, q_2 \equiv_\pi p'_1 \,|\, p_2$.

Si $(p = (\nu\,\mathrm{x})\,p')$: Même chose avec la règle (restric)

Si $(p \equiv_\pi p_1)$ et $(p' \equiv_\pi p_2)$ avec $(p_1 \rightarrow p_2)$: Par transitivité de $\equiv_\pi$ il vient que $(p_1, [\![q]\!]_\pi) \in \mathcal{R}$. En utilisant l'hypothèse d'induction il vient $[\![q]\!]_\pi \longrightarrow_{\mathfrak{R}} S'$ tel que $(p_2, S') \in \mathcal{R}_{\mathfrak{S}}$. Donc il existe q' tel que $S' \longrightarrow^*_{\mathcal{R}} [\![q']\!]_\pi$ avec $q' \equiv_\pi p_2$. Une nouvelle fois, la transitivité de $\equiv_\pi$ nous assure que $q' \equiv_\pi p'$. On en déduit donc $(p', S') \in \mathcal{R}_{\mathfrak{S}}$.

L'autre partie de la proposition utilise le même type d'arguments. Supposons maintenant $[\![q]\!]_\pi \rightarrow S'$ et travaillons par induction sur la structure de $[\![q]\!]_\pi \rightarrow S'$. On remarque tout d'abord qu'étant donné la structure imposée de $[\![q]\!]_\pi$ seules deux règles de $\mathfrak{R}$ peuvent s'appliquer, à savoir (com) et (tau). Pour ces deux cas, on utilise des arguments similaires au premier cas de la preuve précédente. Les autres cas (group), (new) et (struct) sont une simple utilisation de l'hypothèse d'induction.

La seconde partie de la preuve de correction consiste à établir le résultat principal, à savoir que $\mathcal{R}_{\mathfrak{S}}$ est une bisimulation faible. On va évidemment s'appuyer pour cela sur la proposition précédente, mais il reste une difficulté

technique inhérente aux algèbres de processus en général à surmonter. Plusieurs règles peuvent s'appliquer en même temps sur un même système d'agents et peuvent donc donner lieu à différentes évolutions du système. Dans notre cas, il faut noter que les règles de $\mathfrak{S}$ s'appliquent de façon indépendante des autres règles. Ce qui se traduit par une propriété forte s'apparentant à la *propriété du diamant* :

Lemme 4 (Propriété du diamant pour $\mathfrak{S}$) *Soient S, S_1, S_2 trois systèmes d'agents tels que $S_1 \not\equiv S_2$. Soient $\mathfrak{s} \in \mathfrak{S}$ et $\mathfrak{r} \in \mathfrak{R}$ tels que $S \xrightarrow{\mathfrak{r}} S_1$ et $S \xrightarrow{\mathfrak{s}} S_2$. Alors il existe S' tel que $S_1 \xrightarrow{\mathfrak{s}} S'$ et $S_2 \xrightarrow{\mathfrak{r}} S'$.*

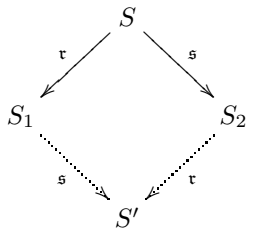


FIG. 2.8 – Propriété du diamant pour $\mathfrak{S}$

Démonstration : Quelle que soit la règle $\mathfrak{r}$ utilisée, la remarque 1 précédente nous assure qu'elle ne change par l'interface de l'agent modifié par $\mathfrak{s}$. De même, les règles de $\mathfrak{S}$ ne peuvent modifier les agents impliqués dans la règle $\mathfrak{r}$. On peut donc appliquer la règle $\mathfrak{s}$ sur S_1 et la règle $\mathfrak{r}$ sur S_2. Dans les deux cas, cela donne le même système d'agents.

Ce lemme, conjugué à la proposition 1, va nous permettre de démontrer la propriété suivante qui établit le principal résultat concernant $\mathcal{R}_{\mathfrak{S}}$:

Théoreme 1 *$\mathcal{R}_{\mathfrak{S}}$ est une bisimulation faible*

Démonstration : Soit $(p, S) \in \mathcal{R}_{\mathfrak{S}}$. Par définition de $\mathcal{R}_{\mathfrak{S}}$, il existe q tel que $S \longrightarrow^*_{\mathfrak{S}} [\![q]\!]_\pi$ et $(p, [\![q]\!]_\pi) \in \mathcal{R}$.

Supposons $p \to p'$: Alors l'application de la proposition 1 nous permet de conclure directement que $[\![q]\!]_\pi \to S'$ avec $(p', S') \in \mathcal{R}_{\mathfrak{S}}$.

Supposons $S \to S'$: On prouve l'existence de p' par induction sur le nombre de réductions de S vers $[\![q]\!]_\pi$. Le cas de base ($S = [\![q]\!]_\pi$) se réduit à une simple application de la proposition 1.

Pour le cas d'induction, deux cas se présentent en fonction de la règle $\mathfrak{r}$ utilisée pour faire évoluer S en S' et de la première règle utilisée dans $S \to^*_{\mathfrak{S}} [\![q]\!]_\pi$. Soit $\mathfrak{s} \in \mathfrak{S}$ cette dernière et S_1 le système obtenu.

Ou bien $\mathfrak{r} = \mathfrak{s}$, auquel cas il suffit de prendre $p' = p$ pour conclure la proposition.

Ou bien $\mathfrak{r} \neq \mathfrak{s}$ et alors on applique le lemme 4 pour trouver S_1' tel que $S_1 \xrightarrow{\mathfrak{r}} S_1'$ et $S' \xrightarrow{\mathfrak{s}} S_1'$. On conclut ensuite à l'aide de l'hypothèse d'induction

Ce théorème et le fait que $\mathcal{R} \subseteq \mathcal{R}_{\mathfrak{S}}$ nous permettent de conclure en établissant comme résultat la correction de la traduction proposée section 2.2.2 :

Corollaire 1 (Correction de $[\![\cdot]\!]_\pi$) *Pour tout processus p du π-calcul, p et $[\![p]\!]_\pi$ sont faiblement bisimilaires.*

2.3 Une version plus élémentaire de la communication

Il est maintenant nécessaire d'analyser la précédente traduction pour comprendre l'éclairage qu'elle apporte aux différents mécanismes qui sous-tendent les deux langages. On a vu que la somme pouvait être formulée sous la forme d'une somme de sites, que l'homologue d'un nom de canal dans le π-calcul est un nom porté par un site dans $g\kappa$-calcul et que le mécanisme du π-calcul a besoin d'une rupture de symétrie, comme le montre la règle (com). Ceci revient en réalité à faire une analyse syntaxique de la somme dans le π-calcul, en séparant le nom de canal utilisé pour la communication, le nom transmis, la polarité de la communication et la continuation du processus. Nous avons par ailleurs naturellement représenté la composition parallèle du π-calcul par celle fournie dans le $g\kappa$-calcul.

Le problème du codage présenté ici est qu'il reste une opération importante du π-calcul qui n'a pas été décomposée, à savoir celle relative à la communication. On s'est en effet simplifié la tâche en n'essayant pas de coder le mécanisme de substitution du π-calcul dans le passage de nom du langage cible. Nous n'avons finalement fourni qu'un interpréteur du π-calcul dans $g\kappa$-calcul pour lequel la substitution reste une opération extérieure. Tout comme le π-calcul, $g\kappa$-calcul ne passe donc pas simplement des noms mais applique des fonctions arbitrairement complexes (la substitution en l'occurrence) sur une valeur portée par un site ou n-uple de valeurs portées par un n-uple de sites. Autrement dit, il n'y a en réalité pas de passage de nom dans le précédent codage. Les règles (com) et (act) en sont d'ailleurs des témoins explicites puisque le nom transmis disparaît lors de la communication pour être mémorisé en tant que valeur et ne réapparaît sous la forme de nom que lors de l'application de la règle (act). Ce codage n'est donc pas très satisfaisant puisque la substitution est préservée comme une opération intégrale alors qu'on aimerait la coder dans un passage de nom que permet le langage d'arrivée.

D'autre part, le fait de s'autoriser à appliquer des fonctions arbitraires lors d'une communication n'est pas un trait de construction insignifiant puisqu'il définit en réalité un cadre bien plus riche que le langage minimal présenté en section 2.1. Le problème lié à l'utilisation d'une telle classe de langage supérieure est que l'on perd beaucoup de notre capacité d'analyse du système. Un deuxième point, plus pragmatique, plaide en faveur du langage minimal : les problèmes d'auto-assemblage et de modélisation biologiques qui vont nous intéresser dans les prochains chapitres ne requièrent pas une telle richesse d'expressivité.

On est donc en droit de se poser la question de savoir s'il existe un fragment du π-calcul qui puisse être représenté sans utiliser un tel mécanisme. Prenons par exemple le processus $q = \mathrm{x(z)}.q'$ dans lequel q' est une pure somme terminée, c'est-à-dire sans continuation, de la forme $\sum_i \alpha_i.\mathbf{O}$. Dans ce cas, nous savons précisément où est utilisé le nom z dans q' et il devient possible de le positionner au bon endroit dans son analogue dans $g\kappa$-calcul. Cette hypothèse induit en effet que le codage $[\![q']\!]_\pi$ est de la forme $\mathtt{proc_n}\langle\phi\rangle$, dans lequel le nom z apparaît explicitement dans l'interface comme un nom de $\mathcal{C}$. Il suffit, lors d'une communication avec un processus envoyant un nom sur le canal x, de remplacer z par le nom transmis dans cette même étape de communication.

Plus généralement, si on appelle *niveau* la profondeur à laquelle un nom apparaît dans l'arbre syntaxique d'un processus du π-calcul, il est possible de définir un fragment du π-calcul dans lequel tout nom reçu par un processus via un nom de canal situé à un niveau n n'utilise ce nom qu'au niveau $n+1$. Autrement dit, il n'y a qu'un seul niveau de différence entre un lieur et le substituande. Le processus $\mathrm{x(z)}.\bar{\mathrm{z}}\mathrm{x}.\mathbf{0}$ répond par exemple à cette contrainte alors que $\mathrm{x(z)}.\bar{\mathrm{x}}\mathrm{y}.\mathrm{z(x)}.\mathbf{0}$ non.

Afin de simplifier la description de ce fragment du π-calcul, nous travaillons désormais sur les formes normales des processus, c'est-à-dire uniquement les processus de la forme $(\tilde{\mathrm{x}}) \prod_i \sum_j \alpha_{ij}.p_{ij}$ et pour lesquels l'opérateur de restriction ν n'apparaît pas dans les sous-processus p_{ij}. Il est à noter que, par α-conversion, tout processus du π-calcul peut se ramener à un tel processus en forme normale.

Définition 12 (Réception transmise) *Une réception sera qualifiée de* transmise *si elle est de la forme*

$$\mathrm{x(z)}.\prod_i \sum_j \alpha_{ij}.p_{ij}$$

avec $\mathrm{z} \notin \mathsf{nl}(p_{ij})$ *pour tout* i *et* j.

Par extension, un processus sera dit *transmetteur* si toutes ses réceptions sont transmises. La définition précédente induit la possibilité d'extraire toutes

les occurrences du nom reçu en considérant simplement l'ensemble constitué des α_{ij}. Autrement dit, pour toute réception transmise, il est possible de déterminer un sous-ensemble d'indices dénotant les α_{ij} utilisant le nom reçu. Il est par ailleurs utile de raffiner ce sous-ensemble en fonction de l'emploi qui est fait du nom reçu, à savoir le fait d'être utilisé pour communiquer ou le fait d'être transmis. Cette distinction trouve son importance dans la traduction de la règle de communication que nous présentons plus bas. Formellement, la propriété peut s'énoncer ainsi :

Propriété 1 *Soit* $\mathrm{x(z)}.\prod_{i\in I}\sum_{j\in J_i}\alpha_{ij}.p_{ij}$ *une réception transmise. Pour tout* i *de* I *il existe deux sous-ensembles,* $\mathsf{ext}^1_{\mathrm{z}}(J_i)$ *et* $\mathsf{ext}^2_{\mathrm{z}}(J_i)$*, d'éléments de* J_i *tels que :*

$$j \in \mathsf{ext}^1_{\mathrm{z}}(J_i) \Longrightarrow \alpha_{ij} = \bar{\mathrm{z}}\mathrm{y} \vee \alpha_{ij} = \mathrm{z(y)}$$
$$j \in \mathsf{ext}^2_{\mathrm{z}}(J_i) \Longrightarrow \alpha_{ij} = \bar{\mathrm{y}}\mathrm{z}$$

Il nous reste maintenant à présenter la règle d'interaction correspondant à une communication entre deux processus transmetteurs. Elle est naturellement basée sur la traduction présentée dans la section 2.2. Soient $p = \sum_{i=1}^{n_1}\alpha_i.p_i$ et $q = \sum_{j=1}^{n_1}\beta_j.q_j$ deux processus transmetteurs tels que $\mathsf{i}_\pi(p) = \langle\phi, \mathrm{x}, \mathrm{E}, \mathrm{y}, p', \phi'\rangle$ et $\mathsf{i}_\pi(q) = \langle\psi, \mathrm{x}, \mathrm{R}, \mathrm{z}, q', \psi'\rangle$. Comme les processus sont ici sous forme normale, nous pouvons également écrire p' et q' sous la forme $\prod_{i\in I_p}\sum_{j\in J_i^p}\alpha_{ij}.p'_{ij}$ et $\prod_{i\in I_q}\sum_{j\in J_i^q}\beta_{ij}.q'_{ij}$. Enfin, pour tout indice i de I_q, on définit l'interface $\phi_i = \langle s_1, \ldots, s_n\rangle$ par :

$$\phi_i = \mathsf{i}_\pi\left(\sum_{j\in J_i^q}\beta_{ij}.q'_{ij}\right)$$

La communication entre les processus p et q a alors un effet direct sur les interfaces ϕ_i et elles seules. Il consiste dans le fait de remplacer le nom z par le nom y qui est transmis. Ceci peut aisément être exprimé syntaxiquement en utilisant la propriété 1. Pour chaque interface ϕ_i on définit l'interface $\phi'_i = \langle s'_1, \ldots, s'_n\rangle$ relative à la communication entre p et q par :

$$s'_j = \begin{cases} \mathrm{y} & \text{si } j = 4i \wedge i \in \mathsf{ext}^1_{\mathrm{z}}(J_i^q) \\ \mathrm{y} & \text{si } j = 4i+2 \wedge i \in \mathsf{ext}^2_{\mathrm{z}}(J_i^q) \\ s_j & \text{sinon} \end{cases}$$

La règle de communication peut alors s'écrire sans appliquer de fonctions annexes au calcul :

$$(\mathsf{com'})\ \frac{\mathtt{proc}_{\mathtt{n}_1}\langle\phi, \mathrm{x}, \mathrm{E}, \mathrm{y}, p', \phi', \mathsf{nl}(p)\rangle\ ,\ \mathtt{proc}_{\mathtt{n}_2}\langle\psi, \mathrm{x}, \mathrm{R}, \mathrm{z}, q', \psi', \mathsf{nl}(q)\rangle}{[\![p']\!]_\pi\ ,\ \prod_{i\in I_q}\mathtt{proc}_{\mathtt{n}_i}\langle\phi'_i, \mathsf{nl}(q)\rangle}$$

Il est donc possible d'exprimer directement le passage du nom dans le π-calcul comme un passage de nom sur les sites dans $g\kappa$-calcul lorsqu'on se restreint au fragment du π-calcul dans lequel un nom communiqué n'apparaît qu'en surface de la continuation du processus qui le reçoit. La substitution dans le cadre général n'est en réalité rien d'autre que ce mécanisme étendu au cas où on peut traverser les couches formées par les opérateurs parallèles et sommes du π-calcul.

Reste à savoir si ce fragment du π-calcul est suffisamment expressif. En particulier, la question de savoir s'il est possible de plonger le π-calcul en entier dans ce fragment reste ouverte et laissée à des travaux ultérieurs. Il semble cependant indispensable pour cela d'introduire des mécanismes de relais permettant à un processus qui reçoit un nouveau nom de le transmettre immédiatement et de ne le récupérer qu'au moment où le processus en a besoin pour communiquer. Ainsi, cherchant à définir un plongement $[\![\cdot]\!]_f$ du π-calcul dans le fragment du π-calcul que nous avons défini, le processus $\mathrm{x(z)}.p$ deviendrait alors le processus transmetteur

$$[\![\mathrm{x(z)}.p]\!]_f \;:=\; \mathrm{x(z)}.(\nu\, \mathrm{f_z})([\![p]\!]_f \mid \bar{\mathrm{f}}_\mathrm{z}\mathrm{z})$$

En terme de programmation cependant, ce fragment ne semble ni très riche, ni très proche d'une implémentation bas-niveau du mécanisme transactionnel du π-calcul. Habituellement, un environnement accompagne les programmes et permet de renommer *à la volée* les variables utilisées lorsque cela devient nécessaire. Une autre question qui se dessine consiste à savoir s'il ne serait pas possible d'étendre le codage présenté en section 2.2 afin d'obtenir directement une représentation de la substitution explicite à l'aide d'une telle notion d'environnement. Cette question, comme celle du plongement du π-calcul dans le fragment défini dans la présente section, reste ouverte pour le moment.

Chapitre 3

Auto-assemblage d'arbres

Sommaire

Nous développons dans ce chapitre un exemple simple, l'auto-assemblage d'arbres, illustrant l'utilisation de notre langage et montrant son adaptation à ce type de problème. Il s'agit d'établir un algorithme qui réalise l'assemblage d'arbres de manière *hautement distribuée*. Deux contraintes sont intimement liées à cette notion. Non seulement il est nécessaire, afin de rester proche d'une véritable implémentation, que les communications permettant de faire évoluer la structure s'effectuent entre deux agents au plus. Mais il faut surtout s'assurer que les décisions prises par les agents sont décentralisées et qu'aucun rôle particulier n'a été attribué arbitrairement à un agent. Autrement dit, à l'état initial, tous les agents doivent avoir les mêmes capacités d'interaction avec leur environnement.

Après avoir présenté intuitivement le rapport entre agents et graphes, la section 3.1 s'attachera à établir formellement la relation entre ces deux notions dans sa formulation la plus générale. Nous verrons ensuite, dans la section 3.2, que la syntaxe de notre langage permet à la fois d'énoncer le problème d'auto-assemblage et de présenter sa solution dans le même cadre théorique. Cela simplifiera substantiellement la formulation de la correction de l'algorithme dans la section 3.3. Enfin, nous terminerons ce chapitre par une discussion, section 3.4, des résultats établis. La critique portera à la fois sur l'approche adoptée et sur les choix qui ont été faits au niveau de la spécification, ainsi que sur leurs répercutions au niveau de l'algorithme.

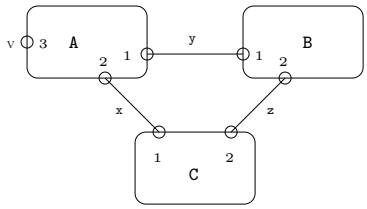
3.1 Spécification

3.1.1 Arbres, graphes et graphes bipartites

Dans la tradition du π-calcul, les noms modélisent des canaux de communication et constituent à ce titre un lien entre deux processus. Si nous regardons l'ensemble de ces liens, nous obtenons alors une représentation des connexions en terme de graphes. Un raffinement dû à la syntaxe permet de préciser la localisation de ces connexions en spécifiant le site de l'interface concernée. Considérons par exemple le réseau[1] suivant :

$$\mathtt{A}\langle \mathrm{x}, \mathrm{y}, \mathrm{v} \rangle \ , \ \mathtt{B}\langle \mathrm{y}, \mathrm{z} \rangle \ , \ \mathtt{C}\langle \mathrm{x}, \mathrm{z} \rangle$$

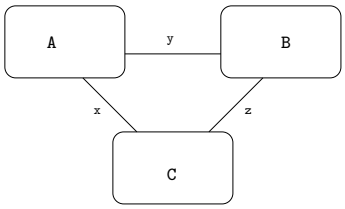
Une façon intuitive de représenter graphiquement ce réseau consiste à modéliser les agents par des boîtes et chaque site d'une interface comme un point d'ancrage présent à la frontière de la boîte. Un arc relie alors deux points d'ancrage distincts si les sites qu'ils représentent partagent un nom commun. Ce nom peut d'ailleurs servir d'étiquette à l'arc. Dans le cas de notre exemple, cela donne :



Cependant, cette représentation est un peu trop riche en ce qui concerne l'auto-assemblage puisque les schémas que l'on cherche à assembler sont de simples graphes ou arbres pour lesquels les liaisons ne sont pas localisées. Il semble donc naturel de s'abstraire un peu de cette représentation pour ne

[1] Dans ce chapitre, ainsi que dans le chapitre suivant, nous utiliserons la dénomination de réseau, terme proche de la thématique de l'auto-assemblage, pour désigner les systèmes d'agents.

considérer que les ensembles de noms partagés par les agents. Cela donne une représentation simplifiée que nous appellerons *graphe de connexions* d'un réseau d'agents. Pour le cas précédent, cela donne le graphe :



C'est cette représentation-là que nous allons utiliser à partir de maintenant pour réaliser les différents auto-assemblages, qu'il s'agisse d'arbres ou de graphes. Cependant, on doit noter que la syntaxe fournit en réalité une représentation plus riche, puisque rien ne restreint, a priori, les noms à n'être partagés que par deux agents. Ainsi le réseau :

$$\mathtt{A}\langle \mathrm{x} \rangle \ , \ \mathtt{B}\langle \mathrm{x} \rangle \ , \ \mathtt{C}\langle \mathrm{x} \rangle$$

ne peut pas être représenté sous la forme d'un graphe simple. La meilleure façon de décrire un tel réseau est donc de définir un nouveau nœud pour chaque nom de connexion et d'établir un arc entre un nœud d'agent et un nœud de connexion, chaque fois qu'un nom apparaît dans l'interface de l'agent. Ce qui donne, pour le réseau précédent :

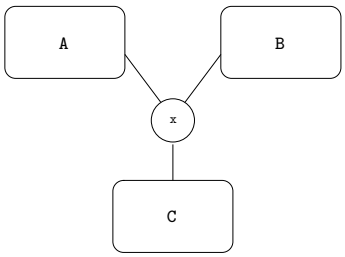


FIG. 3.1 – Une représentation visuelle de $\mathtt{A}\langle \mathrm{x} \rangle$, $\mathtt{B}\langle \mathrm{x} \rangle$, $\mathtt{C}\langle \mathrm{x} \rangle$

Cette représentation, qui correspond en fait à celle des *graphes bipartites* (formalisme équivalent aux hypergraphes), peut se voir comme un graphe dont on a partitionné l'ensemble des nœuds.

Définition 13 (Graphes bipartites) *Un graphe (V, E) est dit bipartite s'il existe V_1, V_2 tels que $V_1 \cup V_2 = V$ et $V_1 \cap V_2 = \emptyset$ et pour tout couple $(x, y) \in E$ on a $x \in V_1$ et $y \in V_2$.*

Par la suite on notera de tels graphes (V_1, V_2, E) afin de les distinguer des graphes simples.

Ainsi, le réseau précédent peut très bien se représenter formellement par un graphe (bipartite) des connexions en prenant $V_1 = \{\mathtt{A}, \mathtt{B}, \mathtt{C}\}$, $V_2 = \{\mathrm{x}\}$ et $E = \{(\mathtt{A}, \mathrm{x}), (\mathtt{B}, \mathrm{x}), (\mathtt{C}, \mathrm{x})\}$.Intéressons-nous maintenant à établir le lien formel entre les réseaux d'agents et les graphes bipartites.

Des agents aux graphes et vice-versa. En suivant l'intuition que l'on vient de présenter, on peut définir la fonction $[\![\cdot]\!]_{\mathsf{gb}}$ des réseaux dans les graphes bipartites.

Définition 14 (Des agents aux graphes) *Soit* $R = \mathtt{t_1}\langle\phi_1\rangle\ ,\ \ldots\ ,\ \mathtt{t_n}\langle\phi_n\rangle$ *un réseau d'agents. On définit la fonction* $[\![\cdot]\!]_{\mathsf{gb}}$ *traduisant* R *en un graphe bipartite* (V_1, V_2, E) *par :*

- $V_1 = \{\mathtt{t_i}\}$
- $V_2 = \cup_i(\phi_i \cap \mathcal{C})$
- $E = \{(\mathtt{t_i}, \mathrm{y}) \mid \mathrm{y} \in \phi_i\}$

Inversement, tout graphe bipartite (V_1, V_2, E) peut être représenté sous la forme d'un réseau d'agents, où chacun des éléments de V_1 est considéré comme un agent et où chaque nœud de V_2 est associé à un nom de canal. Comme précédemment, les arcs présents dans E sont représentés par le partage de noms. Par commodité, on se donne un nouveau type `noeud` pour désigner ces agents. L'interface correspondante ne contient qu'un seul site qui répertorie l'ensemble des nœuds de V_2 possédant un arc avec le nœud de V_1 concerné.

Définition 15 (Des graphes aux agents) *Soit* (V_1, V_2, E) *un graphe bipartite avec* $V_1 = \{v_1, \ldots, v_n\}$. *Soit une fonction injective* $\mathsf{arc}(\cdot) : V_2 \mapsto \mathcal{C}$. *On définit les ensembles suivants :*

- $\forall v \in V_1$ *on définit* $\mathsf{Noms}(v) = \{\mathrm{x} \in \mathcal{C} \mid (v, v') \in E \wedge \mathrm{x} = \mathsf{arc}(v')\}$
- $\mathsf{Noms} = \cup_{v \in V_1} \mathsf{Noms}(v)$

On définit alors le réseau $R = [\![(V_1, V_2, E)]\!]_{\mathsf{col}}$ *correspondant au graphe bipartite* (V_1, V_2, E) *par :*

$$R = (\nu\, \mathsf{Noms})\, (\mathtt{noeud}\langle\mathsf{Noms}(v_1)\rangle\ ,\ \ldots\ ,\ \mathtt{noeud}\langle\mathsf{Noms}(v_n)\rangle)$$

3.1.2 Les arbres cohérents

Une des caractéristiques importantes de notre approche consiste à pouvoir spécifier le problème dans le langage même dans lequel on va formuler la solution. Alors que l'approche traditionnelle pour ce type d'étude demande de définir un système de transition pour la spécification et d'en utiliser un second pour énoncer la solution, notre langage permet de plonger les deux systèmes dans le même cadre théorique, ce qui facilite en partie la preuve de correction.

Soit N un ensemble de nœuds pour lequel on associe une fonction $\mathsf{dg}(\cdot) : N \mapsto \mathbb{N}$ précisant, pour chaque élément de N, le nombre de nœuds auquel il

peut être connecté. On représentera les arbres sous la forme

$$a \quad ::= \quad (r, \{a_1, \ldots, a_n\})$$

où $r \in N$ est la racine et $n \in \mathbb{N}$ est le nombre de branches partant du nœud r.

Ainsi, l'arbre le plus simple sera représenté par $(r, \emptyset)$, ce qu'on abrégera en r. Un autre exemple, un peu moins trivial, pourrait être $(a, \{b, c\})$ représentant l'arbre constitué de trois nœuds (a, b et c) pour lequel le nœud a possède deux fils b et c. Bien sûr, comme la notation le laisse supposer, on ne différencie pas ici l'arbre $(a, \{b, c\})$ de l'arbre $(a, \{c, b\})$. Traditionnellement, on nomme *feuille* tout nœud ne possédant pas de fils.

Un arbre a est dit *cohérent* si tous les nœuds de a ont leur degré respectif correspondant à la valeur de la fonction de degré. Toute feuille a ainsi une arité inférieure ou égale à 1 (0 dans le cas où c'est aussi la racine de l'arbre). Par exemple, si $\mathsf{dg}(a) = 2$ et $\mathsf{dg}(b) = \mathsf{dg}(c) = 1$, alors l'arbre $(a, \{b, c\})$ est cohérent, alors que l'arbre $(a, \{(b, \{c\})\})$ ne l'est pas. De même, un arbre constitué d'un seul nœud racine a sera cohérent si et seulement si le degré de a est 0. On dénote par $\mathsf{nd}(a)$ l'ensemble des nœuds de l'arbre a.

Les arbres en tant qu'agents. Il faut tout d'abord établir une représentation des arbres cohérents. On va pour cela s'aider de la fonction $[\![\cdot]\!]_{\mathsf{col}}$ mais en raffinant légèrement la version des agents représentant les nœuds puisque l'on veut ici se souvenir aussi de la liaison père/fils. On se donne donc un type $\mathtt{a}$ pour chaque nœud a de N avec $\mathtt{n}(\mathtt{a}) = 2$. Le premier site va justement servir à mémoriser cette information en précisant quel nom relie un agent à son père. Dans le cas où l'agent représente la racine de l'arbre, ce site contiendra la valeur $\star$. D'autre part, si l'agent n'est connecté à aucun autre agent, ce site aura la valeur I. Le cas où l'agent a un degré égal à 0 nous permet, entre autre, de différencier formellement cet agent à l'état initial $\mathtt{a}\langle \mathrm{I}, \emptyset \rangle$ de l'arbre cohérent de nœud unique $\mathtt{a}\langle \star, \emptyset \rangle$. Le second site, comme pour la fonction $[\![\cdot]\!]_{\mathsf{col}}$, présente l'ensemble des noms de liaisons.

Soit t un arbre cohérent, on définit le réseau d'agents $[\![t]\!]_{\mathsf{a}}$ correspondant à t d'une manière similaire à la définition 14 de $[\![\cdot]\!]_{\mathsf{col}}$. Mais plutôt que de traduire chaque nœud en un agent de type `noeud`, on le traduit en l'agent de type $\mathtt{a}$ correspondant. Pour cela, il nous faut en particulier définir un nom de canal pour chacun des arcs de t. Il est commode d'utiliser un ensemble de noms de canaux paramétrés par des suites. On note ϵ la suite vide et sn la suite composée par la suite s suivi du nombre n. Soit s une suite de nombres, on utilise la notation x_s pour représenter le nom x paramétré par la suite s.

Définition 16 *Soit t un arbre cohérent. On définit parallèlement une famille de fonctions $\mathsf{Noms}(s, t)$ et $\mathsf{Ag}(s, t)$ paramétrées par une suite s en fonction de*

la structure de t :

$$\mathsf{Noms}(s,\emptyset) = \emptyset$$
$$\mathsf{Noms}(s,(a,\{t_1,\ldots,t_n\})) = \bigcup_{1\leq i\leq n}\{\mathrm{x}_{si}\} \cup \bigcup_{1\leq i\leq n}\mathsf{Noms}(si,t_i)$$
$$\mathsf{Ag}(s,\emptyset) = \emptyset$$
$$\mathsf{Ag}(s,(a,\{t_1,\ldots,t_n\})) = \mathtt{a}\langle \mathrm{x}_s,\{\mathrm{x}_{s1},\ldots,\mathrm{x}_{sn}\}\rangle \ ,\ \mathsf{Ag}(s1,t_1)\ ,\ \ldots\ ,\ \mathsf{Ag}(sn,t_n)$$

Soit $t = (a,\{t_1,\ldots,t_n\})$ un arbre cohérent. On définit alors l'ensemble des noms de canaux par $\mathsf{Noms} = \mathsf{Noms}(\epsilon,t)$ *et le réseau représentant l'arbre t par*

$$[\![t]\!]_{\mathtt{a}} = (\nu\,\mathsf{Noms})\,(\mathtt{a}\langle\star,\{\mathrm{x}_1\ldots,\mathrm{x}_n\}\rangle\ ,\ \mathsf{Ag}(1,t_1)\ ,\ \ldots\ ,\ \mathsf{Ag}(n,t_n))$$

Pour reprendre l'exemple précédent de l'arbre cohérent $t = (a,\{b,c\})$ dans le cas où $\mathsf{dg}(a) = 2$ et $\mathsf{dg}(b) = \mathsf{dg}(c) = 1$, alors sa traduction en terme d'agents sera :

$$[\![t]\!]_{\mathtt{a}} = (\nu\,\mathrm{x}_1,\mathrm{x}_2)\,(\mathtt{a}\langle\star,\{\mathrm{x}_1\mathrm{x}_2\}\rangle\ ,\ \mathsf{Ag}(1,(b))\ ,\ \mathsf{Ag}(2,c))$$

avec

$$\begin{array}{rclcrcl}\mathsf{Ag}(1,(b)) & = & \mathtt{b}\langle\mathrm{x}_1,\emptyset\rangle\ ,\ \mathsf{Ag}(11,\emptyset) & = & \mathtt{b}\langle\mathrm{x}_1,\emptyset\rangle\ ,\ \emptyset & = & \mathtt{b}\langle\mathrm{x}_1,\emptyset\rangle\\ \mathsf{Ag}(2,(c)) & = & \mathtt{c}\langle\mathrm{x}_2,\emptyset\rangle\ ,\ \mathsf{Ag}(21,\emptyset)) & = & \mathtt{c}\langle\mathrm{x}_2,\emptyset\rangle\ ,\ \emptyset & = & \mathtt{c}\langle\mathrm{x}_2,\emptyset\rangle\end{array}$$

Spécification. Le but ici est d'assembler n'importe quel arbre cohérent possédant n nœuds à partir de n agents déconnectés, c'est-à-dire de la forme $\mathtt{a}\langle\mathrm{I},\emptyset\rangle$. On se sert ensuite de la fonction $[\![\cdot]\!]_{\mathtt{a}}$ pour décrire la spécification de notre système de départ. Pour chaque arbre cohérent t tel que $Nd(t) = \{a_1,\ldots,a_n\}$ on ajoute la règle :

$$(\mathtt{t})\ \frac{\mathtt{a_1}\langle\mathrm{I},\emptyset\rangle\ ,\ \ldots\ ,\ \mathtt{a_n}\langle\mathrm{I},\emptyset\rangle}{[\![t]\!]_{\mathtt{a}}}$$

Tout ceci nous permet de définir un système dans notre langage qui servira de spécification.

Définition 17 (Spécification des assemblages d'arbres) *Soit N un ensemble de nœuds avec une fonction de degré* $\mathsf{dg}(\cdot)$*. On définit le système de spécification* $\mathtt{SPEC}_{\mathtt{A}}$ *par la donnée du n-uplet* $(\mathcal{T},\mathfrak{n},\mathcal{V},\mathcal{C},\mathfrak{R},S_0)$ *vérifiant :*

- $\mathcal{T} = N$
- $\forall \mathtt{t}\in\mathcal{T},\ \mathfrak{n}(\mathtt{t}) = 2$
- $\mathcal{V} = \{\mathrm{I},\star\}$
- $\mathfrak{R} = \{(t) \mid t \textit{ est cohérent }\}$

– $S_0 = \mathtt{a_1}\langle \textsc{i}, \emptyset\rangle \ , \ \ldots \ , \ \mathtt{a_n}\langle \textsc{i}, \emptyset\rangle$

Il nous reste maintenant à déterminer un second système, équivalent à celui-ci mais dans lequel les règles d'interaction sont strictement binaires. C'est ce à quoi s'attache la prochaine section.

3.2 L'assemblage

Bien sûr, réussir à assembler de tels arbres de façon décentralisée demande de raffiner en partie l'interface des agents afin de structurer le peu d'informations nécessaires pour atteindre l'assemblage final. Puisque l'assemblage ne se fait plus de manière atomique, cela signifie que la construction de l'arbre va se faire de façon croissante en « poussant » à partir des feuilles. Il y a donc des étapes temporaires pour lesquelles des parties de l'arbre ont fini d'être assemblées et d'autres pour lesquelles les agents attendent encore des connexions. Pour gérer toutes ces situations, nous allons découper l'interface d'un agent en trois parties. Le premier site est destiné, comme précédemment, à mémoriser le nom le liant à son père. Les deuxième et troisième sites stockent les ensembles de noms représentant les liaisons avec les autres agents voisins dans l'arbre. La différence entre les deux sites réside dans le fait que le second (que l'on dénote par T) n'indique que les noms liant l'agent à un sous-arbre dont l'assemblage est terminé. Le troisième site (dénoté par A) mémorise, quant à lui, les noms restants, ce qui signifie que l'assemblage de ces branches est en attente de la terminaison.

3.2.1 Une construction simple

Nous allons présenter un algorithme simple assurant l'assemblage décentralisé de n'importe quel arbre cohérent. La première des règles de cet algorithme permet à un agent inactif (c'est-à-dire dont le premier site présente la valeur i) de décider d'initier une construction. Cet agent devient donc la racine du futur arbre construit, ce qui se traduit par la règle :

$$(\mathsf{init})\ \frac{\mathtt{a}\langle \textsc{i}, \emptyset, \emptyset\rangle}{\mathtt{a}\langle \star, \emptyset, \emptyset\rangle}$$

Vient ensuite la règle de recrutement permettant à un agent (racine de l'arbre ou non) de recruter un agent inactif s'il lui reste au moins une connexion libre (en accord avec la fonction $\mathsf{dg}(\cdot)$). Pour tout nœud a_1 et a_2 on ajoute la règle :

$$(\mathsf{recrut})\ \frac{\mathtt{a_1}\langle p, T, A\rangle \ , \ \mathtt{a_2}\langle \textsc{i}, \emptyset, \emptyset\rangle}{(\nu\, \mathrm{x})\, \mathtt{a_1}\langle p, T, A \cup \{\mathrm{x}\}\rangle \ , \ \mathtt{a_2}\langle \mathrm{x}, \emptyset, \{\mathrm{x}\}\rangle} \qquad \text{si} \ \begin{cases} \mathsf{dg}(\mathtt{a_1}) \leq |\,T\,| + |\,A\,| \\ \mathsf{dg}(\mathtt{a_2}) > 0 \end{cases}$$

La valeur du site p de $\mathtt{a}$ n'est pas spécifiée. L'agent $\mathtt{a}$ peut donc être soit l'initiateur de l'assemblage (on peut ainsi le considérer comme la racine naturelle du futur arbre assemblé), soit être un agent précédemment recruté qui sélectionne à son tour un agent inactif pour poursuivre la construction.

Remarque 2 *La condition d'inactivité posée sur le second agent assure que l'on ne crée pas de connexion entre deux agents appartenant déjà à un même arbre.*

Lorsqu'un agent réussit à recruter le nombre d'agents correspondant à son degré de connexion, il attend simplement que tous ses fils aient terminé le sous-assemblage dont ils sont devenus responsables. Une fois cette opération terminée, il communique à son père cette information. Le problème consiste en ce que la contrainte de binarité sur les règles nous empêche de regarder l'ensemble du réseau pour savoir si la sous-branche en question est assemblée ou non. Il nous faut donc déterminer un critère simple, à partir de l'état même d'un agent, capable de caractériser cette propriété. C'est évidemment à ce moment-là que la structure que l'on a choisie pour mémoriser les noms de canaux se révèle pertinente. Un agent aura terminé l'assemblage de l'arbre dont il est le nœud le plus élevé lorsque tous les noms de liaison feront partie de l'ensemble T, sauf le nom le liant à son père. On peut alors officialiser le fait que pour cet agent, l'assemblage est terminé. Ce que l'on représente par le transfert d'un nom appartenant au site A au site T :

$$(\mathsf{term})\ \frac{\mathtt{a}_1\langle p, T_1, A \cup \{\mathrm{x}\}\rangle\ ,\ \mathtt{a}_2\langle \mathrm{x}, T_2, \{\mathrm{x}\}\rangle}{\mathtt{a}_1\langle p, T_1 \cup \{\mathrm{x}\}, A\rangle\ ,\ \mathtt{a}_2\langle \mathrm{x}, T_2 \cup \{\mathrm{x}\}, \emptyset\rangle} \qquad \text{si } \mathsf{dg}(\mathtt{a}_2) = |\,T_2\,| - 1$$

Alors qu'il y avait un flux de recrutement de la racine aux feuilles, on observe alors un flux inverse, des feuilles vers la racine, pour communiquer à celle-ci que l'assemblage est terminé. Remarquons par ailleurs que, là encore, la valeur du site p du père n'est pas précisée. Ce pourrait être la racine comme un simple nœud de l'arbre.

Une dernière règle, similaire à la précédente, intervient lorsque le dernier agent de l'arbre (donc la racine) a été informé du fait que toutes les branches avaient été assemblées. Cet agent-racine se caractérise par une unique forme $\mathtt{a}\langle\star, T, \emptyset\rangle$ avec $|\,T\,| = \mathsf{dg}(a)$. On peut alors considérer que l'arbre en entier est assemblé, ce qui se traduit par la règle unaire :

$$(\mathsf{racine})\ \frac{\mathtt{a}\langle\star, T, \emptyset\rangle}{\mathtt{a}\langle \mathrm{T}, T, \emptyset\rangle} \qquad \text{si } |\,T\,| = \mathsf{dg}(a)$$

où T est la valeur signifiant quel l'agent est la racine d'un arbre terminé.

Exemple. Il est peut-être utile de présenter un exemple illustrant l'agencement de ces règles. Supposons que le système initial soit S_0 constitué de quatre agents, tous libres, dont un agent de type a, deux agents de type b et un agent de type c, ayant respectivement les degrés 2, 1 et 3.

$$S_0 = \mathtt{a}\langle \mathrm{I}, \emptyset, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{I}, \emptyset, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{I}, \emptyset, \emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{I}, \emptyset, \emptyset\rangle$$

Une évolution possible (et réussie) du système S_0 est donnée par la figure 3.2. Cette succession de synchronisations est la variante décentralisée de

$$\begin{aligned}
S_0 &\xrightarrow{(\mathsf{init})} \mathtt{a}\langle \star, \emptyset, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{I}, \emptyset, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{I}, \emptyset, \emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{I}, \emptyset, \emptyset\rangle && (1)\\
&\xrightarrow{(\mathsf{recrut})} (\nu\, \mathrm{x})\, \mathtt{a}\langle \star, \emptyset, \{\mathrm{x}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x}, \emptyset, \{\mathrm{x}\}\rangle \ , \ \mathtt{b}\langle \mathrm{I}, \emptyset, \emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{I}, \emptyset, \emptyset\rangle && (2)\\
&\xrightarrow{(\mathsf{recrut})} (\nu\, \mathrm{xy})\, \mathtt{a}\langle \star, \emptyset, \{\mathrm{x}, \mathrm{y}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x}, \emptyset, \{\mathrm{x}\}\rangle \ , \ \mathtt{b}\langle \mathrm{y}, \emptyset, \{\mathrm{y}\}\rangle \ , \ \mathtt{c}\langle \mathrm{I}, \emptyset, \emptyset\rangle && (3)\\
&\xrightarrow{(\mathsf{term})} (\nu\, \mathrm{xy})\, \mathtt{a}\langle \star, \{\mathrm{x}\}, \{\mathrm{y}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x}, \{\mathrm{x}\}, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{y}, \emptyset\, \{\mathrm{y}\}\rangle \ , \ \mathtt{c}\langle \mathrm{I}, \emptyset, \emptyset\rangle && (4)\\
&\xrightarrow{(\mathsf{term})} (\nu\, \mathrm{xy})\, \mathtt{a}\langle \star, \{\mathrm{x}, \mathrm{y}\}, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{x}, \{\mathrm{x}\}, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{y}, \{\mathrm{y}\}, \emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{I}, \emptyset, \emptyset\rangle && (5)\\
&\xrightarrow{(\mathsf{racine})} (\nu\, \mathrm{xy})\, \mathtt{a}\langle \mathrm{T}, \{\mathrm{x}, \mathrm{y}\}, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{x}, \{\mathrm{x}\}, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{y}, \{\mathrm{y}\}, \emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{I}, \emptyset, \emptyset\rangle && (6)
\end{aligned}$$

FIG. 3.2 – Exemple d'auto-assemblage réussi d'arbres

l'assemblage de l'arbre $(a, (b, b))$ c'est-à-dire de la règle d'assemblage de haut niveau :

$$(\mathtt{a},(\mathtt{b},\mathtt{b}))\ \frac{\mathtt{a}\langle \mathrm{I}, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{I}, \emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{I}, \emptyset\rangle}{(\nu\, \mathrm{xy})\, \mathtt{a}\langle \star, \{\mathrm{x}, \mathrm{y}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x}, \{\mathrm{x}\}\rangle \ , \ \mathtt{b}\langle \mathrm{y}, \{\mathrm{y}\}\rangle}$$

Bien sûr tous les assemblages ne suivent pas cette succession stricte des événements. L'ordre dans lequel parviennent au père les informations relatives à l'assemblage des différentes branches est totalement indépendant et peut être permuté. Les étapes (4) et (5) auraient pu être inversées ou encore l'étape (4) aurait pu avoir lieu avant l'étape (3).

3.2.2 Sortir des impasses

Cependant, il n'est pas du tout certain que l'assemblage suive un chemin aussi facile que le précédent. Le fait de décentraliser les choix implique en contrepartie un risque de blocage en fonction des recrutements des agents libres. Dans l'exemple précédent, l'agent c disponible reste totalement inactif. Il se pourrait très bien pourtant qu'à l'étape (2) ce soit justement cet agent qui soit recruté par l'agent de type a. Cela conduirait alors à un état du système dans lequel plus aucune règle n'est applicable, comme le montre l'évolution du système présenté dans la figure 3.3.

$$
\begin{array}{lll}
S_0 \xrightarrow{(\mathsf{init})} & \mathtt{a}\langle\star,\emptyset,\emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{I},\emptyset,\emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{I},\emptyset,\emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{I},\emptyset,\emptyset\rangle & (1) \\
\xrightarrow{(\mathsf{recrut})} & (\nu\,\mathrm{x})\,\mathtt{a}\langle\star,\emptyset,\{\mathrm{x}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x},\emptyset,\{\mathrm{x}\}\rangle \ , \ \mathtt{b}\langle \mathrm{I},\emptyset,\emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{I},\emptyset,\emptyset\rangle & (2) \\
\xrightarrow{(\mathsf{recrut})} & (\nu\,\mathrm{xy})\,\mathtt{a}\langle\star,\emptyset,\{\mathrm{x},\mathrm{y}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x},\emptyset,\{\mathrm{x}\}\rangle \ , \ \mathtt{b}\langle \mathrm{I},\emptyset,\emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{y},\emptyset,\{\mathrm{y}\}\rangle & (3') \\
\xrightarrow{(\mathsf{recrut})} & (\nu\,\mathrm{xyz})\,\mathtt{a}\langle\star,\emptyset,\{\mathrm{x},\mathrm{y}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x},\emptyset,\{\mathrm{x}\}\rangle \ , \ \mathtt{b}\langle \mathrm{z},\emptyset,\{\mathrm{z}\}\rangle \ , \ \mathtt{c}\langle \mathrm{y},\emptyset,\{\mathrm{z},\mathrm{y}\}\rangle & (4') \\
\xrightarrow{(\mathsf{term})} & (\nu\,\mathrm{xyz})\,\mathtt{a}\langle\star,\{\mathrm{x}\},\{\mathrm{y}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x},\{\mathrm{x}\},\emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{z},\emptyset,\{\mathrm{z}\}\rangle \ , \ \mathtt{c}\langle \mathrm{y},\emptyset,\{\mathrm{z},\mathrm{y}\}\rangle & (5') \\
\xrightarrow{(\mathsf{term})} & (\nu\,\mathrm{xyz})\,\mathtt{a}\langle\star,\{\mathrm{x}\},\{\mathrm{y}\}\rangle \ , \ \mathtt{b}\langle \mathrm{x},\{\mathrm{x}\},\emptyset\rangle \ , \ \mathtt{b}\langle \mathrm{z},\{\mathrm{z}\},\emptyset\rangle \ , \ \mathtt{c}\langle \mathrm{y},\{\mathrm{z}\},\{\mathrm{y}\}\rangle & (6')
\end{array}
$$

FIG. 3.3 – Exemple d'évolution conduisant à une impasse

Le système est bloqué après la dernière réduction puisque l'agent c a besoin d'un autre agent libre pour conclure.

Réversibilité. Une solution raisonnable dans un cas comme celui-ci consiste à supprimer les connexions qui ont été créées afin de tenter de repartir ensuite en avant, en espérant que la même succession d'événements ne se reproduira pas ; ce qui se concrétise dans notre système par le fait de rendre les règles réversibles. Ces nouvelles règles constituent un complément nécessaire à la partie *constructive* de l'algorithme décrit par les règles (init), (recrut), (term) et (racine). Une précaution doit cependant être prise : lorsqu'un assemblage s'est terminé avec succès, il faut s'assurer qu'aucun agent ne pourra prendre la décision de le défaire. En particulier, la règle (racine) finalisant l'assemblage ne doit en aucun cas être réversible.

Alors que la confirmation de l'assemblage des sous-arbres présentait un mouvement des feuilles vers la racine, le flux de déconnexion va s'effectuer cette fois-ci de la racine (ou du niveau le plus proche de la racine possible) vers les feuilles. En effet, les feuilles, pour prendre le cas extrême, n'ont aucun moyen de savoir si l'assemblage s'est terminé avec succès ou si l'une des branches se trouve enfermée dans une impasse. Mais cette remarque vaut en réalité pour tout nœud ayant confirmé à son père l'assemblage de la branche dont il est lui-même le père. C'est donc lorsque l'on va rendre la règle (term) réversible qu'il va falloir prendre des précautions :

$$
(\mathsf{rev\text{-}term\ 1})\ \frac{\mathtt{a}_1\langle \mathrm{y}, T_1 \cup \{\mathrm{x}\}, A \cup \{\mathrm{y}\}\rangle \ , \ \mathtt{a}_2\langle \mathrm{x}, T_2 \cup \{\mathrm{x}\}, \emptyset\rangle}{\mathtt{a}_1\langle \mathrm{y}, T_1, A \cup \{\mathrm{x},\mathrm{y}\}\rangle \ , \ \mathtt{a}_2\langle \mathrm{x}, T_2, \{\mathrm{x}\}\rangle}
$$

Le fait de préciser que le premier site du père $\mathtt{a}_1$ est un nom y et se retrouve dans l'ensemble T_1 assure que cet agent n'a pas transmis de confirmation à son

propre père ou que le système est revenu sur cette décision en utilisant une des règles réversibles qu'on est en train de décrire. Dans le cas où cet agent est la racine de l'arbre, la condition concernant le premier site est un peu différente. On demande alors que ce soit la valeur $\star$, s'assurant ainsi qu'aucune finalisation n'a été faite (la valeur aurait alors été T).

$$(\text{rev-term 2})\ \frac{\mathtt{a}_1\langle \star, T_1 \cup \{\mathrm{x}\}, A\rangle\ ,\ \mathtt{a}_2\langle \mathrm{x}, T_2 \cup \{\mathrm{x}\}, \emptyset\rangle}{\mathtt{a}\langle \star, T_1, A \cup \{\mathrm{x}\}\rangle\ ,\ \mathtt{b}\langle \mathrm{x}, T_2, \{\mathrm{x}\}\rangle}$$

Ainsi le flux s'effectue bien globalement de la racine vers les feuilles. Ensuite la déconnexion proprement dite peut avoir lieu, prenant directement les règles (recrut) et (init) à contresens :

$$(\text{rev-recrut})\ \frac{\mathtt{a}_1\langle p, T, A \cup \{\mathrm{x}\}\rangle\ ,\ \mathtt{a}_2\langle \mathrm{x}, \emptyset, \{\mathrm{x}\}\rangle}{\mathtt{a}_1\langle p, T, A\rangle\ ,\ \mathtt{a}_2\langle \mathrm{I}, \emptyset, \emptyset\rangle} \qquad \frac{\mathtt{a}\langle \star, \emptyset, \emptyset\rangle}{\mathtt{a}\langle \mathrm{I}, \emptyset, \emptyset\rangle}\ (\text{rev-init})$$

On notera REV l'ensemble constitué de toutes les règles réversibles (rev-term 1), (rev-term 2), (rev-recrut) et (rev-init).

Le système de bas niveau. Tout ceci nous permet de définir formellement le système décentralisé censé réaliser l'assemblage d'arbres.

Définition 18 (Implémentation) *On définit le système* $\mathtt{IMPL}_\mathtt{A}$ *par la donnée du n-uplet* $(\mathcal{T}_i, \mathfrak{n}_i, \mathcal{V}_i, \mathcal{C}_i, \mathfrak{R}_i, I_0)$ *vérifiant :*

- $\mathcal{T}_i = \mathcal{T} = N$
- $\forall \mathtt{t} \in \mathcal{T}_i,\ \mathfrak{n}_i(\mathtt{t}) = 3$
- $\mathcal{V}_i = \{\mathrm{I}, \star, \mathrm{T}\}$
- $\mathfrak{R}_i = (\mathit{init}) \cup (\mathit{recrut}) \cup (\mathit{term}) \cup (\mathit{racine}) \cup \mathit{REV}$
- $I_0 = \mathtt{a}_1\langle \mathrm{I}, \emptyset, \emptyset\rangle\ ,\ \ldots\ ,\ \mathtt{a}_\mathtt{n}\langle \mathrm{I}, \emptyset, \emptyset\rangle$

3.3 La correction

Comme pour le plongement du π-calcul dans $g\kappa$-calcul, on va montrer la correction de l'algorithme en utilisant la notion de bisimulation précédemment introduite. La preuve de correction va s'articuler en deux temps. Tout d'abord, il nous faut établir un certain nombre de propriétés propres aux règles que l'on vient de présenter et prouver que ces propriétés sont préservées par les règles de réduction. Ensuite, on montre que ces invariants suffisent à exhiber une bisimulation entre les deux systèmes $\mathtt{SPEC}_\mathtt{A}$ et $\mathtt{IMPL}_\mathtt{A}$, établissant ainsi la correction de l'algorithme.

Pour la plupart des propriétés que l'on va vérifier, il est commode de ne s'intéresser qu'aux états atteignables de chacun des systèmes. On note $\mathcal{S}^+$ (respectivement $\mathcal{I}^+$) l'ensemble des états S de $\mathtt{SPEC}_\mathtt{A}$ (resp. I de $\mathtt{IMPL}_\mathtt{A}$) tels que $S_0 \longrightarrow^* S$ (resp. $I_0 \longrightarrow^* I$). Ceci permet en quelque sorte de ne se concentrer que sur les déroulements « normaux » des deux systèmes et allège ainsi les preuves.

3.3.1 Propriétés de l'algorithme

Dans cette section, on va tenter de formaliser tous les choix qui ont été faits dans le section 3.2. Il va falloir, par exemple, garantir que les règles rendues réversibles ne s'attaquent jamais à un arbre dont l'assemblage a été finalisé. D'autre part, il faut vérifier que l'organisation interne des agents – à savoir la distinction entre les noms liant les agents à des sous-arbres assemblés et ceux les liant à des sous-arbres dont l'assemblage est toujours partiel – reste cohérente au fur et à mesure de l'évolution du système.

Structure arborescente. Pour cette dernière propriété, une simple étude par cas des règles de $\mathfrak{R}_i$ va pouvoir montrer que (1) un agent n'est jamais connecté à plus d'agents que ne le permet son degré, (2) un agent ne peut pas être lié avec deux noms distincts à un autre agent, (3) (respectivement (4)) les noms contenus dans le second (resp. troisième) site d'un agent se retrouvent de façon unique dans le second (resp. troisième) site d'un autre agent. Ce qui se traduit pas le lemme suivant :

Lemme 5 (Cohérence de la structure interne) *Pour tout état I de $\mathcal{I}^+$ les conditions suivantes sont vérifiées :*

1. $\forall \mathtt{a}\langle p, T, A\rangle \in I$, $|\,T\,| + |\,A\,| \leq \mathtt{dg}(a)$
2. $\forall \mathtt{a_1}\langle p_1, T_1, A_1\rangle \in I$, $\forall \mathrm{x}, \mathrm{y} \in T_1 \cup A_1$ *t.q.* $\mathrm{x} \neq \mathrm{y}$, $\nexists \mathtt{a_2}\langle p_2, T_2, a_2\rangle \in I$ *avec* $\mathrm{x}, \mathrm{y} \in T_2 \cup A_2$
3. $\forall \mathtt{a_1}\langle p, T_1, A_1\rangle \in I$, $\forall \mathrm{x} \in T_1 \setminus \{p\}$, $\exists! \mathtt{a_2}\langle \mathrm{x}, T_2, A_2\rangle$ *avec* $\mathrm{x} \in T_2$
4. $\forall \mathtt{a_1}\langle p, T, A\rangle \in I$, $\forall \mathrm{x} \in A_1 \setminus \{p\}$, $\exists! \mathtt{a_2}\langle \mathrm{x}, T_2, A_2\rangle$ *avec* $\mathrm{x} \in A_2$

où $\{p\}$ est interprété comme l'ensemble vide si p est I, $\star$, *ou* T.

Démonstration : Par induction sur le nombre de réductions qui mènent à l'état atteignable. Le cas de base est vérifié puisque tous les agents sont de la forme $\mathtt{a}\langle \mathrm{I}, \emptyset, \emptyset\rangle$. Pour le cas d'induction, une étude exhaustive des règles permet de vérifier que l'ensemble des propriétés est conservé. En particulier, la condition d'unicité imposée par les conditions 3 et 4 est garantie par le fait que la seule règle créant un nom s'applique entre deux agents.

Stabilité des arbres assemblés. Ce lemme va nous aider à établir deux propriétés relatives à la stabilité des arbres assemblés. Une première proposition permet de caractériser l'ensemble des agents prenant part à un assemblage réussi, ce que l'on décèle dès qu'une règle (racine) est appliquée. Dans ce cas, il est possible d'extraire les agents impliqués dans cet assemblage en reconstruisant la structure à partir de l'agent sur lequel s'est appliquée la règle (racine) et en utilisant les noms contenus dans le second site pour déterminer les agents dont il est le père.

Définition 19 (Processus d'extraction) *Soit* $I \in \mathcal{I}^+$ *et* $a_1 = \mathtt{a_1}\langle \mathrm{T}, T_1, \emptyset\rangle \in I$. *On définit* $\mathtt{ext}_{a_1}(I)$ *comme le sous-ensemble d'agents de* I *constitué par* $\mathtt{a_1}\langle \mathrm{T}, T_1, \emptyset\rangle$, $\mathtt{ext}(T_1, I)$ *avec :*

$$\begin{aligned} \mathtt{ext}(\emptyset, I) &= \mathbf{0} \\ \mathtt{ext}(\mathrm{x} \cup X, I) &= \mathtt{a}\langle \mathrm{x}, \mathrm{x} \cup T, A\rangle \ , \mathtt{ext}(X, I) \, , \mathtt{ext}(T, I) \end{aligned}$$

si $\mathtt{a}\langle \mathrm{x}, \mathrm{x} \cup T, A\rangle \in I$.

La condition 3 du lemme 5 assure qu'il est toujours possible d'extraire un tel agent $\mathtt{a}\langle \mathrm{x}, \mathrm{x} \cup T, A\rangle$ de I. Par ailleurs, ce processus se termine puisque I contient un nombre fini d'agents et que, d'autre part, la remarque 2 garantit qu'on n'extrait pas plusieurs fois le même agent dans le déroulement de la fonction.

Notons que pour l'instant on n'a posé aucune condition sur le troisième site des agents extraits et que l'on ne se sert pas non plus de leur valeur. L'intuition liée à l'algorithme consiste en ce que, quand l'agent racine possède la valeur T, cela signifie que l'assemblage est terminé et que tous les agents participant à la construction ont aussi terminé l'assemblage de la branche dont ils sont la racine. Ce qui se traduit pas la caractérisation suivante :

Proposition 2 (Caractérisation des arbres assemblés) *Soit* $I \in \mathcal{I}^+$ *et* $a_1 = \mathtt{a_1}\langle \mathrm{T}, T_1, \emptyset\rangle \in I$. *Soit* $\mathtt{ext}_{a_1}(I) = a_1 \, , a_2 \, , \ldots \, , a_n$. *Alors pour tout* i *compris entre* 1 *et* n *il existe* $T_i \subseteq \mathcal{C}$ *vérifiant les conditions suivantes :*

- a_i *est de la forme* $\mathtt{a_i}\langle p_i, T_i, \emptyset\rangle$ *avec* $|\, T_i \,| = \mathsf{dg}(a_i)$.
- *pour tout* $x \in T_i$, *il existe un unique* j *tel que* $x \in T_j$

Démonstration : Remarquons tout d'abord que la forme de a_1 répond à la première condition par définition de la règle (racine) dont il provient. La condition 3 du lemme 5 permet alors de montrer inductivement que tous les agents sont de la forme $\mathtt{a_i}\langle p_i, T_i, \emptyset\rangle$. La condition de bord posée sur la règle (term) indique que la taille des T_i est bien égale au degré correspondant aux nœuds a_i. Enfin la deuxième condition n'est rien d'autre que l'application inductive

de la condition 3 du lemme précédent.

La seconde propriété de stabilité concerne les règles réversibles. Il faut en effet s'assurer qu'aucune de ces règles ne s'applique sur un agent faisant parti d'un arbre assemblé. Ceci s'énonce sous la forme d'un corollaire de la proposition précédente :

Corollaire 2 (Non réversibilité des arbres assemblés) *Soient $I \in \mathcal{I}^+$ et $a = \mathtt{a}\langle \mathrm{T}, T, \emptyset\rangle \in I$ tels que $\mathtt{ext}_a(I)$ soit un sous-réseau de I. Alors quelque soit $I' \in \mathtt{IMPL_A}$ et $\mathfrak{r} \in \mathit{REV}$, $\mathtt{ext}_a(I) \overset{\mathfrak{r}}{\nrightarrow} I'$.*

Démonstration : Par cas sur la règle $\mathfrak{r}$ et la forme des agents de $\mathtt{ext}_a(I)$.

3.3.2 Correspondance entre $\mathtt{SPEC_A}$ et $\mathtt{IMPL_A}$

La caractérisation de la proposition 2 nous donne l'intuition nécessaire pour comprendre la correspondance faite entre les agents de $\mathtt{SPEC_A}$ et ceux de $\mathtt{IMPL_A}$. En effet, tout état de $\mathtt{SPEC_A}$ peut être divisé en deux sous-parties : les agents faisant partie d'un arbre construit et ceux encore à l'état initial. À ces derniers on a déjà indiqué qu'on faisait correspondre les agents de la forme $\mathtt{a}\langle \mathrm{I}, \emptyset, \emptyset\rangle$. Pour les autres, on se sert de la caractérisation énoncée précédemment.

Définition 20 ($[S]^{\mathrm{srt}}$) *Soit S un élément de $\mathcal{S}^+$. On définit $[S]^{\mathrm{srt}}$ par induction sur la structure de S :*

$$
\begin{array}{lcl}
[\mathbf{0}]^{\mathrm{srt}} & = & \mathbf{0} \\
[\mathtt{a}\langle \mathrm{I}, \emptyset\rangle]^{\mathrm{srt}} & = & \mathtt{a}\langle \mathrm{I}, \emptyset, \emptyset\rangle \\
[\mathtt{a}\langle \star, X\rangle]^{\mathrm{srt}} & = & \mathtt{a}\langle \mathrm{T}, X, \emptyset\rangle \\
[\mathtt{a}\langle \mathrm{x}, X\rangle]^{\mathrm{srt}} & = & \mathtt{a}\langle \mathrm{x}, X, \emptyset\rangle \\
[S_1\ ,\ S_2]^{\mathrm{srt}} & = & [S_1]^{\mathrm{srt}}\ ,\ [S_2]^{\mathrm{srt}} \\
[(\nu\,\mathrm{x})\,S']^{\mathrm{srt}} & = & (\nu\,\mathrm{x})\,[S']^{\mathrm{srt}}
\end{array}
$$

On peut remarquer que, dans cette correspondance, les troisièmes sites des agents de $\mathtt{IMPL_A}$ n'ont aucun rôle. Ce qui est naturel étant donné que ce troisième site ne contient que les noms liés aux branches dont l'assemblage n'est pas terminé. De même, la valeur $\star$ ne désigne la racine de l'arbre que pendant les étapes d'assemblage. Lorsque l'assemblage est enfin terminé, c'est la valeur T qui désigne la racine. C'est pourquoi, la valeur $\star$ n'apparaît pas dans les agents de $[S]^{\mathrm{srt}}$. Autrement dit, cette correspondance ne met en relation les états de $\mathtt{SPEC_A}$ qu'avec des états *stables* de $\mathtt{IMPL_A}$. Ces états vont évidemment jouer un rôle important dans la définition de la relation de bisimulation.

États stables. Après avoir caractérisé la stabilité des arbres assemblés, on s'attache ici à caractériser plus généralement les états stables du système $\mathtt{IMPL}_{\mathtt{A}}$. Comme le montre la correspondance de la définition 20, ces états sont des traductions parfaites d'arbres cohérents assemblés. L'idée prépondérante qui sous-tend l'algorithme d'assemblage décentralisé réside dans le fait que lorsqu'un arbre est assemblé, aucune connexion n'est plus supprimée. Autrement dit, une fois l'assemblage terminé, plus aucune règle réversible n'est applicable. De ce point de vue, il paraît alors naturel d'associer à un état quelconque I de $\mathtt{IMPL}_{\mathtt{A}}$ un état stabilisé $\mathrm{st}(I)$ défini comme l'état obtenu à partir de I en appliquant le maximum de règles de REV.

Définition 21 (Stabilisation) *Soit I un état de $\mathtt{IMPL}_{\mathtt{A}}$. On définit $\mathrm{st}(S)$ par :*
- $I \to^* \mathrm{st}(I)$ *en n'utilisant que les règles de REV.*
- $\forall \mathfrak{r} \in \mathit{REV}, \forall I', \mathrm{st}(I) \overset{\mathfrak{r}}{\not\to} I'$.

Cette caractérisation va se révéler particulièrement agréable pour définir simplement la bisimulation. L'inconvénient est qu'il faut s'assurer que $\mathrm{st}(\cdot)$ est effectivement une fonction et qu'ainsi elle définit bien un état unique du système. Deux lemmes permettent d'asseoir cette propriété en montrant, d'une part, que le système induit par les règles de REV est localement confluant (lemme 6), puis, d'autre part, qu'il existe un nombre fini d'applications des règles réversibles (lemme 7). Le premier est de plus similaire au lemme 4 qui vérifiait la même propriété mais pour un système de règles différent.

Lemme 6 (Propriété du diamant pour REV) *Soient I, I_1 et I_2 trois états de $\mathtt{IMPL}_{\mathtt{A}}$ et $\mathfrak{r}_1$, $\mathfrak{r}_2$ deux règles de REV. Si $I \overset{\mathfrak{r}_1}{\to} I_1$ et $I \overset{\mathfrak{r}_2}{\to} I_2$ alors il existe I' tel que $I_1 \overset{\mathfrak{r}_2}{\to} I'$ et $I_2 \overset{\mathfrak{r}_1}{\to} I'$.*

Démonstration : Dans le cas où les deux règles s'appliquent à des agents distincts de I, la conclusion est immédiate. Si ce n'est pas le cas, alors il faut regarder le couple (r_1, r_2). Tout d'abord, remarquons qu'aucune des deux règles ne peut être (rev-init). En effet, si l'une des règle est (rev-init), l'agent sur lequel elle s'applique est forcément de la forme $\mathsf{a}\langle\star, \emptyset, \emptyset\rangle$ et seule cette règle peut s'appliquer sur cet agent.
Si les deux règles sont différentes de (rev-init), il faut alors remarquer qu'elles suppriment chacune un nom différent des interfaces. Or la condition (2) du lemme 5 garantit que ces deux noms ne peuvent pas lier deux mêmes agents. On en déduit qu'un seul agent est commun aux deux règles. Une simple étude par cas sur le couple (r_1, r_2) permet de vérifier que, dans tous les cas, les règles suppriment des noms différents des ensembles de noms T ou A, n'empêchant pas la règle concurrente de s'appliquer.

Terminaison du processus de stabilisation. La seconde propriété nécessite par contre d'introduire de nouvelles notions. Il nous faut en effet prouver qu'il n'existe pas de réductions infinies utilisant les règles de REV. Le moyen le plus simple pour ce faire est de définir un ordre strict bien fondé $\prec$ sur l'espace des états de $\texttt{IMPL}_\texttt{A}$ tel que pour toute règle $\mathfrak{r}$ de REV, $I \xrightarrow{\mathfrak{r}} I'$ implique $I' \prec I$. On choisit ici d'utiliser un ordre lexicographique basé sur les valeurs des deux premiers sites des agents. Le site T étant un sous-ensemble de $\mathcal{C}$, il est naturellement muni de l'ordre strict défini par sa taille. De plus, on observe que deux des quatre règles de REV font justement décroître le nombre d'éléments de cet ensemble. Pour la règle (rev-init) en revanche, c'est la valeur du premier site qui est modifiée, faisant passer la valeur de ce site de $\star$ à I. De même, la règle (rev-recrut) change la valeur du site p en I.

Plus formellement, on définit $|\, I \,|_p$ comme étant le nombre de sites p apparaissant dans I dont la valeur n'est pas I. Soit $|\, I \,|_T$ la somme des tailles de l'ensemble T de chaque agent de I. On définit alors la relation $\prec$ sur l'espace des états de $\texttt{IMPL}_\texttt{A}$ par $I \prec I'$ si l'une des deux conditions est vérifiée :

- $|\, I' \,|_p < |\, I \,|_p$
- $|\, I' \,|_p = |\, I \,|_p \wedge |\, I' \,|_T < |\, I \,|_T$

Lemme 7 *Soient I, I' deux états de $\texttt{IMPL}_\texttt{A}$ et $\mathfrak{r}$ une règle de REV. Alors $I \xrightarrow{\mathfrak{r}} I' \Longrightarrow I' \prec I$*

Démonstration : Une étude par cas sur les règles de REV montre que (rev-init) et (rev-recrut) font décroître la valeur $|\, I \,|_p$ tandis que les règles (rev-term 2) et (rev-term 1) laissent la valeur de $|\, I \,|_p$ inchangée mais diminuent la valeur de $|\, I \,|_T$

Corollaire 3 (Terminaison de REV) *Soit I un état de $\texttt{IMPL}_\texttt{A}$. Il n'existe pas de réduction infinie $I \to I_1 \to \ldots \to I_n \to \ldots$ utilisant des règles de REV.*

Ce corollaire, associé au lemme 6, permet de conclure que le processus de stabilisation définit bien un état unique $\mathrm{st}(I)$ associé à I.

Corollaire 4 $\mathrm{st}(\cdot)$ *est une fonction totale.*

3.3.3 La bisimulation proprement dite

Il ne nous reste plus qu'à établir formellement une relation entre nos deux systèmes et à utiliser les résultats des deux sous-sections précédentes pour prouver que cette relation est une bisimulation.

Théoreme 2 (Correction) *La relation binaire définie sur $\mathcal{S}^+ \times \mathcal{I}^+$ par*

$$\mathcal{R} = \left\{ (S, I) \in \mathcal{S}^+ \times \mathcal{I}^+ \mid [S]^{\mathrm{srt}} \equiv \mathrm{st}(I) \right\}$$

est une bisimulation faible.

Démonstration : Supposons $(S, I) \in \mathcal{R}$.

Soient $S' \in \mathcal{S}^+$ et t un arbre tels que $S \xrightarrow{t} S'$. Par définition, $I \rightarrow^* \mathrm{st}(I)$ et $[S]^{\mathrm{srt}} \equiv \mathrm{st}(I)$. Puisque la règle (t) peut s'appliquer sur le réseau S, cela signifie, entre autre, que pour chaque nœud a de $\mathsf{nd}(t)$, il existe un agent $\mathtt{a}\langle \mathrm{I}, \emptyset \rangle$ dans S. D'après la définition 20 de $[\cdot]^{\mathrm{srt}}$, on en déduit qu'il existe aussi un agent associé dans I de la forme $\mathtt{a}\langle \mathrm{I}, \emptyset, \emptyset \rangle$. Reste à trouver un agencement des règles de $\mathfrak{R}_i$ menant à la formation de $[\,[\![t]\!]_{\mathtt{a}}\,]^{\mathrm{srt}}$. Il existe évidemment plusieurs réductions possibles pour cela. Il faut nécessairement commencer par appliquer la règle (init) sur l'agent correspondant à la racine de t. On peut alors dans un premier temps décider d'effectuer tous les recrutements, puis d'appliquer toutes les règles (term). Le lemme 5 nous assure que ces dernières pourront bien être utilisées. Enfin, on termine l'assemblage en appliquant la règle (racine) sur l'agent initiateur de l'assemblage.

L'autre sens de la preuve demande un peu plus de travail. Soient $I' \in \mathcal{I}^+$ et $\mathfrak{r} \in \mathfrak{R}_i$ tels que $I \xrightarrow{\mathfrak{r}} I'$. On définit S' tel que $(S', I') \in \mathcal{R}$ par cas sur la règle $\mathfrak{r}$:

Si $\mathfrak{r} \in$ REV : Alors $I' \rightarrow^* \mathrm{st}(I)$ et il suffit de prendre $S' = S$.

Si $\mathfrak{r} =$ (term) : Alors il est possible d'appliquer la règle (rev-term 1) ou (rev-term 2) en fonction de la valeur du premier site de l'agent a_1 de (term). Il vient alors que $I' \rightarrow I$ et il suffit à nouveau de prendre $S' = S$ pour conclure.

Si $\mathfrak{r} =$ (init) ou (recrut) : Même chose.

Si $\mathfrak{r} =$ (racine) : Soit a_1 l'agent impliqué dans la règle. Par définition, il est de la forme $\mathtt{a_1}\langle \mathrm{T}, T_1, \emptyset \rangle$ dans I'. On peut alors appliquer la fonction d'extraction $\mathtt{ext}_{a_1}(I') = a_1\,,\, a_2\,,\, \ldots\,,\, a_n$. Il est donc d'affiner la description des états I et I' en déduisant qu'il existe I_1 tel que

$$\begin{aligned} I &\equiv I_1\,,\, \mathtt{a_1}\langle \star, T_1, \emptyset \rangle\,,\, \mathtt{a_2}\langle p_2, T_2, \emptyset \rangle\,,\, \ldots\,,\, \mathtt{a_n}\langle p_n, T_n, \emptyset \rangle \\ I' &\equiv I_1\,,\, \mathtt{a_1}\langle \mathrm{T}, T_1, \emptyset \rangle\,,\, \mathtt{a_2}\langle p_2, T_2, \emptyset \rangle\,,\, \ldots\,,\, \mathtt{a_n}\langle p_n, T_n, \emptyset \rangle \end{aligned}$$

Il vient alors que $\mathrm{st}(I) \equiv \mathrm{st}(I_1)\,,\, \mathtt{a_1}\langle \mathrm{I}, \emptyset, \emptyset \rangle\,,\, \mathtt{a_2}\langle \mathrm{I}, \emptyset, \emptyset \rangle\,,\, \ldots\,,\, \mathtt{a_n}\langle \mathrm{I}, \emptyset, \emptyset \rangle$. Comme d'autre part $\mathrm{st}(I) \equiv [S]^{\mathrm{srt}}$, on en déduit là encore que l'on peut préciser la forme de S. Il existe donc S_1 tel que $[S_1]^{\mathrm{srt}} \equiv \mathrm{st}(I_1)$ et

$$S \equiv S_1\,,\, \mathtt{a_1}\langle \mathrm{I}, \emptyset \rangle\,,\, \mathtt{a_2}\langle \mathrm{I}, \emptyset \rangle\,,\, \ldots\,,\, \mathtt{a_n}\langle \mathrm{I}, \emptyset \rangle$$

La règle (t) peut donc s'appliquer sur S et il suffit alors de prendre $S' = S_1\,,\, [\![t]\!]_{\mathtt{a}}$ pour conclure que $[S']^{\mathrm{srt}} = [S_1]^{\mathrm{srt}}\,,\, [\,[\![t]\!]_{\mathtt{a}}\,]^{\mathrm{srt}} \equiv \mathrm{st}(I')$.

3.4 Discussion

En guise de conclusion, on esquissera quelques remarques destinées à comparer l'approche qui a été celle de ce chapitre avec d'autres alternatives offertes. Deux directions s'offrent pour la critique. On peut, d'une part, se demander quels étaient les autres cadres théoriques permettant de résoudre le problème de l'auto-assemblage, que ce soit en termes de description de l'algorithme, aussi bien qu'en termes de correction de celui-ci. Il est, d'autre part, tout à fait légitime de se demander quelles ont été les répercussions sur l'algorithme présenté dans la section 3.2 des choix qui ont été faits lors de la phase de spécification et quelles sont alors les extensions possibles.

Pour ce qui est du cadre théorique, on présentera une autre approche basée sur RCCS [DK04, DK05]. On ne donnera ici qu'une description informelle de la méthode et des résultats dans une simple perspective de comparaison. La description complète nécessiterait de développer des outils hors de propos du présent travail.

Dans un second temps, on mettra en évidence quels sont les choix qui ont été faits et qui ont conduit à une version simple de l'algorithme, à la fois en termes de description et de correction. On proposera alors des solutions pour étendre la version proposée dans le présent chapitre, de façon à introduire le chapitre 4 destiné à l'assemblage de graphes, qui s'affirme ainsi comme une généralisation de celui-ci.

3.4.1 Une autre approche

On l'a vu dans la section 3.2, l'algorithme peut clairement se découper en deux parties, une première établissant les règles réalisant l'assemblage – c'est à dire créant les connexions et gérant les changements répercutés sur l'interface des agents – et une seconde s'employant à défaire les assemblages ayant conduit à une impasse. Cette dichotomie se retrouve d'ailleurs clairement dans la section dédiée à la correction puisqu'un certain nombre de notions ne sont introduites que pour gérer ce mécanisme de retour en arrière. Ce problème est totalement indépendant de la question de l'auto-assemblage proprement dit et est en réalité inhérent à l'utilisation même des algèbres de processus. Il est donc légitime de chercher à découpler formellement les deux parties (évolution en avant et en arrière) pour ne se concentrer, dans la phase de description de l'algorithme, que sur la partie constructive de celui-ci.

Or la théorie des algèbres de processus se dote justement de plus en plus de variantes, baptisées *algèbres réversibles*, qui permettent de rendre explicite ce découplage. Le langage RCCS développé par Vincent Danos et Jean Krivine constitue l'une des variantes qui s'adapte le mieux à l'auto-assemblage d'arbres. Ce calcul se construit comme une extension de CCS dans laquelle les processus sont équipés de mémoires leur permettant, sous certaines conditions, de revenir sur certaines synchronisations en distinguant syntaxiquement

les synchronisations réversibles de celles qui ne le sont pas.

Du point de vue de l'élaboration d'un algorithme distribué, le développement se déroule en deux étapes. Une première phase dans laquelle on décrit l'algorithme en termes de CCS en annotant les actions qui sont destinées à être irréversibles (ce qu'on représentera par des actions soulignées). Et une deuxième phase, transparente du point de vue du développeur, dans laquelle on plonge ce système dans RCCS, récupérant du même coup le mécanisme de retour arrière inhérent au langage. Encore faut-il être capable de décrire l'algorithme en utilisant la syntaxe offerte par CCS. Dans le cas de l'auto-assemblage d'arbres, le travail réalisé dans [DKT06] montre qu'il est justement possible, quoique quelque peu ardu, de décrire les règles (init), (recrut), (term) et (racine) de la section 3.2 en termes de CCS comme on peut le constater sur la figure 3.4.

$$\begin{array}{lcll} \text{NODE}_a & \stackrel{def}{=} & \tau.(\text{BUILD}_a^{\mathsf{dg}(a)} \mid \text{WAIT}_{a\star}^{\mathsf{dg}(a)}) & (1) \\ & + & \sum_{b\in N} \text{x}_{ab}.(\text{BUILD}_a^{\mathsf{dg}(a)-1} \mid \text{WAIT}_{ab}^{\mathsf{dg}(a)-1}) & \\ & & & \\ \text{BUILD}_a^{n+1} & \stackrel{def}{=} & \sum_{b\in N} \bar{\text{x}}_{ab}.\text{BUILD}_a^n & (2) \\ \text{BUILD}_a^0 & \stackrel{def}{=} & 0 & (3) \\ & & & \\ \text{WAIT}_{a\alpha}^{n+1} & \stackrel{def}{=} & \text{w}_a.\text{WAIT}_{a\alpha}^n & (4) \\ \text{WAIT}_{ab}^0 & \stackrel{def}{=} & \bar{\text{w}}_b.0 & (5) \\ \text{WAIT}_{a\star}^0 & \stackrel{def}{=} & \underline{\text{ok}_a}.0 & (6) \end{array}$$

FIG. 3.4 – L'algorithme d'auto-assemblage d'arbres décrit en CCS.

Chaque nœud a de N est ici représenté par un processus NODE_a qui peut soit initier la construction d'un nouvel arbre (action τ de la règle (1)), ce qui correspond alors à la règle (init) de notre description, soit se synchroniser avec un agent actif, ce qui correspond alors à la description de l'agent a_2 de la règle (recrut). Dans les deux cas ce processus se divise en deux sous-processus BUILD_a^n et WAIT_{ap}^n où n désigne le nombre de connexions restantes (en fonction de $\mathsf{dg}(a)$) et p le père de l'agent ($\star$ dans le cas où il est la racine). La phase de recrutement décrite par (recrut) se retrouve ici divisée en deux termes complémentaires : la règle (2), qui spécifie le comportement de l'agent recruteur, et la seconde partie de la règle (1). De même, la phase de transmission du signal de fin, couverte en une déclaration par la règle (term),

est à mettre en relation avec les processus complémentaires (4) et (5). Finalement, la finalisation de l'assemblage par (racine) se retrouve décrite ici par une règle similaire (6) dans laquelle on observe que le nom de canal utilisé pour la synchronisation est souligné, correspondant au fait que cette étape est irréversible.

On le voit, la description en terme CCS de la partie constructive de l'algorithme demande quelques contorsions d'ordre syntaxique dues en réalité à une asymétrie dans les communications qui est propre au langage. La dualité émission/réception ne présente, dans notre cas, aucun intérêt particulier. En contrepartie, la notion de correction se trouve simplifiée, et ce pour deux raisons. Tout d'abord parce que les règles relatives à la partie destructive de l'algorithme sont écartées de la description et donc n'entrent pas en compte dans les preuves. Ensuite, de façon plus profonde, parce qu'il n'est plus nécessaire de s'intéresser à l'ensemble de tous les états atteignables de l'implémentation mais seulement au sous-ensemble constitué des états *causalement* atteignables, c'est-à-dire dont la suite de réductions menant à ces états se termine par une action irréversible et dans laquelle toutes les étapes intermédiaires sont nécessaires à la réalisation cette dernière action. Le système de transition causal ainsi défini est évidement plus simple (en nombre d'états notamment) que le système interprété dans CCS. Dans notre cas, cela reviendrait à ne considérer que le sous-ensemble $\{\mathrm{st}(I) \mid I \in \mathcal{I}^+\}$ des états stables au lieu de la totalité des éléments de $\mathcal{I}^+$. Son étude suffit pourtant à déterminer la correction du système de départ. Un méta-théorème lié à RCCS montre en effet que tout processus CCS, une fois interprété dans la sémantique de RCCS, est faiblement bisimilaire au système de transition causal induit par ce processus. Il ne reste plus alors qu'à montrer que ce système causal est lui-même faiblement bisimilaire à la spécification pour obtenir la correction générale, ce qui correspond essentiellement ici au lemme 5, à la proposition 2 et à une version simplifiée du théorème 2 de la section 3.3.

3.4.2 Le trucage

Plusieurs des choix qui ont été faits dans la section 3.1.2 ont eu des répercussions non négligeables sur l'algorithme présenté en section 3.2. On peut remarquer par exemple qu'une simplification substantielle a été faite à propos du critère de finalité des assemblages. On s'est contenté dans cette version de tester l'adéquation entre le nombre d'agents connectés et les degrés respectifs autorisés par la fonction $\mathsf{dg}(\cdot)$. Autrement dit, on s'autorise à construire absolument n'importe quel arbre cohérent, sans aucune décision préalable par l'agent initiateur quant à la forme qu'aura l'arbre final. Les interactions se font de façon hasardeuse tant qu'elles n'entrent pas en conflit avec les degrés de agents. On pourrait imaginer un problème similaire mais plus complexe dans lequel seul un sous-ensemble d'arbres cohérents serait recherché. Il est

tout à fait possible d'étendre l'algorithme de la section 3.2 pour tenir compte de cette nouvelle contrainte. Il suffirait de rajouter un quatrième site à chacun des agents de $\texttt{IMPL}_{\texttt{A}}$ dont le but serait de mémoriser la carte des connexions qu'il doit rechercher. Cette carte serait tout simplement la branche de l'arbre cible dont est en charge l'agent. La règle d'initiation de l'arbre deviendrait alors plus contraignante puisque pour chaque arbre $t = (a, \{t_1, \ldots, t_n\})$ que l'on chercherait à assembler, on ajouterait la règle :

$$(\mathsf{init\ t})\ \frac{\mathsf{a}\langle \mathrm{I}, \emptyset, \emptyset, \emptyset\rangle}{\mathsf{a}\langle \star, \emptyset, \emptyset, \{t_1, \ldots, t_n\}\rangle}$$

Seuls les agents racines d'un arbre cible seraient cette fois autorisés à initier un assemblage. La règle de recrutement serait quelque peu modifiée pour ne lier que des agents conformes aux racines des arbres contenus dans le quatrième site. Ce qui se traduirait pas la règle :

$$(\mathsf{recrut'})\ \frac{\mathsf{a}_1\langle p, T, A, \{(a_2, \{t_1, \ldots, t_n\})\} \cup C\rangle\ ,\ \mathsf{a}_2\langle \mathrm{I}, \emptyset, \emptyset, \emptyset\rangle}{(\nu\, \mathrm{x})\, \mathsf{a}_1\langle p, T, A \cup \{\mathrm{x}\}, C\rangle\ ,\ \mathsf{a}_2\langle \mathrm{x}, \emptyset, \{\mathrm{x}\}, \{t_1, \ldots, t_n\}\rangle}$$

Le reste des règles seraient alors similaires à celles présentées section 3.2 mais en tenant compte de la nouvelle interface des agents.

Dans un même ordre d'idées, une fois ciblé un arbre cohérent que l'on désire assembler, on pourrait contraindre le chemin de construction à éviter certains assemblages partiels car ici tous les chemins menant à la construction d'un arbre cohérent sont possibles. Or il paraît assez naturel, si l'on pense à des assemblages de molécules notamment, d'avoir besoin de contraindre quelque peu la suite de connexions établissant l'assemblage final.

De même, la règle (recrut) ne permet de recruter que des agents inactifs. Il serait pourtant possible de relâcher cette contrainte pour permettre le recrutement de n'importe quel agent racine d'un arbre, tant que cet assemblage fait progresser la structure générale vers l'assemblage final.

Les preuves présentées dans la sections 3.3 s'accommoderaient en fait très bien des types de contraintes proposés ici mais ces notions vont en réalité être reprises dans le prochain chapitre et développées dans le but de permettre l'auto-assemblage de graphes avec toute la flexibilité que l'on vient d'énoncer.

Chapitre 4

Auto-assemblage de graphes

Sommaire

Dans le chapitre précédent nous avons présenté un algorithme effectuant l'assemblage distribué d'arbres cohérents. Comme nous l'avons précisé dans la partie discussion, le critère choisi pour déterminer l'arbre à construire était relativement simple et avait l'avantage de trouver une solution concise dans notre langage. Nous allons à présent généraliser le procédé en complexifiant la spécification de manière à incorporer plus de flexibilité dans les règles et changer la nature de la structure à assembler. Les objets dont nous nous préoccupons ici sont désormais des graphes et non plus des arbres[1].

La section 4.1 s'attache à reformuler le problème d'auto-assemblage pour l'étendre au cas des graphes. Puis la section 4.2 en propose une solution sous forme d'algorithme décentralisé et donne une preuve partielle de la correction de ce dernier. La correction totale ne vient que dans la section 4.3 dans laquelle nous étendons le système de règles pour tenir compte des impasses éventuelles. Nous terminons enfin le chapitre en revenant sur les résultats obtenus.

[1]Ce chapitre est tiré de la publication [DT05] qui a été étendue pour le présent travail

4.1 Reformulation du problème

La section 3.1 a établi formellement le rapport entre notre langage, défini au chapitre 2, et la structure sous-jacente basée sur le partage des noms de $\mathcal{C}$. Il s'est avéré que celle-ci pouvait s'exprimer en terme d'hypergraphes. Dès lors, il était naturel d'étendre le cas de l'assemblage d'arbres afin de tenir compte de cette richesse.

Bien que l'étude d'assemblage d'hypergraphes soit donc possible, il nous semble cependant intéressant de considérer le cas des graphes, étant donné que les applications potentielles sont plus nombreuses. En outre, il sera facile de constater tout au long du présent chapitre que les règles et propriétés énoncées s'adaptent parfaitement à une telle généralité et qu'il n'y a pas de raison pour que les résultats qui vont être présentés ne puissent pas être étendus au cas des hypergraphes.

4.1.1 Une notation pour les graphes

Nous allons donc reformuler la définition 15 afin de l'adapter au cas particulier des graphes. Soit $G = (V, E)$ un graphe. On représente chaque élément de V par un agent de type `noeud` dont l'interface se compose d'un unique site, et chaque arc $e = (v_1, v_2)$ de E par le partage d'un nom $\mathrm{x}_e \in \mathcal{C}$ entre les deux agents représentant les nœuds v_1 et v_2. Il faut pour cela nommer chacune des connexions de manière unique.

Définition 22 *Soit (V, E) un graphe. Soit une fonction injective $\mathsf{arc}(\cdot) : E \mapsto \mathcal{C}$. Pour tout élément de v de V on définit l'ensemble $\mathsf{Noms}(v)$ des noms partagés par v par :*

$$\mathsf{Noms}(v) = \left\{ \mathrm{x} \in \mathcal{C} \mid \exists v' \text{ tel que } (v, v') \in E \wedge \mathrm{x} = \mathsf{arc}(v, v') \right\}$$

Soit $\mathsf{Noms} = \cup_{v \in V} \mathsf{Noms}(v)$, on définit alors le réseau $[\![(V, E)]\!]_{\mathsf{col}}$ associé au graphe (V, E) par :

$$[\![(V, E)]\!]_{\mathsf{col}} = (\nu\, \mathsf{Noms})\, (\mathtt{noeud}\langle \mathsf{Noms}(v_1) \rangle \ , \ \ldots \ , \ \mathtt{noeud}\langle \mathsf{Noms}(v_n) \rangle)$$

Rappelons par ailleurs que tout réseau d'agents est assorti de son graphe de connexions qui se définit en considérant l'ensemble des noms de $\mathcal{C}$ contenu dans l'interface des agents. Formellement, on peut traduire tout réseau d'agents en un réseau de graphes par la fonction d'abstraction $w(\cdot)$ définie inductivement sur la structure d'un réseau :

$$\begin{aligned}
w(\mathbf{0}) &= \mathbf{0} \\
w(\mathtt{t}\langle s_1, \ldots, s_n \rangle) &= \mathtt{noeud}\langle (\cup_i s_i) \cap \mathcal{C} \rangle \\
w(S_1 \,,\, S_2) &= w(S_1) \,,\, w(S_2) \\
w((\nu\, \mathrm{x})\, S) &= (\nu\, \mathrm{x})\, w(S)
\end{aligned}$$

Spécification. Soit $G = (V, E)$ un graphe. Nous cherchons à assembler une population de n agents déconnectés, c'est-à-dire de la forme $\texttt{noeud}\langle\emptyset\rangle$, en un maximum de copies de G. Par la suite, nous noterons $\langle\emptyset\rangle_n$ pour le réseau composé de n agents $\texttt{noeud}\langle\emptyset\rangle$. Le système de spécification $\texttt{SPEC}(G, n)$ se définit donc très simplement à l'aide de l'unique règle d'interaction :

$$(\mathsf{G})\ \frac{\langle\emptyset\rangle_{|\,V\,|}}{[\![G]\!]_{\mathsf{col}}}$$

Définition 23 (Spécification) *Soient G un graphe et n un entier. Le système de spécification $\texttt{SPEC}(G, n)$ est défini par la donnée de $(\mathcal{T}, \mathfrak{n}, \mathcal{V}, \mathcal{C}, \mathfrak{R}_G, S_0^n)$ vérifiant :*

- $\mathcal{T} = \{\texttt{noeud}\}$
- $\mathfrak{n}(\texttt{noeud}) = 1$
- $\mathcal{V} = \emptyset$
- $\mathfrak{R}_G = \{(\mathsf{G})\}$
- $S_0^n = \langle\emptyset\rangle_n$

Comme pour le chapitre 3, il nous faut maintenant déterminer un second système, équivalent à $\texttt{SPEC}(G, n)$ mais dans lequel les règles d'interaction sont strictement binaires.

4.1.2 Scénarios de construction

Comme pour l'auto-assemblage d'arbres, le problème de la règle (G) est qu'elle implique la participation d'un nombre arbitraire d'agents et la création, en une seule étape de toutes les connexions définies par G. Afin de rendre ces interactions élémentaires, on propose une première étape dans laquelle on décompose la construction de G en un ensemble de règles ajoutant les arcs un à un. On définit donc un ensemble de graphes $\mathcal{G} = \{G_1, \ldots, G_n\}$ constitué de sous-graphes de G. En un sens, $\mathcal{G}$ représente un ensemble de graphes intermédiaires par lesquels il est nécessaire de passer pour atteindre le graphe final G. Afin de représenter formellement cette construction en deux étapes, nous définissons donc un premier système $\texttt{M}(\mathcal{G}, n)$ qui définit la *macro-implémentation* de $\texttt{SPEC}(G, n)$, construite sur $\mathcal{G}$, mais qui ne résout pas encore le problème de l'auto-assemblage. Elle constitue cependant une étape essentielle pour déterminer la *micro-implémentation* $\texttt{I}_{\mathfrak{m}}(\mathcal{G}, n)$ qui, elle, répondra à toutes les contraintes du problème.

Notation 1 *Afin de différencier plus facilement les graphes de $\mathcal{G}$ du graphe final qui est recherché, nous noterons ce dernier $G_{\texttt{F}}$.*

Graphes concrets et graphes abstraits. Deux opérations vont être nécessaires pour traduire la construction incrémentale du graphe $G_{\mathtt{F}}$. Mais auparavant, nous allons donner une représentation plus stricte des graphes, qui nous servira, non seulement pour définir ces opérations de construction, mais aussi dans la phase d'élaboration du système $\mathtt{I}_{\mathtt{m}}(\mathcal{G}, n)$ de la section 4.2. Nous appelons *graphe concret* tout graphe de la forme $G = (\{1, \ldots, n\}, E)$ avec $n > 0$. Ce graphe est qualifié de *non-trivial* si $n > 1$. Étant donné un graphe concret G, on dénote par $[G]$ sa classe d'isomorphisme, ce que nous nommons *graphe abstrait* afin de le distinguer de G. Par extension, on dira qu'un graphe abstrait est non trivial s'il n'est pas la classe d'isomorphisme d'un graphe concret trivial.

Nous définissons deux opérations sur les graphes concrets. La première permet l'ajout d'un arc dans une composante connexe :

Définition 24 (Jonction interne) *Soit $G = (V, E)$ un graphe concret et u et v deux éléments de V tels que $(u, v) \notin E$. On définit la* jonction interne *de u et v dans G, notée $G + (u, v)$ par le graphe $(V, E \cup \{(u, v)\})$.*

La seconde opération permet de définir le graphe concret résultant de l'ajout d'un arc entre deux composantes disjointes.

Définition 25 (Jonction binaire) *Soient $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ deux graphes concrets et $u_1 \in V_1$ et $u_2 \in V_2$ deux nœuds. On définit la* jonction binaire *G_1 et G_2 par les nœuds u_1 et u_2, notée $G_1.u_1 + G_2.u_2$, par le graphe $G = (V, E)$ vérifiant :*

$$\begin{aligned} V &= \{1, \ldots, |\, V_1 \,| + |\, V_2 \,|\} \\ E &= E_1 \cup \{(u + |\, V_1 \,|, v + |\, V_1 \,|) \mid (u, v) \in E_2\} \cup \{(u_1, u_2 + |\, V_1 \,|)\} \end{aligned}$$

Cette dernière opération décale les entiers désignant les nœuds du second graphe par le nombre de nœuds du premier graphe. Si, par exemple, $H_1 = (\{1, 2, 3\}, \{(1, 2), (2, 3)\})$ et $H_2 = (\{1, 2\}, \{(1, 2)\})$. Alors le graphe résultant de la jonction binaire entre H_1 et H_2 via 3 et 1 produit le graphe $H = (\{1, 2, 3, 4, 5\}, \{(1, 2), (2, 3), (4, 5), (3, 4)\})$. Plus visuellement, cela donne la transformation suivante :

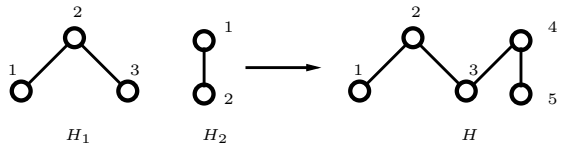


FIG. 4.1 – Jonction binaire entre graphes concrets

Ces deux définitions s'étendent naturellement aux graphes abstraits et définissent un ordre partiel, noté $<$ [2], sur les graphes concrets et sur les graphes abstraits. Cet ordre nous aide à caractériser les contraintes à poser sur $\mathcal{G}$ afin qu'il donne effectivement une version incrémentale de la construction de $G_{\mathtt{F}}$. Il faut en particulier que l'on puisse exhiber une suite croissante d'éléments $\mathcal{G}$ telle que chacun de ces éléments puisse se décomposer en éléments plus petits par l'ajout d'un seul arc. Ce qui se caractérise formellement par la propriété suivante :

Propriété 2 (Scénario de construction) *Soit $G_{\mathtt{F}}$ un graphe. Un ensemble $\mathcal{G}$ constitué de sous-graphes de $G_{\mathtt{F}}$ est un* scénario de construction *de $G_{\mathtt{F}}$ si $G_{\mathtt{F}}$ est l'élément maximal de $\mathcal{G}$ et si, pour tout graphe non trivial H de $\mathcal{G}$, l'une des conditions suivantes est vérifiée :*

- *il existe un graphe H' et deux nœuds u et v tels que $H' + (u, v) \simeq H$.*
- *il existe deux graphes H_1 et H_2 et deux nœuds u_1 et u_2 tels que $H_1.u_1 + H_2.u_2 \simeq H$.*

Notons qu'il est toujours possible de définir un ensemble $\mathcal{G}$ vérifiant ces conditions, ne serait-ce qu'en saturant $\mathcal{G}$ de tous les sous-graphes possibles de $G_{\mathtt{F}}$. Mais le but recherché ici est plutôt de permettre de *paramétrer* l'assemblage d'un graphe $G_{\mathtt{F}}$ à l'aide d'un tel ensemble $\mathcal{G}$.

Étant donné un graphe $G_{\mathtt{F}}$ et un scénario de construction $\mathcal{G}$ associé, on peut donc proposer un premier système $\mathtt{M}(\mathcal{G}, n)$ qui implémente $\mathtt{SPEC}(G_{\mathtt{F}}, n)$ en utilisant la traduction dans $g\kappa$-calcul des règles de $\mathcal{G}$. En effet, les deux conditions de la propriété 2 trouvent naturellement leur interprétation en terme de règles de $g\kappa$-calcul en utilisant la fonction $[\![\cdot]\!]_{\mathsf{col}}$ de la définition 22. On notera par la suite $\mathfrak{R}_{\mathcal{G}}$ pour désigner l'ensemble de ces règles.

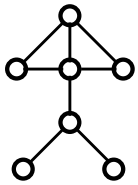
Définition 26 (Macro-implémentation) *Soit n un entier, $G_{\mathtt{F}}$ un graphe et $\mathcal{G}$ un scénario de construction ayant $G_{\mathtt{F}}$ pour graphe terminal, on définit $\mathtt{M}(\mathcal{G}, n)$ par la donnée du n-uplet $(\mathcal{T}_M, \mathfrak{n}_M, \mathcal{V}_M, \mathcal{C}, \mathfrak{R}_{\mathcal{G}}, M_o^n)$ avec :*

- $\mathcal{T}_M = \mathcal{T} = \{\mathtt{noeud}\}$
- $\mathfrak{n}_M(\mathtt{tnoeud}) = 1$
- $\mathcal{V}_M = \mathcal{V} = \emptyset$
- $M_0^n = S_0^n = \langle\emptyset\rangle_n$

Reste que, si ce système donne bien une description incrémentale de la construction des connexions requises pour assembler $G_{\mathtt{F}}$, les agents participant aux règles d'interaction sont toujours arbitrairement nombreux et donc contredisent les conditions d'auto-assemblage.

[2] $G < G + (u, v)$ et $G_i < G_1.u_1 + G_2.u_2$ pour $i = 1, 2$

Un exemple. Avant de continuer, arrêtons nous un instant sur un exemple complet illustrant à la fois les notions introduites jusqu'ici et l'argumentation soutenant la construction en deux étapes. Supposons par exemple que l'on veuille assembler le graphe $G_{\mathtt{F}}$ suivant :



Il serait simple de définir une unique règle d'assemblage qui connecte, de façon simultanée, 8 agents déconnectés selon la forme voulue. Du point de vue de la question de l'auto-assemblage, une telle étape atomique est inenvisageable. Le problème consiste donc à déterminer un ensemble de règles élémentaires (en terme d'agents participants) tel que, quelque soit le nombre n d'agents déconnectés, ceux-ci s'assemblent en $\lfloor n/7 \rfloor$ copies de $G_{\mathtt{F}}$. Une première simplification permet de décomposer cette règle en spécifiant les étapes qui ajoutent les connexions une à une. Prenons par exemple le scénario de construction $\mathcal{G}$ de la figure 4.2

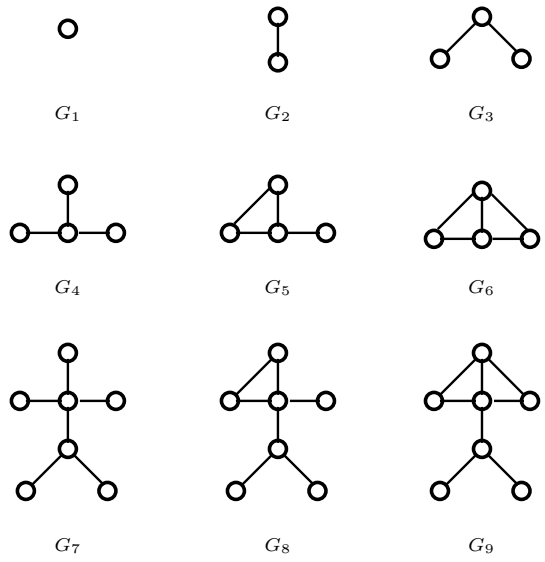


FIG. 4.2 – Un exemple de scénario de construction

Tout élément de cet ensemble peut être vu comme un réseau d'agents en reprenant la traduction donnée par la définition 22. Par exemple, à G_3 correspond le réseau

$$(\nu\, \mathrm{x}, \mathrm{y})\,(\mathtt{noeud}\langle\{\mathrm{x}\}\rangle\ ,\ \mathtt{noeud}\langle\{\mathrm{x}\,,\ \mathrm{y}\}\rangle\,,\mathtt{noeud}\langle\{\mathrm{y}\}\rangle)$$

ou n'importe quel autre réseau équivalent. Il est aussi possible de fournir une vision plus dynamique de $\mathcal{G}$ en se basant sur la propriété 2 vérifiée par $\mathcal{G}$. On peut en effet déterminer toutes les interactions possibles entre les graphes de $\mathcal{G}$ en terme de jonctions internes ou binaires. Cela conduit au diagramme de la figure 4.3, auquel on fera référence par la suite sous le nom de *graphe d'assemblage*, où les nœuds correspondent aux éléments de $\mathcal{G}$ et dans lequel les arcs simples font références aux jonctions internes, tandis que les arcs doubles renvoient aux jonctions binaires.

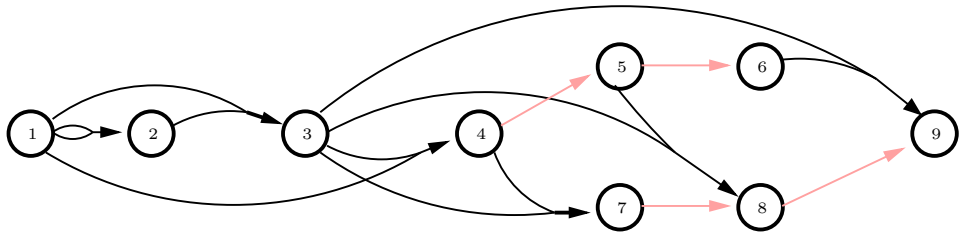


FIG. 4.3 – Un exemple de graphe d'assemblage

Encore une fois, ce diagramme doit être lu comme une description des interactions entre réseaux d'agents. Ainsi, l'arc double entre les graphes G_3 et G_1 et qui mène au graphe G_4 correspond formellement à la règle d'interaction suivante :

$$\frac{\texttt{noeud}\langle\{\mathrm{x}\}\rangle\ ,\ \texttt{noeud}\langle\{\mathrm{x},\mathrm{y}\}\rangle\ ,\ \texttt{noeud}\langle\{\mathrm{y}\}\rangle\ ,\ \texttt{noeud}\langle\emptyset\rangle}{(\nu\,\mathrm{z})\,\texttt{noeud}\langle\{\mathrm{x}\}\rangle\ ,\ \texttt{noeud}\langle\{\mathrm{x},\mathrm{y},\mathrm{z}\}\rangle\,,\ \texttt{noeud}\langle\{\mathrm{y}\}\rangle\,,\ \texttt{noeud}\langle\{\mathrm{z}\}\rangle}$$

À cette étape, on peut définir la macro-implémentation $\texttt{M}(\mathcal{G}, n)$ qui consiste, d'une part, en la recherche exhaustive de toutes les interactions possibles à partir de la description de $\mathcal{G}$ en terme de jonctions binaires et internes, puis, d'autre part, en leur traduction en règles d'interaction. Il faut bien voir cependant que, si chacune des règles est élémentaire en terme de nombre de connexions établies, du point de vue de l'assemblage décentralisé, on n'a en revanche pas progressé. Certaines règles, comme celle présentée ci-dessus, demandent toujours un consensus entre un nombre arbitraire d'agents, ce qui contredit la contrainte d'élémentarité liée au problème. Cette étape constitue cependant un bon pas en avant puisque chacune des règles ne crée qu'une seule connexion entre deux agents. Il reste donc à déterminer comment traduire chaque règle en séries d'interactions entre deux agents uniquement.

4.2 Implémentation des scénarios

Cette section est dédiée à la définition des règles implémentant un scénario $\mathcal{G}$. Nous commençons tout d'abord par une présentation informelle de l'algo-

rithme avant de définir précisément les différents types d'agents qui entrent en jeu et les règles d'interactions qui leur sont associées. Nous poursuivrons ensuite par deux sous-sections consacrées à la preuve de la correction du système ainsi défini vis-à-vis de $\mathtt{M}(\mathcal{G}, n)$.

4.2.1 La micro-implémentation

Commençons donc par une présentation informelle. Nous cherchons ici à structurer l'interface des agents afin d'organiser les informations nécessaires pour implémenter les règles du scénario. L'idée principale qui va sous-tendre l'algorithme consiste en ce que, pour chaque composante connexe, un seul agent est autorisé à créer des connexions à un instant donné. Afin de décider si une telle connexion conduit à une modification autorisée par $\mathcal{G}$, cet agent possède une *image* de la structure de sa propre composante sous forme de graphe concret. Le fait de n'autoriser qu'un seul agent à créer des connexions à un instant donné évite des modifications concurrentes de la structure. Le besoin d'aboutir à un consensus au sein de la composante n'est donc plus nécessaire. Cet agent, que l'on qualifiera désormais d'*agent actif*, est donc le seul à avoir besoin de l'image de la composante à laquelle il appartient.

Une autre information essentielle, et qui va de pair avec l'image de la composante, est celle du rôle joué par un agent dans la composante du point de vue de cette image. Autrement dit, quel est l'entier qui représente l'agent dans le graphe concret. Afin de pouvoir être identifié lors des communications, chaque agent doit donc être muni à tout moment de cet entier décrivant sa position dans la composante.

Si on regarde dans le détail la définition 25, on voit que l'effet d'une jonction entre deux composantes disjointes n'affecte le rôle des agents que dans l'une des deux composantes. Dans la perspective d'une implémentation concrète, il est donc tout à fait possible de choisir la plus petite de ces composantes. D'autre part, on voit aussi qu'il est très simple de déterminer le nouveau rôle des agents concernés. Il est défini par un simple ajout, à l'entier représentant l'ancien rôle, du nombre d'agents de l'autre composante. Au moment de la mise à jour il suffira donc simplement de transmettre ce nombre aux agents concernés.

Le cas de la jonction interne est encore plus simple puisque les rôles restent inchangés. Les mises à jour ne sont donc pas requises et seule l'image de la composante, que possède l'agent actif, est modifiée. Par contre, il est nécessaire d'être capable de reconnaître un agent faisant partie de la même composante. Les informations utilisées jusque-là ne suffisent plus et nous utiliserons un nom particulier, dénoté par id, commun à tous les agents d'une même composante.

Pour éviter les conflits lors de la phase de mises à jour, dans le cas où la structure des connexions est cyclique, on utilise un *arbre de recouvrement*, bâti sur la structure de la composante, pour transmettre les informations relatives aux changements. Cet arbre de recouvrement est lui-même une structure dy-

namique qui s'enrichit en même temps que la composante entière. Un nouvel arc est ajouté à cet arbre à chaque jonction entre deux composantes disjointes. Afin de conserver cette distinction entre les connexions ayant eu lieu entre deux composantes disjointes et celles ayant créé un cycle dans la composante, deux sites différents vont être utilisés dans l'interface des agents. On dénotera celui mémorisant les connexions faisant partie de l'arbre de recouvrement par A et le second, conservant les connexions cycliques, par C. Notons que l'arbre de recouvrement ainsi construit n'est pas dirigé et que le sens de transmission des informations sera lui-aussi déterminé de façon dynamique.

Les agents. À partir de cette explication informelle, il est possible de définir les différentes interfaces d'agents. Soit $\mathcal{T}_g = \{\texttt{act}, \texttt{maj}, \texttt{pass}\}$ avec $\mathfrak{n}(\texttt{act}) = 5$, $\mathfrak{n}(\texttt{maj}) = 6$ et $\mathfrak{n}(\texttt{pass}) = 4$. Nous utiliserons les notations suivantes pour faire références aux différents sites des interfaces :

$\texttt{act}\langle \text{id}, A, C, r, G\rangle$	**Agent actif**
$\texttt{maj}\langle \text{id}, A, C, r, M, d\rangle$	**Agent en phase de mise à jour**
$\texttt{pass}\langle \text{id}, A, C, r\rangle$	**Agent passif**

où :

- **id** $\in \boldsymbol{\mathcal{C}}$ est un nom utilisé comme identifiant de groupe. Il est partagé par tous les agents appartenant à la même composante connexe
- $\boldsymbol{A} \subseteq \boldsymbol{\mathcal{C}}$ est le site conservant l'ensemble des noms de connexions qui ont été créés entre deux agents appartenant à des composantes disjointes (et donc voisins maintenant dans l'arbre de recouvrement)
- $\boldsymbol{C} \subseteq \boldsymbol{\mathcal{C}}$ est le site mémorisant les autres noms de connexions
- $\boldsymbol{r} \in \mathbb{N}$ est le nœud représenté par l'agent dans le graphe concret détenu par l'agent actif
- $\boldsymbol{G}$ est le graphe concret désignant la composante
- $\boldsymbol{M} \subseteq \boldsymbol{R}$ est le sous-ensemble des noms contenus dans $\boldsymbol{R}$ connectant l'agent de type `maj` aux autres agents dont il faut mettre à jour l'interface
- $\boldsymbol{d}$ est le décalage à réaliser pour déterminer le nouveau rôle des agents.

Les règles d'interaction. Soit $G_{\mathtt{F}} = (V, E)$ un graphe. Par abus de notation, on écrira par la suite $|\,G_{\mathtt{F}}\,|$ pour dénoter le nombre $|\,V\,|$ de nœuds du graphe. Soit $\mathcal{G}$ un scénario d'assemblage de $G_{\mathtt{F}}$. Nous pouvons maintenant définir les règles décrivant le comportement des agents détaillés ci-dessus. Commençons par les règles permettant de faire évoluer la structure des composantes. Ces règles vont évidemment reposer sur $\mathcal{G}$ et sur la propriété 2 qui lui est associée. Une première règle, correspondant à la définition 25, permet de créer une connexion entre deux agents appartenant à deux composantes dis-

jointes. Soient $G' = G_1.r_1 + G_2.r_2$ et $d = |\, G_1 \,|$. Si $[G']$ est un graphe abstrait appartenant à $\mathcal{G}$, alors on ajoute la règle ($\mathsf{JoncBin(G')}$) suivante :

$$\frac{\mathtt{act}\langle \mathrm{id}_1, A_1, C_1, r_1, G_1\rangle \ , \ \mathtt{act}\langle \mathrm{id}_2, A_2, C_2, r_2, G_2\rangle}{(\nu\, \mathrm{x})\, \mathtt{act}\langle \mathrm{id}_1, A_1 \cup \{\mathrm{x}\}, C_1, r_1, G'\rangle \ , \ \mathtt{maj}\langle \mathrm{id}_1, A_2 \cup \{\mathrm{x}\}, C_2, r_2 + d, A_2, d\rangle}$$

La connexion est représentée par l'ajout d'un nouveau nom x aux seconds sites des deux interfaces. Seul un des agents reste encore actif après l'application de cette règle et c'est évidemment celui qui conserve l'image de la nouvelle composante. Le second agent, quant à lui, modifie son identifiant de groupe – récupérant celui de l'agent actif id_1 – ainsi que son rôle, qui est calculé à partir de son ancien rôle r_2 et de la taille du graphe G_1. Il se prépare par ailleurs à transmettre ces nouvelles informations aux agents faisant partie de son ancienne composante. Il s'appuie pour cela sur l'arbre de recouvrement de son ancienne composante représenté par A_2. Nous noterons par la suite ($\mathsf{JoncBin}(\mathcal{G})$) l'ensemble $\bigcup_{G\in\mathcal{G}}(\mathsf{JoncBin(G)})$ de toutes les règles d'interaction binaire.

La seconde règle permettant de faire évoluer la structure d'une composante précise les conditions d'ajout d'une connexion entre des agents qui appartiennent déjà à une même composante. Soit $G' = G + (r_1, r_2)$. Si $[G'] \in \mathcal{G}$ alors on ajoute la règle suivante :

$$(\mathsf{JoncInt(G')})\ \frac{\mathtt{act}\langle \mathrm{id}, A_1, C_1, r_1, G_1\rangle \ , \ \mathtt{pass}\langle \mathrm{id}, A_2, C_2, r_2\rangle}{(\nu\, \mathrm{x})\, \mathtt{act}\langle \mathrm{id}, A_1, C_1 \cup \{\mathrm{x}\}, r_1, G'\rangle \ , \ \mathtt{pass}\langle \mathrm{id}, A_2, C_2 \cup \{\mathrm{x}\}, r_2\rangle}$$

Notons que, cette fois-ci, les sites A_1 et A_2 restent inchangés puisque les agents font partie de la même composante connexe, comme l'indique le fait qu'ils possèdent tous deux le même identifiant de groupe id. C'est pourquoi le nouveau nom x est ajouté aux troisièmes sites respectifs (l'ajouter aux seconds sites créerait alors un cycle dans l'arbre de recouvrement). Comme nous l'avons déjà précisé aucune mise à jour n'est requise puisque le rôle des agents ne change pas (cf. définition 24), ni même l'identifiant de groupe. Là encore nous désignerons par ($\mathsf{JoncInt}(\mathcal{G})$) l'ensemble $\bigcup_{G\in\mathcal{G}}(\mathsf{JoncInt(G)})$.

Les deux règles suivantes sont destinées à actualiser l'interface des agents au fur et à mesure de l'évolution du système. En particulier, la phase de mise à jour qui s'opère dans l'une des deux composantes réunies par l'application de ($\mathsf{JoncBin}$) requiert un peu d'attention :

$$(\mathsf{Maj})\ \frac{\mathtt{maj}\langle \mathrm{id}_1, A_1, C_1, r_1, M \cup \{\mathrm{x}\}, d\rangle \ , \ \mathtt{pass}\langle \mathrm{id}_2, A_2 \cup \{\mathrm{x}\}, C_2, r_2\rangle}{\mathtt{maj}\langle \mathrm{id}_1, A_1, C_1, r_1, M, d\rangle \ , \ \mathtt{maj}\langle \mathrm{id}_1, A_2 \cup \{\mathrm{x}\}, C_2, r_2 + d, A_2, d\rangle}$$

La mise à jour d'un agent passif consiste à changer son rôle dans la composante (calculé à partir de d), son identifiant de groupe, ainsi que son type. Cet agent devient à son tour de type `maj` afin de poursuivre la vague de mises à jour le long de l'arbre de recouvrement. Notons en particulier que le nom qui a servi à actualiser cet agent ne fait pas partie de l'ensemble des noms utilisés par celui-ci pour poursuivre la mise à jour.

Cette vague de mises à jour prend fin pour un agent lorsque tous ses voisins dans l'arbre de recouvrement ont été contactés. Ce qui se traduit par la règle suivante :

$$(\mathsf{Fin})\ \frac{\mathtt{maj}\langle \mathrm{id}, A, C, r, \emptyset, d\rangle}{\mathtt{pass}\langle \mathrm{id}, A, C, r\rangle}$$

Pour finir, l'agent actif peut décider à tout moment de transmettre la capacité de créer des connexions à un autre agent faisant partie de sa composante connexe :

$$(\mathsf{Pass})\ \frac{\mathtt{act}\langle \mathrm{id}, A_1, C_1, r_1, G\rangle\ ,\ \mathtt{pass}\langle \mathrm{id}, A_2, C_2, r_2\rangle}{\mathtt{pass}\langle \mathrm{id}, A_1, C_1, r_1\rangle\ ,\ \mathtt{act}\langle \mathrm{id}, A_2, C_2, r_2, G\rangle}$$

Notons que cette règle n'intervient que si les agents possèdent le même identifiant de groupe. Cela signifie en particulier que l'agent qui devient actif a reçu la mise à jour s'il en avait besoin.

La définition du système. On peut désormais définir formellement le système implémentant le scénario de construction $\mathcal{G}$. Il ne nous reste plus qu'à décrire l'état initial qui consiste en n agents déconnectés. Ces agents sont par conséquent seuls dans leur composante, qui est réduite à un graphe à un seul nœud. Un seul graphe concret est donc possible pour la représenter. Soit $G_i = (\{1\}, \emptyset)$, ces agents ont ainsi comme seul rôle possible 1. D'autre part ils doivent tous avoir un identifiant de groupe différent au départ. Chaque agent sera donc de la forme $(\nu\,\mathrm{id})\,\mathtt{act}\langle \mathrm{id}, \emptyset, \emptyset, 1, G_i\rangle$. Dans la suite, nous désignerons par $\langle \mathrm{id}, \emptyset, \emptyset, 1, G_i\rangle_n$ le réseau composé de n agents $(\nu\,\mathrm{id})\,\mathtt{act}\langle \mathrm{id}, \emptyset, \emptyset, 1, G_i\rangle$.

Définition 27 (Micro-implémentation) *Soient n un entier et $\mathcal{G}$ un scénario de construction. On définit $\mathtt{I_m}(\mathcal{G}, n)$ par la donnée de $(\mathcal{T}_i, \mathfrak{n}_i, \mathcal{V}_i, \mathcal{C}, \mathfrak{R}_i(\mathcal{G}), I_0^n)$ vérifiant :*

- $\mathcal{T}_i = \{\mathtt{act}, \mathtt{maj}, \mathtt{pass}\}$
- $\mathfrak{n}_i(\mathtt{t}) = \begin{cases} 5 & si \quad \mathtt{t} = \mathtt{act} \\ 6 & si \quad \mathtt{t} = \mathtt{maj} \\ 4 & si \quad \mathtt{t} = \mathtt{pass} \end{cases}$

- $\mathcal{V}_i = \{G \mid G \text{ est un graphe concret }\} \cup \mathbb{N}$
- $\mathfrak{R}_i(\mathcal{G}) = (\mathit{JoncBin}(\mathcal{G})) \cup (\mathit{JoncInt}(\mathcal{G})) \cup \{(\mathit{Maj}), (\mathit{Fin}), (\mathit{Pass})\}$
- $I_0^n = \langle \mathrm{id}, \emptyset, \emptyset, 1, G_i \rangle_n$

La correction de l'algorithme est cette fois-ci plus complexe et c'est pourquoi nous allons la décomposer en deux parties reflétant le découpage en deux étapes qui a mené à la définition de $\mathtt{I}_\mathtt{m}(\mathcal{G}, n)$. En effet, nous sommes passés par une première phase dans laquelle nous avons simplifié le problème en décomposant les connexions créées, afin que chaque règle ne génère qu'une seule connexion à la fois. Puis nous avons décrit la manière de générer les règles implémentant cette construction en ne faisant intervenir que deux agents au plus lors des interactions.

Nous allons donc commencer par démontrer la correction de la micro-implémentation vis-à-vis du scénario de construction. Cela demande de vérifier que les règles de $\mathtt{I}_\mathtt{m}(\mathcal{G}, n)$ préservent un certain nombre d'invariants, comme le fait qu'il n'existe qu'un seul agent actif par composante et que cet agent possède une image correcte de sa composante. La correction totale de l'algorithme nécessite par contre, comme dans le chapitre 3, d'établir un mécanisme gérant les impasses. C'est ce à quoi se consacrera la section 4.3.

4.2.2 Cohérence du réseau

Afin de caractériser ces invariants, quelques notations supplémentaires vont être introduites ici. On défini un *graphe recouvert pointé* comme un triplet (G, A, n) où G est un graphe connexe, A un arbre de recouvrement de G et n un nœud particulier considéré comme la racine. Comme pour la fonction d'abstraction w qui extrait le graphe sous-jacent à un réseau d'agents R, il est possible d'isoler le graphe recouvert pointé sous-jacent à R s'il existe.

Définition 28 (Agents recouvrant) *Soit* $\mathtt{garp}$ *un type tel que* $\mathtt{n}(\mathtt{garp}) = 3$. *Un agent est un* agent recouvrant *s'il est de la forme* $\mathtt{garp}\langle A, C, i\rangle$ *avec* $A, C \subseteq \mathcal{C}$ *et* $i \in \mathbb{N}$. *La fonction* RVT *faisant correspondre aux agents de* $\mathtt{I}_\mathtt{m}(\mathcal{G}, n)$ *les agents recouvrant est définie par :*

$$\begin{aligned}
\textsc{rvt}(\mathtt{act}\langle \mathrm{id}, A, C, r, G\rangle) &= \mathtt{garp}\langle A, C, 1\rangle \\
\textsc{rvt}(\mathtt{maj}\langle \mathrm{id}, A, C, r, M, d\rangle) &= \mathtt{garp}\langle A, C, 0\rangle \\
\textsc{rvt}(\mathtt{pass}\langle \mathrm{id}, A, C, r\rangle) &= \mathtt{garp}\langle A, C, 0\rangle
\end{aligned}$$

Cette définition s'étend naturellement aux réseaux comme pour w. Notons que RVT étiquette les agents actifs avec un 1 et les autres avec un 0. Puisque RVT préserve tous les noms de $\mathcal{C}$, il vient que pour tout agent a, $w(\textsc{rvt}(a)) = w(a)$. Deux agents $\mathtt{garp}\langle A_1, C_1, i_1\rangle$ et $\mathtt{garp}\langle A_2, C_2, i_2\rangle$ sont dits voisins dans l'arbre de recouvrement si $A_1 \cap A_2 \neq \emptyset$. Notons que le réseau obtenu $\textsc{rvt}(R)$ n'est

pas forcément la représentation d'un graphe recouvert pointé. S'il l'est, alors on dénotera par $[\textsc{rvt}(R)]$ le réseau abstrait recouvert pointé sous-jacent. On dénotera par GARP l'ensemble des graphes abstraits recouverts pointés.

Notation 2 *Comme pour le chapitre 3, nous ne nous intéresserons qu'aux états atteignables de* $\texttt{SPEC}(G_{\texttt{F}}, n)$ *(resp.* $\texttt{M}(\mathcal{G}, n)$ *et* $\texttt{I}_{\texttt{m}}(\mathcal{G}, n)$*) que nous noterons dorénavant* $\mathcal{S}_g^+$ *(resp.* $\mathcal{M}^+$ *et* $\mathcal{I}_m^+*$*).*

Notation 3 *Pour tout agent, nous dénoterons par* $\mathsf{id}(a)$ *la valeur de son premier site* id *et par* $\mathsf{arvt}(a)$ *la valeur de son second site A.*

Munis de ces notations, il nous est maintenant possible de définir formellement les caractéristiques laissées invariantes par les règles de $\mathfrak{R}_i$. La propriété suivante les caractérise, établissant qu'un réseau I de $\texttt{I}_{\texttt{m}}(\mathcal{G}, n)$ est cohérent si (1) son réseau de connexions dénote un graphe abstrait recouvert pointé, si (2) dès que deux agents partagent le même identifiant de groupe, alors ils appartiennent à la même composante connexe, si (3) dès qu'un agent est actif, son image pointée (G, r) est isomorphe à la composante à laquelle il appartient dans $[\textsc{rvt}(I)]$ et si (4) lorsqu'un agent est de type `maj`, tout nom apparaissant dans sa liste de noms à mettre à jour apparaît aussi dans l'interface d'un agent voisin dans l'arbre de recouvrement. Notons au passage que la réciproque de la condition 2 n'est en général pas vraie puisque l'identifiant de groupe peut ne pas avoir été encore propagé à tous les agents de la composante.

Définition 29 (Cohérence) *Un réseau* I *de* $\mathcal{I}_m^+$ *est dit cohérent si :*

1. $[\textsc{rvt}(I)] \in$ GARP
2. *Pour tout agent* a_1, a_2 *de* I, *si* $\mathsf{id}(a_1) = \mathsf{id}(a_2)$ *alors* a_1 *et* a_2 *appartiennent à la même composante*
3. *Pour tout agent* $\texttt{act}\langle \mathrm{id}, A, C, r, G\rangle$ *de* I, (G, r) *est isomorphe à sa composante dans* $[\textsc{rvt}(I)]$
4. *Pour tout agent* $\texttt{maj}\langle \mathrm{id}, A, C, r, M, d)\rangle$ *de* I, $M \subseteq A$, *et pour tout* $\mathrm{x} \in M$, *il existe un unique autre agent* a *de* I *tel que* $\mathrm{x} \in \mathsf{id}(a)$.

Une des propriétés indispensable pour obtenir la correction de l'algorithme consiste à prouver, dans un premier temps, que toutes les règles décrites dans la section précédente préservent la cohérence du réseau.

Proposition 3 (Invariance de la cohérence) *Pour tout scénario* $\mathcal{G}$, *pour tout réseau* $I \in \mathcal{I}_m^+$, *si* I *est cohérent et* $I \rightarrow_{\mathfrak{R}_i} I'$ *alors* I' *est cohérent.*

Démonstration : Par induction sur les règles de $\mathfrak{R}_i$. Le cas de base est trivial puisque tous les agents sont actifs, libres de connexion, possèdent un identifiant de groupe différent, ont pour image le graphe initial G_i et jouent le seul rôle

possible dans ce graphe.
Supposons maintenant que $I_0^n \rightarrow_{\mathfrak{R}_i}^* I \xrightarrow{\mathfrak{r}}_{\mathfrak{R}_i} I'$. Le pas inductif dépend alors de la règle $\mathfrak{r}$ appliquée :

(JoncBin) : Soient a_1 et a_2 les deux agent impliqués dans la règle $\mathfrak{r}$, c'est-à-dire tels que :

$$\begin{aligned} a_1 &= \mathtt{act}\langle \mathrm{id}_1, A_1, C_1, r_1, G_1 \rangle \\ a_2 &= \mathtt{act}\langle \mathrm{id}_2, A_2, C_2, r_2, G_2 \rangle \end{aligned}$$

Puisque $[\textsc{rvt}(I)] \in$ GARP et que les deux agents sont actifs dans I, on en déduit qu'ils appartiennent à des composantes disjointes. Ajouter un nom dans le voisinage de recouvrement préserve donc la structure de recouvrement. Comme seul un des agents reste actif après l'application de la règle, il vient que la condition 1 est préservée. De plus, cette condition garantit également que pour tout nom x de A_2, il existe un unique agent a dans I' tel que x $\in \mathsf{arvt}(a)$. Ainsi, a_2 vérifie la condition 4 dans I'. Par 3, il vient également que (G_1, r_1) et (G_2, r_2) concordent avec les rôles que jouent a_1 et a_2 dans leur composante respective. Or la définition 25 montre que la nouvelle image $G_1.r_1 + G_2.r_2$ ajoute précisément un arc entre les nœuds r_1 et r_2 et que le rôle du nœud r_1 reste inchangé. L'agent a_1 qui reste actif après l'application de la règle a donc toujours une image cohérente pointée $(G_1.r_1 + G_2.r_2\ ,\ r_1)$ de la composante à laquelle il appartient. Finalement, la condition 2 reste vraie puisque les seuls identifiants égalisés par la règle sont ceux de a_1 et a_2 qui font désormais partie de la même composante connexe.

(JoncUn) : Cette règle ajoute une boucle dans la composante mais l'arbre de recouvrement reste identique. D'autre part, l'agent actif reste le même. Puisque $[\textsc{rvt}(N)] \in$ GARP (condition 1), on en déduit que cette condition est toujours valide après application de la règle. En utilisant un argument similaire au cas précédent, il vient que l'agent actif concerné par $\mathfrak{r}$ vérifie toujours la condition 3. Les conditions 2 et 4 ne sont pas affectées par la règle.

(Maj) : Seules les conditions 2 et 4 sont concernées par la règle. L'identifiant de groupe de l'agent qui subit la mise à jour est modifié, mais la condition 4 garantit que les deux agents appartiennent à la même composante. Comme pour le cas de (JoncBin), le fait que $[\textsc{rvt}(I)]$ soit un graphe abstrait recouvrant pointé assure que cet agent vérifie maintenant la condition 4. Cette condition est aussi valide pour l'agent responsable de la mise à jour puisque la règle fait décroître l'ensemble des agents contenus dans le site M de celui-ci.

(Fin) : Rien n'est changé, excepté le type d'un agent entrant en mode passif. Toutes les conditions sont satisfaites par induction.

(Pass) : La règle requiert que les deux agents concernés possèdent un identifiant de groupe commun. D'après la condition 2, ils appartiennent à la même composante. Comme le nouveau rôle d'un agent est transmis en même temps que l'identifiant de groupe, il s'ensuit que le rôle de l'agent qui reçoit l'activité correspond effectivement à son rôle dans le graphe concret transmis par

l'agent précédemment actif. Cet agent a donc une représentation correcte de la composante à laquelle il appartient et du rôle qu'il y joue (condition 3). Les autres conditions ne sont pas affectées par la règle.

La seconde propriété qu'il est nécessaire de vérifier est que les agents ainsi définis ne construisent que des graphes autorisés par le scénario de construction. Formellement :

Proposition 4 $I \in \mathcal{I}_m^+ \Longrightarrow w(I) \in \mathcal{M}^+$.

Démonstration : La démonstration se fait par induction sur la longueur de la séquence de réductions $I_0^n \rightarrow_{\mathfrak{R}_i}^* I$. Le cas de base est évident puisque $w(I_0^n) = w(\langle \mathrm{id}, \emptyset, \emptyset, 1, G_i \rangle_n) = \langle \emptyset \rangle_n$. Supposons maintenant que $I_0^n \rightarrow_{\mathfrak{R}_i}^* I \xrightarrow{\mathfrak{r}}_{\mathfrak{R}_i} I'$. On raisonne par cas sur $\mathfrak{r}$.

Si une connexion est créée, alors les agents impliqués par la règle ont une vue cohérente dans I de la composante à laquelle ils appartiennent (d'après la condition 3 de la proposition 3). Les conditions requises pour que la règle puisse s'appliquer – à savoir $[G_1.r_1 + G_2.r_2] \in \mathcal{G}$ pour (JoncBin) et $[G + (r_1, r_2)] \in \mathcal{G}$ pour (JoncUn) – garantissent que la jonction (binaire ou interne) peut effectivement avoir lieu dans $w(I)$, amenant cet état à devenir $w(I')$.

Si aucune connexion n'est créée, alors la règle ne modifie pas la structure de la composante et $w(I') = w(I)$.

4.2.3 Correction partielle

Il est temps d'établir la correction de $\mathtt{I}_\mathtt{m}(\mathcal{G}, n)$ vis-à-vis de $\mathtt{M}(\mathcal{G}, n)$. Une partie de la preuve vient d'être faite par la proposition 4, qui stipule que les graphes assemblés sont bien ceux du scénario $\mathcal{G}$. Il nous reste donc à démontrer la réciproque, à savoir que chaque étape du scénario peut effectivement être simulée par les règles de la micro-implémentation. Ceci demande de définir formellement une relation entre les états des deux systèmes et de montrer ensuite que cette relation est une bisimulation.

Notation 4 *Soit G un graphe, nous dénoterons par* (Jonc(G)) *l'ensemble des règles constituées de l'union entre* (JoncBin(G)) *et* (JoncInt(G)) *et par* (Jonc($\mathcal{G}$)) *l'union* $\bigcup_{G \in \mathcal{G}}$(Jonc(G)).

Définition 30 (Relation $\approx_\mathcal{G}$) *Les règles*

$$(\textit{Init}) \qquad \frac{}{\langle\emptyset\rangle_n \approx_{\mathcal{G}} \langle \mathrm{id}, \emptyset, \emptyset, 1, G_i\rangle_n}$$

$$(\textit{Etape}) \qquad \frac{M \xrightarrow{G}_{\mathfrak{R}_{\mathcal{G}}} M' \qquad M \approx_{\mathcal{G}} I \qquad I \xrightarrow{\mathfrak{r}}_{\mathfrak{R}_i(\mathcal{G})} I' \qquad \mathfrak{r} \in (\textit{Jonc}(\mathcal{G}))}{M' \approx_{\mathcal{G}} I'}$$

$$(\textit{Inv}) \qquad \frac{M \approx_{\mathcal{G}} I \qquad I \xrightarrow{\mathfrak{r}}_{\mathfrak{R}_i(\mathcal{G})} I' \qquad \mathfrak{r} \notin (\textit{Jonc}(\mathcal{G}))}{M \approx_{\mathcal{G}} I'}$$

définissent inductivement une relation binaire $\approx_{\mathcal{G}}$ *sur* $\mathcal{M}^+ \times \mathcal{I}_m^+$.

Remarquons que la relation préserve la correspondance entre les graphes sous-jacents aux réseaux des deux systèmes.

Proposition 5 $M \approx_{\mathcal{G}} I \Longrightarrow M \equiv w(I)$.

Démonstration : Par induction sur $\approx_{\mathcal{G}}$ et proposition 4.

Le reste de cette section est consacré à prouver que $\approx_{\mathcal{G}}$ est une bisimulation.[3] La seule difficulté restante consiste à prouver que les réseaux de $\mathtt{I}_{\mathtt{m}}(\mathcal{G}, n)$ sont capables de simuler n'importe quelle étape du scénario, car les mises à jour peuvent ne pas avoir encore été effectuées. En effet, une part non négligeable des communications entre agents est utilisée à transmettre les informations qui ont affecté leur composante après la création d'une connexion. Cette phase de mise à jour a été conçue dans le but d'être la plus petite possible en terme de communications. Par exemple, on ne demande pas à ce que les agents d'une composante soient mis à jour avant de permettre à la structure d'évoluer à nouveau si cela n'est pas nécessaire. Remarquons par ailleurs que la règle (Maj) transmet à la fois le nouveau rôle joué par l'agent dans la composante et l'identifiant de groupe dans une même interaction. Puisque dans le pire des cas seul l'agent actif possède les informations correctes concernant le nouvel état de la composante – ce qui arrive juste après l'application d'une règle (JoncBin) – il devient aisé de caractériser l'ensemble des agents mis à jour à un instant donné pour une composante. On peut dès lors séparer cette composante en deux groupes disjoints : d'un côté l'ensemble des agents qui ont le même identifiant de groupe que l'agent actif (et ont par conséquent déjà subi la vague de mises à jour) et, de l'autre côté, le reste des agents.

[3] La propriété obtenue est légèrement plus générale que celle recherchée puisqu'on ne présuppose jamais qu'il existe un unique graphe terminal dans le scénario $\mathcal{G}$.

Cette flexibilité trouve sa contrepartie dans la preuve de correction, puisqu'il nous faut maintenant garantir que chaque agent appartenant à une composante *peut* être mis à jour et que, si un agent est mis à jour, son interface ne sera plus changée à moins qu'un événement ne vienne modifier la structure de la composante. Ces propriétés s'appuient directement sur la structure utilisée pour transmettre les informations. En particulier, l'arbre recouvrant va garantir qu'il n'y a pas de boucle dans la phase de mises à jour. Dans le but d'exprimer ceci formellement, on définit pour chaque réseau I de $\mathcal{I}_m^+$ son *état propre* noté $\text{PR}(I)$, obtenu en lui appliquant systématiquement toutes les règles (Maj) et(Fin).

Définition 31 (État propre) *Soit I un élément de $\mathcal{I}^+$. On définit $\text{PR}(I)$ comme le réseau vérifiant les conditions :*

- *$I \rightarrow^* \text{PR}(I)$ en utilisant seulement les règles (Maj) et (Fin)*
- *$\forall r \in \{(\mathsf{Maj}), (\mathsf{Fin})\}$, $\forall I'$, $\text{PR}(I) \overset{r}{\nrightarrow} I'$.*

Notons que le système est localement confluent puisque les règles de mises à jour n'empêchent pas l'application des autres règles de mises à jour. De plus, les conditions 1 et 4 de la proposition 3 impliquent que la structure utilisée pour transmettre les mises à jour est un arbre, garantissant du même coup l'absence de boucle. Puisque l'ensemble d'agents à contacter décroît lors de l'application de ces règles il s'ensuit que le processus termine. On en déduit que le système est confluent et que $\text{PR}(\cdot)$ définit un $\text{PR}(I)$ unique.

Bien sûr, comme nous l'avons déjà remarqué, ces réductions ne changent pas la structure sous-jacente au réseau, ni les images détenues par les agents actifs.

Lemme 8 $M \approx_{\mathcal{G}} I \Longrightarrow M \approx_{\mathcal{G}} \text{PR}(I)$.

Démonstration : En utilisant la règle (Invis)

Enfin, nous pouvons présenter et prouver le résultat établissant la correction partielle de l'algorithme :

Théoreme 3 (Correction partielle) *Pour tout entier n et tout scénario de construction $\mathcal{G}$, les systèmes $\mathtt{M}(\mathcal{G}, n)$ et $\mathtt{I_m}(\mathcal{G}, n)$ sont bisimilaires.*

Démonstration : On montre que $\approx_{\mathcal{G}}$ est une bisimulation.

Supposons que $M \approx_{\mathcal{G}} I$ et que $M \overset{G}{\rightarrow}_{\mathfrak{R}_{\mathcal{G}}} M'$ pour un graphe G et un réseau M' donnés. Par la définition 31, nous avons que $I \rightarrow^*_{\mathfrak{R}_i(\mathcal{G})} \text{PR}(I)$ et, par le lemme 8, il vient que $M \approx_{\mathcal{G}} \text{PR}(I)$. En fonction de l'agent actif à ce moment-là dans la composante concernée par r, le système $\text{PR}(I)$ peut avoir besoin d'utiliser la règle (Pass) pour transmettre l'activité aux agents concernés, menant le système en I'. Par (Inv), $M \approx_{\mathcal{G}} I'$ et par la proposition 5, il vient que M et I' ont exactement les mêmes graphes sous-jacents. Maintenant on peut

déterminer une règle $\mathfrak{r} \in Jonc(G)$ applicable à I', menant à I'' et $M' \approx_{\mathcal{G}} I''$ par application de (Etape).

Inversement, supposons que $M \approx_{\mathcal{G}} I$ et $I \xrightarrow{\mathfrak{r}}_{\mathfrak{R}_i(\mathcal{G})} I'$ pour $\mathfrak{r}$ et I' donnés. On montre qu'il existe M' tel que $M \rightarrow^*_{\mathfrak{R}_{\mathcal{G}}} M'$ et $M' \approx_{\mathcal{G}} I'$ en raisonnant par cas sur $\mathfrak{r}$. Si $\mathfrak{r} \in \mathsf{Jonc(G)}$, alors c'est une conséquence directe de la proposition 5. Puisque les deux systèmes ont les mêmes graphes sous-jacents, (G) peut être appliqué à M, menant le système en M'. On prouve alors $M' \approx_{\mathcal{G}} I'$ par application de (Etape). Si par contre $\mathfrak{r} \notin \mathsf{Jonc}(\mathcal{G})$, il suffit de prendre $M' = M$ et nous concluons par l'application de (Inv).

4.3 Dislocation des composantes

Le système de règles de $\mathtt{I}_{\mathtt{m}}(\mathcal{G}, n)$, défini dans la précédente section, est monotone en ce sens que les arcs ne peuvent être qu'ajoutés. Comme dans le chapitre 3, les agents peuvent pourtant rivaliser pour l'obtention des ressources. Des graphes partiellement construits, au sens du graphe final $G_{\mathtt{F}}$, peuvent donc être bloqués dans leur évolution, alors que d'autres choix de connexions auraient pu mener à l'assemblage de $G_{\mathtt{F}}$. Il est donc certain que le système $\mathtt{I}_{\mathtt{m}}(\mathcal{G}, n)$ ainsi défini ne peut être bisimilaire à $\mathtt{SPEC}(G_{\mathtt{F}}, n)$. Nous consacrons cette section à l'implémentation d'un mécanisme permettant de gérer de telles impasses.

4.3.1 Remise à zéro

La solution que nous proposons est basée sur le travail d'Eric Klavins [Kla02], qui suggère de munir les agents d'un compte à rebours au bout duquel ils détruisent les connexions qu'ils ont établies au sein de leur composante, afin de recommencer l'assemblage depuis l'état initial. Dans notre approche formelle nous ne représentons pas directement ce compteur. Nous intégrons simplement une version abstraite de ce mécanisme. Le chapitre 5, consacré à l'implémentation réelle de l'algorithme que nous avons présenté ici, commentera, entre autre, cette partie. Pour le moment la démarche consiste à étendre la définition de $\mathfrak{R}_{\mathcal{G}}$, afin d'autoriser chaque composante connexe à supprimer toutes les connexions en une étape. Soit $\mathcal{G}$ un scénario de construction, et G un graphe de $\mathcal{G}$ tel que $|\,G\,| = n$, on définit la règle suivante :

$$(\mathsf{Dest(G)})\ \frac{[\![G]\!]_{\mathsf{col}}}{\langle\emptyset\rangle_n}$$

Ce qui nous permet d'étendre l'ensemble des règles de $\mathfrak{R}_{\mathcal{G}}$ pour tenir compte

de ces remises à zéro. On définit donc $\mathfrak{R}^{\mathtt{et}}_{\mathcal{G}}$ par

$$\mathfrak{R}^{\mathtt{et}}_{\mathcal{G}} = \mathfrak{R}_{\mathcal{G}} \cup \bigcup_{G \in \mathcal{G}} (\mathsf{Dest(G)})$$

Le système étendu de la macro-implémentation se définit alors très simplement en ne remplaçant que l'ensemble des règles qui régit le comportement des agents. Ce qui donne le système suivant :

$$\mathtt{M}^{\mathtt{et}}(\mathcal{G}, n) = (\mathcal{T}_M, \mathfrak{n}_M, \mathcal{V}_M, \mathcal{C}, \mathfrak{R}^{\mathtt{et}}_{\mathcal{G}}, M^n_o)$$

Micro-implémentation étendue. Ces modifications se répercutent dans $\mathtt{I}_{\mathtt{m}}(\mathcal{G}, n)$ par l'introduction d'un nouveau type, `dest`, qui indique que l'agent est en train de propager un message de dislocation de la composante. Puisque celle-ci est vouée à être décomposée, il n'est plus nécessaire de conserver les informations relatives à sa structure. Par contre, il faut que l'agent se souvienne des autres agents à contacter pour s'assurer que la composante sera bien totalement désassemblée. C'est pourquoi cet agent ne comporte qu'un seul site qui contient l'ensemble des noms (correspondant à l'arbre de recouvrement ou non) des connexions précédemment établies. L'identifiant de groupe, l'image éventuelle de la composante et le rôle que jouait l'agent sont supprimés.

Pour finir, il reste à déterminer les règles qui vont permettre de rendre effective cette dislocation. Nous commençons par la règle qui initie le processus. Afin de prévenir d'éventuelles communications avec la composante en cours de dislocation, nous forçons l'agent actif à être celui qui déclenche ce processus :

$$(\mathsf{D\text{-}init})\ \frac{\mathtt{act}\langle \mathrm{id}, A, C, r, G\rangle}{\mathtt{dest}\langle A \cup C\rangle}$$

Suivent les règles de transmission du message de rupture, qui se propage via toutes les connexions à la fois et non plus seulement le long de l'arbre de recouvrement :

$$\text{(D-prop-pass A)}\ \frac{\mathtt{dest}\langle X \cup \{\mathrm{x}\}\rangle\ ,\ \mathtt{pass}\langle \mathrm{id}, A \cup \{\mathrm{x}\}, C, r\rangle}{\mathtt{dest}\langle X\rangle\ ,\ \mathtt{dest}\langle A \cup C\rangle}$$

$$\text{(D-prop-pass C)}\ \frac{\mathtt{dest}\langle X \cup \{\mathrm{x}\}\rangle\ ,\ \mathtt{pass}\langle \mathrm{id}, A, C \cup \{\mathrm{x}\}, r\rangle}{\mathtt{dest}\langle X\rangle\ ,\ \mathtt{dest}\langle A \cup C\rangle}$$

$$\text{(D-prop-maj A)}\ \frac{\mathtt{dest}\langle X \cup \{\mathrm{x}\}\rangle\ ,\ \mathtt{maj}\langle \mathrm{id}, A \cup \{\mathrm{x}\}, C, r, M, d\rangle}{\mathtt{dest}\langle X\rangle\ ,\ \mathtt{dest}\langle A \cup C\rangle}$$

$$\text{(D-prop-maj C)}\ \frac{\mathtt{dest}\langle X \cup \{\mathrm{x}\}\rangle\ ,\ \mathtt{maj}\langle \mathrm{id}, A, C \cup \{\mathrm{x}\}, r, M, d\rangle}{\mathtt{dest}\langle X\rangle\ ,\ \mathtt{dest}\langle A \cup C\rangle}$$

$$\text{(D-prop-dest)}\ \frac{\mathtt{dest}\langle X_1 \cup \{\mathrm{x}\}\rangle\ ,\ \mathtt{dest}\langle X_2 \cup \{\mathrm{x}\}\rangle}{\mathtt{dest}\langle X_1\rangle\ ,\ \mathtt{dest}\langle X_2\rangle}$$

Enfin vient la règle qui met fin à cette vague de suppressions, permettant à un agent de type `dest` de redevenir actif à nouveau. C'est le cas lorsque l'agent a contacté tous les agents contenus dans son unique site. Dans ce cas, il repart à l'état initial :

$$\text{(D-Fin)}\ \frac{\mathtt{dest}\langle \emptyset\rangle}{(\nu\,\mathrm{id})\,\mathtt{act}\langle \mathrm{id}, \emptyset, \emptyset, 1, G_i\rangle}$$

On peut finalement définir la micro-implémentation étendue qui correspond à cette implémentation du mécanisme de dislocation. Soit $\mathfrak{R}_i^{\mathtt{et}}(\mathcal{G})$ l'ensemble constitué de l'union de $\mathfrak{R}_i(\mathcal{G})$ avec les règles (D-Init), (D-prop-pass 2), (D-prop-maj 1), (D-prop-maj 2), (D-prop-dest) et (dest-Fin). Comme pour la macro-implémentation étendue, on définit le système $\mathtt{I}_{\mathtt{m}}^{\mathtt{et}}(\mathcal{G}, n)$ en se basant sur $\mathtt{I}_{\mathtt{m}}(\mathcal{G}, n)$ par :

$$\mathtt{I}_{\mathtt{m}}^{\mathtt{et}}(\mathcal{G}, n) = (\mathcal{T}_i, \mathfrak{n}_i, \mathcal{V}_i, \mathcal{C}, \mathfrak{R}_i^{\mathtt{et}}(\mathcal{G}), I_0^n)$$

On est à même maintenant, en se basant sur les résultats établis dans la section 4.2, de montrer que la micro-implémentation $\mathtt{I}_{\mathtt{m}}^{\mathtt{et}}(\mathcal{G}, n)$ est bisimilaire à la spécification $\mathtt{SPEC}(G_{\mathtt{F}}, n)$.

4.3.2 Correction complète

La correction de cet algorithme étendu est essentiellement basée sur le théorème 3 présenté dans la section 4.2.3. Nous n'allons pas re-développer ici toutes les preuves de la section précédente mais simplement donner les définitions nécessaires et reformuler les propriétés principales qui sont requises pour gérer cette nouvelle situation. Comme dans le chapitre 3, nous travaillerons sur l'ensemble $\mathcal{M}^{\mathtt{et}^+}$ et $\mathcal{I}_m^{\mathtt{et}^+}$ des états atteignables respectifs de $\mathtt{M}^{\mathtt{et}}(\mathcal{G}, n)$ et $\mathtt{I}_{\mathtt{m}}^{\mathtt{et}}(\mathcal{G}, n)$. Par ailleurs, nous noterons $\mathcal{D} = \mathfrak{R}_i^{\mathtt{et}}(\mathcal{G}) \setminus \mathfrak{R}_i(\mathcal{G})$ l'ensemble des règles gérant le mécanisme de désassemblage et $\mathcal{P}$ le sous-ensemble $\mathcal{D}\setminus\{(\mathsf{D\text{-}Init})\}$ qui ne s'occupe que de la propagation du message de dislocation.

L'aspect le plus important consiste à garder la même correspondance entre les réseaux $\mathtt{M}^{\mathtt{et}}(\mathcal{G}, n)$ et $\mathtt{I}_{\mathtt{m}}^{\mathtt{et}}(\mathcal{G}, n)$ aussi longtemps qu'aucun message de dislocation ne se propage dans une composante. Dès que cela se produit, nous considérons alors la composante comme totalement désassemblée, et nous l'assimilons à la composition des agents à l'état $\mathtt{act}\langle \mathrm{id}, \emptyset, \emptyset, 1, G_i\rangle$. Similairement à la définition 31 qui établissait le réseau propre d'un réseau de $\mathcal{I}_m^+$, nous définissons, pour chaque réseau I de $\mathcal{I}_m^{\mathtt{et}^+}$, son *état stable*, noté $\mathrm{ST}(I)$, obtenu en lui appliquant systématiquement toutes les règles de $\mathcal{P}$.

Définition 32 (État stabilisé) *Soit I un élément de $\mathcal{I}_m^{\mathtt{et}^+}$. On définit $\mathrm{ST}(I)$ comme le réseau vérifiant les conditions :*

- *$I \to^* \mathrm{ST}(I)$ en utilisant seulement les règles de $\mathcal{P}$*
- *$\forall \mathfrak{r} \in \mathcal{P}$, $\forall I'$, $\mathrm{ST}(I) \overset{\mathfrak{r}}{\not\to} I'$.*

Puisque le nom utilisé pour propager l'alarme est supprimé de l'ensemble X de l'unique site de l'agent `dest`, il vient que cet ensemble décroît et qu'un agent ne peut pas redevenir de type `dest` lorsque cette phase est terminée. Comme pour la définition 31, cette procédure sur les états de $\mathcal{I}^+$ est confluente. De plus, puisque l'alarme n'est déclenchée que par l'agent actif, il s'ensuit que si un agent est actif, aucun message de dislocation n'est en train de se propager dans la composante à laquelle il appartient. L'agent en a par conséquent toujours une image cohérente au sens de la définition 29. On en déduit que pour tout état $I \in \mathcal{I}_m^{\mathtt{et}^+}$, $\mathrm{ST}(I)$ satisfait les conditions de la proposition 3.

D'autre part, soit I un réseau tel que $\mathrm{RVT}(I)$ est un unique graphe recouvert pointé. Alors il est facile de caractériser l'état de $\mathrm{ST}(I')$ si I' est obtenu à partir de I par l'application de la règle $(\mathsf{D\text{-}Init})$. Il suffit pour cela de remarquer que, dès qu'une composante connexe a commencé à se disloquer, elle n'a d'autre choix que de continuer à supprimer toutes les connexions avant que les agents qui la composaient ne redeviennent à nouveau actifs.

Lemme 9 (Désassemblage) *Soit I un réseau tel que $\mathrm{RVT}(I)$ soit un* GARP *composé d'un seul graphe à n nœuds. Si $I \overset{(\mathsf{D\text{-}Init})}{\longrightarrow} I'$, alors $\mathrm{ST}(I') = \langle \mathrm{id}, \emptyset, \emptyset, 1, G_i\rangle_n$.*

Il ne reste plus qu'à exhiber une relation de bisimulation entre $\mathcal{M}^{\mathtt{et}^+}$ et $\mathcal{I}_m^{\mathtt{et}^+}$. On va se servir ici de la technique de bisimulation modulo, déjà utilisée au chapitre 2 pour montrer que les processus p et $[\![p]\!]_\pi$ étaient bisimilaires. Cette fois-ci, le rôle de $\mathcal{R}$ est joué par la relation $\approx_{\mathcal{G}}$ de la définition 30, dont nous avons déjà établi la principale propriété. La véritable relation de bisimulation correspondant à $\mathcal{R}_{\mathfrak{S}}$ reste ici à définir :

Définition 33 ($\approx_{\mathcal{G}}^{\mathtt{et}}$) *Soit $\approx_{\mathcal{G}}^{\mathtt{et}}$ la relation binaire définie sur $\mathcal{M}^{\mathtt{et}^+} \times \mathcal{I}_m^{\mathtt{et}^+}$ par :*

$$M \approx_{\mathcal{G}}^{\mathtt{et}} I \iff M \approx_{\mathcal{G}} \mathrm{ST}(I)$$

Munis des résultats de la section 4.2.3, il nous est maintenant possible de vérifier la proposition suivante :

Proposition 6 *Soient M et I tels que $M \approx_{\mathcal{G}} I$.*
- *Si $M \to M'$ alors il existe I' tel que $I \to I'$ et $M' \approx_{\mathcal{G}}^{\mathtt{et}} I'$*
- *Si $I \to I'$ alors il existe M' tel que $M \to M'$ et $M' \approx_{\mathcal{G}}^{\mathtt{et}} I'$*

Démonstration : Conséquence directe du théorème 3 et du lemme 9.

Ce qui nous permet d'étendre la portée du théorème 3 pour englober le cas des règles de dislocation :

Théoreme 4 (Correction) *Pour tout n et tout scénario de construction $\mathcal{G}$, les systèmes $\mathtt{M}^{\mathtt{et}}(\mathcal{G}, n)$ et $\mathtt{I}_{\mathtt{m}}^{\mathtt{et}}(\mathcal{G}, n)$ sont bisimilaires*

Démonstration : On montre que $\approx_{\mathcal{G}}^{\mathtt{et}}$ est une bisimulation. Soit $M \approx_{\mathcal{G}}^{\mathtt{et}} I$. Par les définitions 32 et 33, il vient que $I \to^* \mathrm{ST}(I)$ et $M \approx_{\mathcal{G}} \mathrm{ST}(I)$.

Supposons $M \to M'$: Alors par la proposition 6, nous pouvons conclure immédiatement que $\mathrm{ST}(I) \to I'$ avec $M' \approx_{\mathcal{G}}^{\mathtt{et}} I'$.

Supposons $I \to I'$: Deux cas sont possibles en fonction de la règle appliquée. Soit cette règle ne fait pas partie de l'ensemble $\mathcal{P}$ et on peut conclure en utilisant la confluence de $\mathcal{P}$ et en appliquant la proposition 6. Sinon, il vient que $\mathrm{ST}(I') = \mathrm{ST}(I)$ et il suffit de prendre $M' = M$ pour conclure.

En corollaire, on obtient la correction complète du problème d'auto-assemblage pour G.

Corollaire 5 *Pour tout n, tout graphe $G_{\mathtt{F}}$ et tout scénario de construction $\mathcal{G}$ ayant $G_{\mathtt{F}}$ comme graphe terminal, le système de spécification $\mathtt{SPEC}(G_{\mathtt{F}}, n)$ et le système d'implémentation $\mathtt{I}_{\mathtt{m}}^{\mathtt{et}}(\mathcal{G}, n)$ sont bisimilaires.*

Démonstration : On peut toujours simuler l'étape $\langle\emptyset\rangle_n \to G_{\mathtt{F}}$ en désassemblant autant de composantes non isomorphes à $G_{\mathtt{F}}$ que nécessaires pour utiliser les agents en fonction du chemin choisi dans le scénario $\mathcal{G}$ menant à $G_{\mathtt{F}}$.

Chapitre 5

Implémentation

Sommaire

5.1 Préambule

Après avoir élaboré un algorithme pour l'auto-assemblage de graphes, nous allons maintenant aborder son implémentation. Celle-ci, réalisée en Ocaml, permet de programmer l'assemblage décentralisé de graphes à l'aide des règles de la micro-implémentation étendue. La définition de ce système passait, on l'a vu, par une première phase décrivant les étapes élémentaires de construction du graphe final avant d'engendrer, dans un second temps, les règles qui y étaient associées. L'exécution du programme va naturellement refléter cette décomposition en deux étapes. Ainsi l'utilisateur se voit tout d'abord présenter une *fenêtre de description* dans laquelle il spécifie les éléments du scénario de construction $\mathcal{G}$ et, en particulier, le graphe final (ou les graphes finaux) qu'il désire assembler. Le programme génère ensuite toutes les interactions relatives à la propriété 2 de $\mathcal{G}$. Lorsque ce calcul est terminé l'utilisateur détermine l'état initial du système, c'est-à-dire simplement le nombre d'agents disponibles au départ. Enfin, il peut assister à une exécution des règles de $\mathfrak{R}_i^{\mathtt{et}}(\mathcal{G})$. Bien sûr, il est nécessaire que la description des sous-graphes autorisés produise bien un

scénario de construction vérifiant la propriété 2. Ceci est laissé à la seule charge de l'utilisateur et n'est pas testé par le programme.

Nous ferons désormais référence à la version présentée au chapitre 4 par l'appellation de *version abstraite*. Nous réserverons le terme *implémentation* au programme présenté dans le présent chapitre. D'autre part, nous ne fournirons que les extraits de codes qui nous semblent pertinents pour l'argumentation. Une partie plus complète et commentée du code sur lequel nous nous penchons peut toutefois se retrouver dans l'annexe A. La totalité du programme est quant à lui disponible sur Internet[1]. Nous nous intéressons ici principalement au rapport entre la version abstraite et la version implémentée.

Certaines caractéristiques de la version abstraite sont adaptées au cadre mathématique du chapitre précédent, mais s'accommodent très mal d'une implémentation. Cette dernière exige de définir constamment des priorités et de préciser l'ordre dans lequel les opérations s'appliquent. La notion d'appartenance à un ensemble est, par exemple, difficilement compréhensible par un langage de programmation si on ne lui indique pas la procédure à effectuer pour statuer sur cette question. Il faut donc s'appuyer sur les caractéristiques propres du langage utilisé pour traduire ces notions, tout en faisant attention à ce que ces choix de traduction ne faussent pas la bonne marche de l'algorithme. Ainsi, dans le cadre des ensembles utilisés dans l'interface des agents, il s'agit de noter que les opérations qui s'y appliquent ne sont qu'extractions et ajouts. Il suffit alors de définir un ordre dans l'extraction des éléments d'un ensemble, ce qui s'établit très bien à l'aide de la notion de liste, qui est un constructeur au cœur du langage Ocaml.

Dans le même ordre d'idées, si le nombre de règles générées par la traduction de $\mathfrak{R}_{\mathcal{G}}$ en $\mathfrak{R}_i(\mathcal{G})$ n'a aucune incidence sur l'algorithme lui-même dans sa version abstraite, il est indispensable de traiter ce point lors de la production du code simulant ces règles. Le mécanisme de *filtrage par motif* qui sous-tend Ocaml permet de lever en grande partie la difficulté liée à la profusion des règles. Mais il faut tout de même se pencher sur la question du rapport entre graphes concrets et graphes abstaits. Dans ce cadre ce sont les règles (JoncBin) et (JoncInt) qui posent le plus de problèmes, puisque la condition de bord qui les accompagne exige un test d'appartenance à une classe d'isomorphisme de graphes, ce que nous savons être très coûteux en nombre d'opérations.

En plus des problèmes soulevés par la traduction de l'algorithme en code Ocaml et des modifications liées à ces inadaptations - qui sont exposés dans la section 5.2 - la mise en place de l'implémentation s'accompagne de l'ajout à la version abstraite d'éléments totalement nouveaux - traités dans la section 5.3 - tels que l'extension spatiale et les modifications de l'algorithme qui en découlent. Une modélisation de l'espace qui entoure les agents a été in-

[1] `http://www.pps.jussieu.fr/~tarissan/self/implem_sag.tar.gz`

troduite, en équipant les agents de coordonnées cartésiennes. Ceci induit une restriction sur les possibilités de connexions, assujétissant les règles de communication entre agents à n'être possibles que si ceux-ci sont suffisamment « proches ». Une fois ce modèle établi, il devient naturel de l'utiliser pour traduire différemment les règles de désassemblage de la section 4.3 en se basant sur l'impossibilité de communiquer entre les agents pour propager le message de dislocation de la composante.

D'autre part, si le reste de l'algorithme s'accommode très bien du cadre non-déterministe du choix des règles, la décision de désassembler la composante demande, quant à elle, une gestion plus fine si l'on veut espérer obtenir l'assemblage du graphe final. D'où l'introduction de probabilités pour encadrer cette décision.

L'implémentation diffère donc de la version abstraite, en partie pour des raisons propres à la traduction d'un algorithme dans un langage de programmation (traduction de certaines notions abstraites et optimisations), mais aussi parce que nous allons un peu plus loin que ce que nous avons présenté dans la version abstraite, à savoir que l'on propose ici une modélisation de l'espace. Pour cette seconde catégorie de modifications nous aurons soin de vérifier que les propriétés qui ont été prouvées correctes au chapitre 4 sont toujours valides dans le cadre de ces modifications.

5.2 Une traduction pas si naturelle

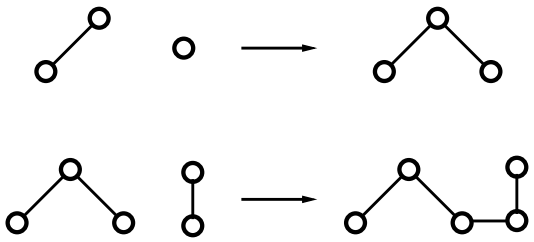
5.2.1 La représentation des graphes.

L'un des traits les plus importants concernant l'implémentation de l'algorithme est évidemment la gestion des graphes ou, plus précisément, les choix relatifs à leur représentation. De ces choix va énormément dépendre l'efficacité du code produit. Si certains aspects utilisés dans la version abstraite du problème convenaient bien au cadre purement théorique, il n'en est pas de même de la partie consacrée à l'implémentation proprement dite. Ainsi la règle (JoncBon) et sa variante interne (JoncInt), qui sont des règles clefs du système $\mathtt{I}_{\mathtt{m}}(\mathcal{G}, n)$ que nous cherchons à implémenter, précisent en prémisses que les graphes concrets détenus pas les agents appartiennent aux classes d'isomorphismes définies pas $\mathcal{G}$. Or les tests d'isomorphisme de graphes sont, dans le cas général, extrèmement coûteux. De plus, on voit bien qu'une partie non négligeable du système consiste en des allers retours entre création de connexions et destruction des composantes. On imagine aisément que les tests d'isomorphisme entre les mêmes graphes vont se répéter alors qu'il serait assez avantageux de pouvoir répondre à cette question une fois pour toutes.

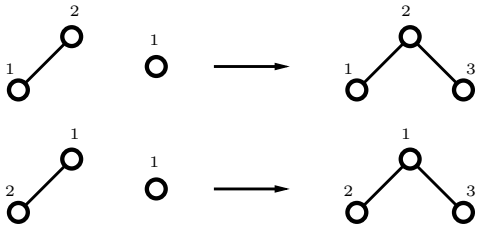
De ce point de vue il paraît donc naturel d'essayer d'optimiser cette règle d'interaction. C'est pourquoi nous avons fait en sorte que ce test d'appar-

tenance à une classe d'isomorphisme ait un coût fixe. Pour cela, nous avons fortement contraint la représentation des graphes concrets, à laquelle on associe le type Ocaml *graph*. Celui-ci consiste en un simple couple d'entiers, le premier identifiant le graphe auquel appartient l'agent actif et le second désignant le nombre de nœuds du graphe (information utile pour optimiser les recherches). Pour savoir si une connexion entre deux agents actifs est possible, on se réfère à une base de données générée au moment de la construction de l'ensemble $\mathcal{G}$.

La question de l'efficacité du code produit se reporte alors sur l'implémentation de cette base de données. En effet, l'optimisation apportée à la phase de communication proprement dite est contrebalancée par le fait que cette base de données peut être arbitrairement grande puisqu'elle contient, pour les couples de graphes dont les connexions sont possibles, tous les isomorphismes existants. Supposons par exemple que les scénarios de construction autorisent les deux jonctions binaires suivantes entre graphes abstraits :



Du point de vue de l'implémentation, la première de ces règles induit alors deux entrées dans la base de données des interactions entre graphes concrets, en fonction du nœud de la première composante à laquelle se lie l'unique nœud de la seconde composante :



Cette duplication de la représentation du graphe construit se répercute alors sur l'implémentation de la seconde des jonctions puisqu'il faut, non seulement gérer le nœud sur lequel va se créer la connexion, mais aussi la forme concrète qu'auront les graphes impliqués. Cela produit donc les quatre entrées de la figure 5.1.

Outre qu'il semble inutile d'avoir toutes ces représentations différentes pour une même structure, l'inconvénient majeur réside dans le fait que l'on va générer autant de nouveaux tests d'isomorphisme qu'il y a de graphes différents et donc augmenter d'autant le nombre d'entrées dans la base de données. En

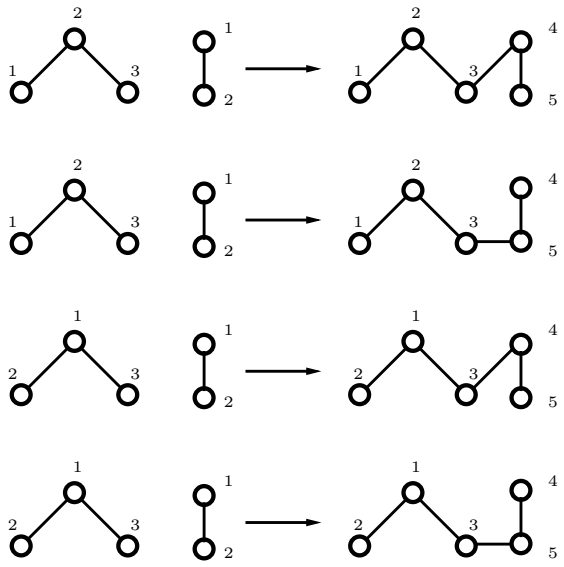


FIG. 5.1 – Exemple de la multiplication des entrées dans la base de données

termes de mémoire utilisée, ce choix semble donc aberrant. Cependant on obtient un bon équilibre grâce à la notion de graphe abstrait qui est supprimée, ou plus précisément à laquelle nous ajoutons la notion de *graphe canonique*, c'est à dire un graphe concret qui sert de référence pour la classe d'isomorphisme. Il va de soi que le choix de ce graphe de référence est totalement arbitraire. Il est effectué dans la phase d'élaboration du scénario de construction. Dans le cas précédent, on peut par exemple fixer la composante en forme de triangle de telle sorte que l'indice du nœud central soit 2. Ce qui entraîne la suppression des deux dernières entrées de la figure 5.1.

Un agent actif possède donc en définitive la référence au graphe (concret) canonique auquel il appartient. Le test qui autorise une communication avec un autre agent devient alors élémentaire, puisqu'il s'agit de l'appartenance d'un quadruplet `(graph,role,graph,role)` à une base de données. Ce qui est moins élémentaire en revanche, c'est la phase de mises à jour qui succède à la connexion si celle-ci est permise. En effet, le graphe canonique de la structure nouvellement constituée peut maintenant avoir modifié les rôles de tous les agents. La propagation de la mise à jour devient alors, non plus la transmission de l'entier représentant le nombre de nœuds de l'un des deux graphes, mais celle d'une fonction injective $f : \mathbb{N} \mapsto \mathbb{N}$ attribuant son nouveau rôle à chaque agent (identifié par son ancien rôle). Le fait d'utiliser des graphes concrets permet alors de représenter cette fonction par un tableau à deux dimensions (type ***`shift`*** dans le code), en utilisant l'indice du tableau comme ancien rôle de l'agent.

Autre conséquence directe de cette représentation imposée de la composante à laquelle un agent appartient, il est désormais nécessaire que les mises à jour se produisent dans les deux sous-composantes lors d'une jonction binaire, alors qu'elles n'étaient auparavant requises que dans la composante où le rôle des agents était décalé. De même, lors d'un jonction interne, il est maintenant impératif d'effectuer une mise à jour du rôle de tous les agents, tandis qu'aucune mise à jour n'était utile dans la version abstraite.

Le scénario de construction, représenté par le type `assembly` dans notre code Ocaml, est donc caractérisé par quatre paramètres : le graphe canonique et le rôle joué par l'agent actif à l'état initial, la liste des graphes finaux, la liste des quadruplets représentant les connexions possibles entre deux nœuds appartenant à deux graphes disjoints, et la liste des triplets désignant les connexions possibles entre deux nœuds d'un même graphe. Ces deux dernières listes sont accompagnées chacune du graphe canonique de la nouvelle structure et des mises à jour à propager dans la(les) composantes(s). Tout ceci se traduit concrètement par le code suivant :

```
TYPE assembly = {
  init     : graph * role ;   (* graphe initial *)
  final    : graph list;      (* liste de graphes finaux *)

  (* Liste des jonctions binaires autorisées *)
  binaire : ((graph * role *  graph * role) *
              (graph * (shift * shift))) list;

  (* Liste des jonctions internes autorisées *)
  interne : ((graph * role * role)
             * (graph * shift)) list
}
;;
```

5.2.2 Autres modifications mineures.

La représentation des agents. Comme précisé dans l'introduction, l'utilisation des listes se substitue désormais à la notion d'ensembles, au centre de la définition de l'interface des agents. La raison de ce choix est double. Tout d'abord, les listes s'accompagnent naturellement d'un ordre lorsque l'on cherche à parcourir l'ensemble des éléments appartenant à cette structure. Puis, surtout, les manipulations de listes sont au cœur du langage Ocaml. On gagne de ce fait une certaine aisance dans la formulation des opérations de base sur les interfaces (comparaisons, extractions, ajouts) ainsi qu'une indéniable efficacité due à l'intégration de ces opérations dans le langage.

Autres changements relatifs aux interfaces : le type d'un agent va cette fois être *intégré* à l'interface, en utilisant les spécificités propres aux différents types. Par exemple, l'agent actif étant le seul à posséder l'image du graphe,

cette information constitue l'attribut caractéristique d'un tel type d'agent. On introduit par ailleurs un nouveau type *Final* désignant l'état d'un agent lorsqu'il appartient à une composante représentant un graphe final. L'introduction de ce nouvel état implique alors un léger changement relatif aux agents en phase de mises à jour. Il est en effet nécessaire de mémoriser le fait que la composante nouvellement assemblée correspond à un graphe final, ce qui se représente très simplement à l'aide d'un booléen. À la fin de la mise à jour les agents prennent alors le type `Final` au lieu de `Actif` ou `Passif`.

Enfin, étant donné que les mises à jour ont lieu dans les deux composantes impliquées lors d'une jonction binaire, il est nécessaire de se souvenir de l'agent qui reste actif dans la composante après avoir transmis les nouvelles informations. Dans la version abstraite, la composante liée à l'agent qui restait actif ne recevait pas de mises à jour puisque les rôles restaient inchangés. Heureusement, puisque seul l'agent actif devra posséder le graphe concret de la composante, il suffit de réserver cette information au seul agent restant actif, ce qui se décrit aisément à l'aide de la construction *option* de Ocaml. Le type *state* dénotant l'état d'un agent se représente donc par le code :

```
(* Type de l'état d'un agent *)
TYPE state =
    Passive
  | Active OF graph
  | Update OF com list * shift * bool * graph option
  | Alarme OF float
  | Final
;;
```

Mis à part ces changements mineurs, l'interface des agents reste conforme à celle de la version abstraite. On remarque en particulier que les quatre sites spécifiant l'identifiant de groupe, les connexions liées à un agent voisin dans l'arbre de recouvrement, les autres connexions et le rôle joué par l'agent dans la composante, sont communs à tous les types d'agents de $\mathtt{I_m}(\mathcal{G}, n)$. Les autres informations sont stockées dans le dernier site, introduit ici pour mémoriser précisément le type de l'agent.

Déroulement du programme. Ces différents états se reflètent par ailleurs dans la description de l'état du système, puisque l'on classe dorénavant les agents en fonction de leur état courant. Ceci simplifie, là encore, la recherche d'un partenaire en fonction de la règle de communication utilisée (un agent actif si la règle est (JoncBin), un agent passif si c'est (JoncInt) ou (Maj)). De plus, cela convient à l'implémentation de la dynamique du système que nous avons choisie. Pour déterminer le prochain état du système, on établit aléatoirement la règle que l'on va tenter d'appliquer parmi celles de l'ensemble $\mathfrak{R}_i^{\mathtt{et}}(\mathcal{G})$. En fonction de ce choix, on désigne un agent (si c'est possible) dont le type correspond à celui de l'un des agents impliqués par la règle. Si la règle

est binaire, on sélectionne, si cela est réalisable, un second agent en fonction du type attendu. Il est donc extrêmement opportun de pouvoir les prendre au hasard dans une liste d'agents ayant le type désiré.

D'autre part, les informations sur l'ensemble des types $\mathcal{T}$, la fonction d'arité des types $\mathfrak{n}(\cdot)$ qui lui est associée, ainsi que sur les ensembles de valeurs et de noms $\mathcal{V}$ et $\mathcal{C}$ qui font partie de la spécification d'un système dans le cas général, ne sont pas, dans l'application qui nous intéresse, destinées à être changées ni paramétrées. Ces quatre éléments sont donc figés une fois pour toutes dans le programme et ne font plus partie de la définition du système proprement dite. Seul le scénario de construction reste un paramètre nécessaire. Ces éléments conduisent à la définition suivante du système :

```
(* Un système est controlé par six paramètres *)
TYPE system = {
  MUTABLE active   : agent list; (* agents actifs  *)
  MUTABLE update   : agent list; (* agents en phase de mise à
                                    jour *)
  MUTABLE passive  : agent list; (* agents en mode passif *)
  MUTABLE break    : agent list; (* agents en mode alarme *)
  MUTABLE final    : agent list; (* agents ayant atteint une
                                    formation stable *)

  assembly : assembly           (* scénario de construction *)
}
;;
```

5.3 Une modélisation de l'espace

On s'intéresse maintenant à la partie de l'implémentation qui diffère totalement de la version abstraite. L'aspect simulation lié à l'implémentation en Ocaml invite naturellement à munir les agents de coordonnées cartésiennes leur permettant à tout instant de connaître leur position vis-à-vis des autres agents. Il semble alors naturel de modifier les règles d'interaction de $\mathfrak{R}_i^{\mathtt{et}}(\mathcal{G})$ pour tenir compte de cette nouvelle caractéristique. Cela se traduit notamment par une restriction sur l'application des règles. Deux agents ne pourront dorénavant communiquer que si la distance les séparant est inférieure à une valeur fixée. Une seconde conséquence de cette extension est qu'il n'est plus assuré que toutes les règles aient la même chance de s'appliquer. Il faut donc compenser cette restriction par de la mobilité. On autorise alors les agents à se déplacer, favorisant ainsi la diversité des interactions.

5.3.1 Espace et mobilité

L'idée de base est donc de contraindre les règles d'interaction entre deux agents à ne s'appliquer que si la distance les séparant est suffisamment faible.

Soit d cette distance maximale autorisée, on munit tout agent a de deux coordonnées réelles cartésiennes $\mathtt{pos_x}$ et $\mathtt{pos_y}$ positionnant cet agent dans l'espace à deux dimensions. Nous ferons dorénavant référence à ces positions par $\mathtt{pos_x}(a)$ et $\mathtt{pos_y}(a)$.

On équipe par ailleurs les agents d'une *sphère de perception* permettant de sélectionner, dans le système, les autres agents avec lesquels ils peuvent communiquer. Soit $\mathsf{dt}(a_1, a_2)$ la distance séparant deux agents a_1 et a_2, on définit le *prédicat de perception* $\mathcal{P}(a_1, a_2)$ par :

$$\mathcal{P}(a_1, a_2) = vrai \iff \mathsf{dt}(a_1, a_2) \leq d$$

On conditionne alors l'application des règles (JoncBin) et (JoncInt), (Maj), (Pass) à la validité du prédicat $\mathcal{P}(a_1, a_2)$, où a_1 et a_2 sont les deux agents impliqués dans ces règles. Les règles de propagation du message de dislocation de la composante sont pour l'instant laissées de côté car, comme nous le verrons dans la section suivante, nous réservons un autre traitement au cas du désassemblage.

Du point de vue de la correction de l'algorithme, la restriction des règles n'a pas d'incidence puisque nous n'avons pas changé les effets des règles sur l'interface des agents. Par contre, il devient plus difficile de garantir que toutes les règles ont une chance de s'appliquer. Pour alléger cette contrainte, on dote les agents d'une certaine liberté de mouvement. Il faut faire attention, cependant, à ce que ces déplacements n'éloignent pas trop les agents qui sont liés dans une composante.

La dynamique du système est donc régie par le principe suivant : tous les agents impriment une force répulsive aux autres agents. Cette force est inversement proportionnelle au carré de la distance séparant les deux agents. Un agent a a donc tendance à s'éloigner de tous les autres agents perceptibles. Cette force est alors compensée par une force attractive, calculée à partir des agents qui lui sont liés. Le calcul de cette force attractive est basé sur le modèle d'un ressort, où une distance optimale d_{eq} entre les deux agents est recherchée. En dernier recours, toujours pour favoriser les interactions et la mobilité des agents, on imprime une force minimale $\overrightarrow{d_{\mathsf{min}}}$ au déplacement de a si celui-ci est trop faible.

Soit $\mathsf{depl}(a)$ le vecteur dénotant le dernier déplacement d'un agent a et soit $\mathcal{L}(a)$ l'ensemble des connexions de a. Soient k_{rep} et k_{att} les deux coefficients utilisés dans le calcul des forces de répulsion et d'attraction. Soit d_ϵ le déplacement minimal à effectuer pour un agent. Le prochain déplacement de a est déterminé par la procédure indiquée en figure 5.2, traduisant en terme algorithmique l'intuition présentée dans le précédent paragraphe.

1: POUR TOUT agent a FAIRE
2: POUR TOUT agent b FAIRE
3: $\mathsf{depl}(a) \leftarrow \mathsf{depl}(a) - k_{\mathsf{rep}} \cdot \frac{1}{\mathsf{dt}(a,b)^2} \cdot \overrightarrow{ab}$
4: SI $\mathcal{L}(b) \cap \mathcal{L}(a) \neq \emptyset$ ALORS
5: $k \leftarrow \frac{\mathsf{dt}(a,b)}{2} - d_{\mathsf{eq}}$
6: $\mathsf{depl}(a) \leftarrow \mathsf{depl}(a) + k_{\mathsf{att}} \cdot k \cdot \overrightarrow{ab}$
7: FIN SI
8: FIN POUR
9: SI $||\mathsf{depl}(a)|| < d_\epsilon$ ALORS
10: $\mathsf{depl}(a) \leftarrow \overrightarrow{d_{\mathsf{min}}}$
11: FIN SI
12: FIN POUR

FIG. 5.2 – Algorithme partiel calculant le prochain déplacement d'un agent

5.3.2 Gestion des impasses

Le cadre non-déterministe est, d'un point de vue expérimental, inadapté au problème. On comprend bien, en effet, qu'assembler des structures impliquant un nombre élevé d'agents est plus difficile que d'assembler de petites structures et requiert donc plus de temps et de tentatives pour créer les bonnes connexions. Il paraît donc naturel de favoriser les grosses structures dans la poursuite de l'assemblage et de laisser les petites structures se désassembler plus rapidement. Ceci peut se traduire aisément en incorporant des probabilités dans le tirage au sort de la règle qui va s'appliquer au système. Par exemple, si n est le nombre d'agents de la composante à un instant donné, on alloue à un agent actif la probabilité $\frac{1}{n^2}$ de déclencher la destruction de la structure au lieu d'effectuer une autre action. Ce rapport traduit le fait que plus la structure est grosse, plus elle est proche de l'assemblage final et donc plus elle a de chance d'y parvenir. Notons au passage que l'information sur le nombre d'agents formant une composante est aisément récupérable par l'agent actif puisqu'il possède l'image de sa composante (elle est d'ailleurs mémorisée en tant que second élément du couple formant l'image de type `graph`).

D'autre part, une fois déclenché le désassemblage de la composante, on peut s'appuyer sur l'ensemble des connexions $\mathcal{L}(a)$ d'un agent pour déterminer si la composante de a est en train de se désassembler. En effet, la ligne 3 de l'algorithme présenté figure 5.2 montre que l'agent a recherche les agents partageant une connexion avec lui afin de déterminer son prochain déplacement. Il suffit donc d'interpréter le fait que l'agent ne trouve pas son partenaire comme le message d'un désassemblage. Ce qui veut dire, entre autre, qu'il faut modifier la règle (D-init). Lorsqu'un agent actif déclenche une dislocation, il oublie simplement en une seule étape toutes les connexions qu'il avait créées :

$$(\textsf{D-init '})\ \frac{\texttt{act}\langle \mathrm{id}, A, C, r, G\rangle}{\texttt{dest}\langle\emptyset\rangle}$$

Les agents partageant les connexions de $A \cup C$ percevront ce message lorsqu'ils tenteront de déterminer les forces d'attractions. Dans l'état suivant du système, tous les agents qui étaient auparavant connectés à l'agent actif vont ainsi être au courant de la destruction de la composante. Ce message va donc se propager de façon concentrique (du point de vue de la topologie des connexions) à partir de l'agent actif. La figure 5.3 montre comment on étend l'algorithme de la figure 5.2 pour tenir compte du cas de désassemblage.

1: POUR TOUT agent a FAIRE
2: POUR TOUT agent b FAIRE
3: $\mathsf{depl}(a) \leftarrow \mathsf{depl}(a) - k_{\mathsf{rep}} \cdot \frac{1}{\mathsf{dt}(a,b)^2} \cdot \overrightarrow{ab}$
4: SI $\mathcal{L}(b) \cap \mathcal{L}(a) = \{\mathrm{x}\}$ ALORS
5: $k \leftarrow \frac{\mathsf{dt}(a,b)}{2} - d_{\mathsf{eq}}$
6: $\overrightarrow{v} \leftarrow k_{\mathsf{att}} \cdot k \cdot \overrightarrow{ab}$
7: $\mathsf{depl}(a) \leftarrow \mathsf{depl}(a) + \overrightarrow{v}$
8: $\mathsf{depl}(b) \leftarrow \mathsf{depl}(b) - \overrightarrow{v}$
9: $\mathcal{L}(a) \leftarrow \mathcal{L}(a) \setminus \{\mathrm{x}\}$
10: $\mathcal{L}(b) \leftarrow \mathcal{L}(b) \setminus \{\mathrm{x}\}$
11: FIN SI
12: FIN POUR
13: SI $||\mathsf{depl}(a)|| < d_\epsilon$ ALORS
14: $\mathsf{depl}(a) \leftarrow \overrightarrow{d_{\mathsf{min}}}$
15: FIN SI
16: SI $\mathcal{L}(a) \neq \emptyset$ ALORS
17: $a \leftarrow \texttt{dest}\langle\emptyset\rangle$
18: FIN SI
19: FIN POUR

FIG. 5.3 – Algorithme complet calculant le prochain déplacement d'un agent

Notons pour finir que les propriétés importantes de la section 4.3 du chapitre 4 sont toujours valides : c'est toujours l'agent actif qui déclenche la destruction de sa composante ; un agent de type `dest` ne redevient actif qu'une fois toutes ses connexions perdues ; cet agent transmet bien le message de désassemblage à tous ses voisins – la seule différence étant que cette étape est maintenant atomique au lieu de demander autant de règles (D-prop) que d'agents liés.

5.4 Utilisation du programme

Les explications que nous venons de fournir concernant l'implémentation ne représentent en réalité qu'une petite partie du programme exécutable que nous avons réalisé. Cette section a pour but d'expliquer brièvement comment utiliser le programme en entier.[2]

Ce programme est téléchargeable à l'adresse `http://www.pps.jussieu.fr/~tarissan/self/implem_sag.tar.gz`. Une fois le fichier récupéré et désarchivé, il est possible de compiler[3] le code source à l'aide de la commande `make` se référant au fichier `Makefile` situé dans le répertoire principal. Ceci crée un fichier exécutable `main`.

Il existe deux moyens d'exécuter ce programme en fonction du choix du déroulement de la première phase, à savoir la description du scénario de construction. Soit l'utilisateur charge un fichier contenant déjà une telle description – ces fichiers se trouvent dans le dossier nommé `test` – soit il réalise lui-même cette description. La première option est réalisée grâce à l'instruction `./main -l test/fichier` où `fichier` est le nom du fichier choisi. Une première fenêtre présente alors les graphes terminaux (qui sont distingués pas l'attribut *final* en bas de la fenêtre) et l'ensemble des sous-graphes autorisés pour la construction. Tous ces éléments sont consultables en entrant le numéro du graphe que l'on désire afficher[4].

La deuxième option nécessite de décrire les graphes du scénario de construction. Ceci se fait à l'aide de la souris. En cliquant dans un endroit de la fenêtre, on crée un nœud. Pour relier deux nœuds déjà créés, il suffit de cliquer sur le premier nœud et de déplacer la souris (en maintenant le bouton enfoncé) jusqu'au second nœud. Lorsque la structure est totalement décrite, il reste à la valider à l'aide, soit de la touche `<entrée>` si c'est un sous-graphe partiel de l'assemblage, soit de la touche `<f>` si c'est un graphe terminal.

À tout moment, il est possible de consulter les graphes précédemment décrits en tapant le numéro représentant le graphe décrit, ainsi que d'effacer celui-ci en pressant la touche `<e>`. Lorsque tous les graphes du scénario de construction sont décrits, il suffit de valider l'ensemble à l'aide de la touche `<entrée>`. Le programme génère alors les règles conformément à l'algorithme présenté dans le chapitre 4 et qui se trouve modifié par les ajouts proposés dans le présent chapitre. Une deuxième fenêtre apparait alors dans laquelle on peut

[2] Il est par ailleurs possible d'observer l'évolution de deux exemples d'assemblage, sans télécharger le code source, en se rendant à l'adresse `http://www.pps.jussieu.fr/~tarissan/self/index.html`.

[3] Ceci requiert l'installation au préalable du *système Objective Caml* dont on trouvera une description et les fichiers concernant son installation à l'adresse `http://caml.inria.fr/ocaml/`.

[4] Les graphes du scénario de construction sont classés par nombre de nœuds puis par nombre d'arcs si les nombres de nœuds de deux graphes différents sont égaux. Le graphe minimal possède le nombre 1.

spécifier le nombre d'agents présents dans le système de départ (un clic pour chaque agent). La pression d'une touche permet ensuite au programme d'appliquer les règles d'interaction sur ce système. À tout moment, il est possible de rajouter un agent en cliquant sur un endroit de la fenêtre, de suspendre l'exécution du programme en appuyant sur la touche `<p>`, ou bien d'interrompre l'exécution du programme en pressant la touche `<q>`.

L'affichage graphique de l'évolution du programme comporte quelques informations visuelles. Tout d'abord, le type d'un agent est représenté par une couleur différente pour chacun des états. L'agent actif est représenté en rouge, un agent en phase de mise à jour en vert, un agent en phase de dislocation en bleu et un agent appartenant à une structure assemblée en jaune. Les agents passifs (majoritaires) n'ont pas de couleur associée. Les connexions sont elles aussi représentées différemment en fonction de la règle qui a été appliquée pour les créer. Si cette règle est (JoncBin), la connexion est représentée en rouge, si celle-ci est (JoncInt), elle est en bleu. L'ensemble des connexions représentées en rouge désignent donc l'arbre de recouvrement de la composante connexe. Il est alors possible de vérifier visuellement deux affirmations qui ont été faites précédemment. Tout d'abord ces connexions définissent bien un arbre. Ensuite, cette structure de recouvrement est bien une structure dynamique, créée lors des interactions entre les agents et donc dépendante de la succession des interactions qui a eu lieu dans la composante connexe. En particulier, on pourra observer que deux graphes isomorphes ne possèdent pourtant pas forcément deux arbres de recouvrement isomorphes. Ceci indique que deux chemins différents ont été choisis pour réaliser l'assemblage.

Une dernière constation relative aux propriétés du scénario de construction peut être faite à l'aide des exemples présents dans le dossier `test`. Nous avons expliqué, en conclusion du chapitre 3, que l'un des raffinements apporté par les règles d'assemblage présentées dans le chapitre 4 permettait à la description du scénario de construction de contraindre la construction du graphe terminal. Ceci peut se vérifier en comparant l'évolution de systèmes basés sur les scénarios `etoile5` et `etoile5cercle` du dossier `test`. Ces deux scénarios de construction cherchent à assembler le même graphe (une étoile). La différence tient au fait que, dans le premier cas, le scénario de construction est saturé par la description de tous les sous-graphes possibles alors que, dans le deuxième cas, on contraint l'assemblage à passer par une forme intermédiaire (un cercle) avant d'aller plus avant dans la construction.

Chapitre 6

Une extension pour la biologie moléculaire

Sommaire

6.1 Motivations

L'un des problèmes majeurs liés à la biologie moléculaire est de réussir à extraire le sens fonctionnel de la masse de données actuelles. Ce problème plaide pour le développement d'outils spécifiques qui puissent décrire la biologie de manière convenable. Parmi ces outils, l'utilisation des algèbres de processus, bien que récente, s'est révélée suffisamment riche pour pouvoir formaliser un certain nombre d'activités importantes des systèmes biologiques et rendre compte naturellement de la nature massivement parallèle et concurrente des interactions moléculaires, ainsi que pour analyser le comportement général de ces

systèmes. Deux caractéristiques, inhérentes aux systèmes qui nous intéressent, ont notamment conduit à cette approche. Tout d'abord, le fait que, dans les algèbres de processus, la syntaxe même des termes spécifie leur capacité d'interaction avec leur environnement en fait un langage proche de la biologie, dans laquelle la structure de l'objet étudié a des répercussions directes sur son activité. Ainsi, la structure tridimensionnelle d'une protéine ou encore sa définition en tant que suite d'acides aminés sont directement liées à sa capacité d'interaction avec les autres protéines. Bien que cette caractéristique ait certainement été à l'origine de l'attrait des algèbres de processus pour la biologie, nous nous sommes éloignés, comme nous l'avons déjà vu, d'une telle dépendance stricte entre définitions syntaxiques et règles d'interaction puisque ces deux notions sont formellement séparées dans notre langage.

Le deuxième argument en faveur de cette approche repose sur le fait que, afin d'obtenir une modélisation robuste d'un système biologique, il semble important que le modèle qui est construit reste *ouvert*. Plus précisément, il est essentiel, et ce dans un soucis de réutilisabilité, qu'un modèle soit autonome vis-à-vis des autres modèles qui pourraient en dépendre et qu'il soit capable de modifier son niveau de description en fonction des nouvelles données qui sont découvertes ou des besoins liés à une recherche particulière. De ce point de vue, les algèbres de processus offrent un cadre naturel pour rendre compte de cette modularité.

Deux développements différents, basés sur les algèbres de processus, ont apporté des résultats variés et intéressants. Une première approche, basée sur le π-calcul [Mil99], et suivant en ce sens les principes proposés dans [RSS01], utilise le partage de noms pour représenter les connexions entre protéines. Cette approche a notamment donné lieu au développement du κ-calcul [DL04], langage à même de décrire les interactions entre protéines au niveau des domaines. Ce langage s'est révélé particulièrement pertinent pour décrire des modèles de voies de transduction du signal ou de régulation de réseaux génétiques. Une autre famille de langages, les *calculs de membranes* [Car05], poursuivant les idées avancées par Paun [Pau02] puis par Regev *et al.* [RPS$^+$04], repose sur les Ambiants Mobiles [CG00] et utilise le principe de complémentarité entre actions et co-actions localisées sur la surface de la membrane des cellules. De tels langages ont prouvé leur adéquation pour représenter aussi bien le transport moléculaire que les infections virales. Comme ces deux familles de langages se sont avérées être deux paradigmes très différents, il semblait avantageux de construire un cadre formel unifié, capable de manipuler les deux types de modèles. C'est le but de ce chapitre. En particulier, le défi consiste à isoler quelques primitives d'interaction basiques qui permettent de décrire des systèmes utilisant les mécanismes du κ-calcul et des calculs de membranes.

Dans cette perspective, étant donné que les primitives utilisées dans les calculs de membranes s'abstraient des descriptions moléculaires à l'origine de

leur actions, il était naturel de se baser sur le κ-calcul pour bâtir ce langage unifié. D'autre part, parmi les questions qui ont émergé de [DL04], l'une d'elles consistait à donner une traduction de ce langage dans le π-calcul. Cette question n'est pas sans rappeler le travail sur l'auto-assemblage qui a été présenté jusqu'à présent, puisqu'il s'agit d'être capable de décomposer une interaction entre plusieurs protéines en une suite d'interactions élémentaires (entre deux protéines) coordonnées pour réaliser l'interaction de haut-niveau. La solution apportée dans [DL04] passe par une phase d'exploration partielle du graphe de voisinage des connexions des protéines pour implémenter ces interactions. Afin de faciliter la traduction du κ-calcul dans le π-calcul, un langage intermédiaire, le $\mathtt{m}\kappa$-calcul, est défini. Il se présente comme un fragment du κ-calcul (et donc du $g\kappa$-calcul) dans lequel seules les interactions binaires sont autorisées. Par ailleurs, une contrainte de monotonicité (d'inspiration biologique) est posée sur les règles d'interaction. Le présent travail reprend ce langage, pour en étendre la portée, en termes de modèles pouvant y être décrits. Plus précisément, nous cherchons ici à intégrer les modèles bâtis sur les primitives du Brane-calcul.

Ce nouveau langage, que nous nommons $\mathtt{bio}\kappa$-calcul, se base essentiellement sur les complexations et décomplexations de deux protéines. Ces interactions suivent en réalité le même schéma que celles du $\mathtt{m}\kappa$-calcul. De plus, le $\mathtt{bio}\kappa$-calcul ajoute la possibilité de distinguer des *compartiments* dans lesquels ont lieu les interactions, représentant ainsi les cellules. Cette particularité nous permet alors de décrire avec précision les interactions entre protéines lorsque l'une d'entre elles est située *dans* la membrane entourant le contenu de la cellule tandis que l'autre se trouve à l'intérieur ou à l'extérieur de la cellule. Dans de tels cas, un effet additionnel peut se produire : l'interaction change la capacité de la membrane, préparant ainsi la cellule à des interactions plus complexes, comme la fusion de membranes ou l'endocytose.

Les fusions entre membranes permettent d'ouvrir le contenu d'une cellule à une autre cellule. En particulier, le *cytoplasme*, dont les interactions avec la solution extérieure étaient jusque-là médiatisées par la membrane, peut directement interagir avec une autre solution (le cytoplasme contenu par l'autre membrane fusionnée) après la fusion. Pour représenter la fusion, on utilise un mécanisme similaire à celui défini dans le *π-calcul d'ordre supérieur* [San93].

D'autres événements pertinents impliquant les membranes sont aussi considérés, comme les translocations, qui permettent le transport de matériel protéique à l'intérieur de la cellule, et l'endocytose, qui permet à une cellule d'ingérer du matériel extérieur à celle-ci. Dans le processus d'endocytose, le matériel entrant est enveloppé dans une membrane qui est extraite de la cellule hôte. Cette extraction est particulièrement difficile à formaliser puisqu'elle demande de vérifier au préalable que la membrane de la cellule hôte possède suffisamment de protéines pour la nouvelle membrane.

Le travail présenté ici nécessite par ailleurs une adaptation de la sémantique

opérationnelle de notre langage. En effet, un des aspects qui nous intéresse plus particulièrement, et qui est proche de l'utilité pratique que nous dédions à ce travail, consiste à comparer les modèles décrits à l'aide de ce langage. La comparaison a pour but le raffinement du modèle de base ou, au contraire, son abstraction. En d'autres termes, nous aimerions, à partir d'un modèle de bas-niveau comportant des événements élémentaires et détaillés, nous en abstraire pour arriver à un modèle moins élaboré, et donc moins proche de la réalité moléculaire sous-jacente, mais dont le sens et le fonctionnement sont alors plus compréhensibles. C'est en quelque sorte le cheminement inverse de celui de l'auto-assemblage où nous partions d'une spécification haut-niveau pour rechercher une implémentation bas-niveau réalisant cette spécification. Ici nous possédons *déjà* l'implémentation, que nous pouvons observer via des technologies diverses, et c'est justement la spécification de ce système que nous recherchons, en terme de description *fonctionnelle*.

Les solutions biologiques sont donc modélisées en bioκ-calcul par des systèmes de transitions étiquetées dans lesquels les étiquettes portent l'information sur les éléments qui interagissent et sur la règle utilisée. Il est bien connu que de tels systèmes de transitions sont des objets trop *intentionnels* et les équivalences ont alors pour rôle de quotienter ces systèmes. Dans notre approche, nous utilisons la bisimulation faible [Mil89b] et démontrons que celle-ci est une congruence dans le bioκ-calcul : deux systèmes biologiques sont faiblement bisimilaires s'ils ont le même comportement lorsqu'ils sont plongés dans le même environnement.

En dépit de la simplicité des interactions du bioκ-calcul (interactions binaires entre protéines ou entre membranes), le langage est suffisamment expressif. Nous discutons en détail deux exemples biologiques importants : la cascade RTK-MAPK et une infection virale. Cependant, le but de bioκ-calcul est d'être un langage de base pour la biologie moléculaire, que l'on espère étendre ensuite pour englober des mécanismes biologiques plus complexes.

La prochaine section définit une version restreinte du bioκ-calcul, dans laquelle les mécanismes d'interaction sont limités à deux protéines. La section 6.3 étend ensuite les mécanismes d'interaction basiques aux cas des fusions, entraînant alors des modifications d'ordre structurel dans l'organisation hiérarchique des systèmes. La section 6.4 discute d'autres extensions du langage pour tenir compte des translocations et des mécanismes de phagocytose. Enfin, la section 6.5 tire quelques conclusions de cette approche et évoque la possibilité de futurs travaux[1].

[1] Ce chapitre est tiré de la publication [LT06] qui a été étendue pour le présent travail

6.2 Un fragment simple du bioκ-calcul

Dans cette section nous présentons seulement un fragment de bioκ-calcul pour lequel les interactions ne changent pas l'organisation hiérarchique des solutions. Nous définissons la syntaxe et la sémantique opérationnelle, ainsi que la bisimulation faible associée. Nous analysons aussi son expressivité en modélisant la voie de signalisation RTK-MAPK.

6.2.1 Syntaxe et sémantique

Notations préliminares. Un troisième ensemble dénombrable va désormais être utilisé en plus de l'ensemble des types $\mathcal{T}$, contenant l'ensemble des *noms de protéines*, et de l'ensemble des *noms de connexions* $\mathcal{C}$. Nous cherchons à encapsuler les agents, renommés *protéines* dans ce contexte biologique, à l'intérieur de compartiment. Pour les besoins des interactions complexes qui seront décrites dans la section 6.3, nous distinguerons, comme pour les protéines, différents types de membranes. L'ensemble des *noms de membranes* est défini par un ensemble dénombrable $\mathcal{M}$ supposé disjoint de $\mathcal{T}$ et $\mathcal{C}$ et dont les éléments sont dénotés par M, N, ... Les noms de protéines sont, comme auparavant, classés en fonction du nombre de *sites* qu'elles possèdent. Soit $\mathfrak{n}(\cdot)$ la fonction associant le nombre de sites pour un nom de protéine donné. Les sites d'une protéine sont indiqués par un entier naturel dans l'ensemble $\{1, \cdots, \mathfrak{n}(\mathtt{a})\}$.

Reste à décrire l'ensemble $\mathcal{V}$ des valeurs que peuvent prendre les sites des protéines et, plus généralement, la forme des interfaces. Les modélisations qui se sont révélées les plus pertinentes pour la biologie moléculaire distinguent en général trois états possibles. Les sites peuvent être soit *liés* à un autre site, partageant alors un élément de $\mathcal{C}$ avec cet autre site, soit *visibles*, *i.e.* non connectés à un autre site, ce qu'on représentera par la valeur V, soit *cachés*, c'est-à-dire inutilisables pour une interaction avec le site d'une autre protéine et dénoté ici par la valeur C. Comme nous le voyons la possibilité que les sites contiennent un ensemble de valeurs ou un ensemble de noms n'est pas nécessaire ici. Ceci simplifie de façon substantielle la description des interfaces : nous représenterons désormais l'état d'un site s par s^{x} s'il est lié par le nom x, par $\overline{s}$ si c'est un site caché et simplement par s si c'est un site visible.

Une autre simplification guidée par la biologie va faciliter la description des règles d'interaction. Contrairement au cadre général dans lequel nous avons défini notre langage, les protéines ne sont pas amenées à changer de type. Une protéine a reste invariablement de type a ; seule son interface change au cours de l'évolution du système. Dès lors, il devient possible de réduire le nombre d'informations nécessaires à la description d'une règle d'interaction pour se limiter à la partie de l'interface qui est modifée par la règle[2]. Il s'ensuit que

[2]Ceci est à mettre en relation avec la remarque faite dans la section 5.1 sur le nombre important de règles qui sont générées lorsque l'on ne dispose pas d'un mécanisme de filtrage

l'utilisation des interfaces partielles va prendre une importance accrue dans ce chapitre. Nous réserverons désormais les lettres ϕ, ψ pour les interfaces partielles et utiliserons σ, σ' pour les interfaces totales. D'autre part, mis à part la création ou la suppression d'une connexion entre deux protéines, les complexations et décomplexations vont avoir pour effet de modifier les capacités d'interactions d'une protéine. Étant donné que l'état d'un site qui n'est pas lié est soit visible soit caché, ces modifications ne portent donc que sur la permutation entre ces deux états. Nous appelons *interface permutable* toute interface partielle pour laquelle l'état des sites est soit visible soit caché et utilisons les même lettre ϕ, ψ pour dénoter de telles interfaces. Nous définissons ensuite l'opération de permutation sur ces interfaces, noté $\bar{\phi}$ par :

$$\overline{\phi}(s) \;=\; \begin{cases} \text{H} & \text{si } \phi(s) = \text{V} \\ \text{V} & \text{si } \phi(s) = \text{H} \end{cases}$$

Par ailleurs, afin de ne pas surcharger l'utilisation du signe ',' nous utiliseront l'opérateur '+' pour séparer les sites dans les interfaces.

Définition 34 (La syntaxe du $\texttt{bio}\kappa$-calcul retreint) *La syntaxe de* $\texttt{bio}\kappa$*-calcul définit les* solutions *(restreintes) biologiques* S *par :*

$$\mathsf{S} ::= \underset{\textit{(vide)}}{\emptyset} \;\mid\; \underset{\textit{(protéine)}}{\mathtt{a}\langle\sigma\rangle} \;\mid\; \underset{\textit{(cellule)}}{\text{M}(\!|\,\mathsf{S}\,|\!)[\mathsf{S}]} \;\mid\; \underset{\textit{(groupe)}}{\mathsf{S}\,,\,\mathsf{S}}$$

Une solution est soit vide, soit consituée d'une *protéine* $\mathtt{a}\langle\sigma\rangle$ indiquant son nom et son interface, soit une *cellule* $\text{M}(\!|\,\mathsf{S}\,|\!)[\mathsf{T}]$, c'est-à-dire une solution T, appelée *cytoplasme*[3], entourée par une autre solution S, appelée *membrane*, soit un *groupe* de solutions S , T. Soit $\mathtt{nb}(S, \text{x})$ la fonction qui retourne le nombre de sites s de S de la forme s^x. Trois fonctions auxiliaires sont définies sur les solutions et les interfaces. La fonction $\mathtt{nc}(\cdot)$ retourne l'ensemble des *noms de connexions* apparaissant dans l'argument de la fonction ; la fonction $\mathtt{np}(\cdot)$ retourne l'ensemble des *noms pendants* de son argument, à savoir tous les noms apparaissant exactement une fois ; la fonction $\mathtt{nlk}(\cdot)$ retourne l'ensemble des *noms liés* de l'argument, c'est-à-dire ceux qui apparaissent au minimum deux fois. Par exemple, si nous prenons la solution $\mathsf{S} = \text{M}(\!|\,\mathtt{c}\langle 1^{\text{y}} + 2\rangle\,|\!)[\mathtt{a}\langle 1^{\text{x}} + 2 + 3\rangle\ ,\ \mathtt{b}\langle 1 + 2^{\text{x}}\rangle\,]$, l'ensemble $\mathtt{nc}(\mathsf{S})$ est $\{\text{y}, \text{x}\}$ et l'ensemble $\mathtt{np}(\mathsf{S})$ est $\{\text{y}\}$.

Dans le reste du chapitre, nous identifierons les solutions qui sont égales à un renommage près des noms de connexion liés (ce qui correspond à l'α-conversion) et nous supposerons que les solutions vérifient les conditions de *bonne*

par motif.

[3]Nous désignons en fait par le terme générique de « cytoplasme » tout matériel entouré d'une structure interdisant les interactions avec le mileu extérieur. Ainsi le noyau d'une cellule sera représenté comme un cytoplasme, tout comme un brin d'ADN entouré d'une capside.

formation, à savoir que les noms de connexions apparaissent au plus deux fois (condition 1), que toute membrane est un groupe de protéines et ne contient par conséquent pas de cellules (condition 2), et que les connexions pendantes du cytoplasme sont connectées à des protéines localisées dans la membrane qui l'entoure (condition 3).

Propriété 3 (Bonne formation) *Une solution* S *est dite* bien formée *si elle vérifie les conditions suivantes :*

1. (condition sur les connexions) *Tout* x *de* $\mathtt{nc}(\mathsf{S})$ *est tel que* $\mathtt{nb}(\mathsf{S}, \mathrm{x}) \leq 2$
2. (condition sur les membranes) *Pour toute cellule* $\mathrm{M}(\!|\, \mathsf{S}' \,|\!)[\mathtt{T}]$ *de* S*, on a* $\mathsf{S}' \equiv \prod_{i \in 1 \ldots n} \mathtt{a_i}\langle \sigma_i \rangle$*, où les* $\mathtt{a_i}\langle \sigma_i \rangle$ *désignent des protéines*
3. (condition sur les cytoplasmes) *Pour toute cellule* $\mathrm{M}(\!|\, \mathsf{S}' \,|\!)[\mathtt{T}]$ *de* S*, on a* $\mathtt{np}(\mathsf{T}) \subseteq \mathtt{np}(\mathsf{S}')$

La solution $\mathrm{M}(\!|\, \mathtt{c}\langle 1^{\mathrm{x}} + 2 \rangle \,|\!)[\mathtt{a}\langle 1^{\mathrm{x}} + 2 + 3 \rangle \ , \ \mathtt{b}\langle 1 + 2^{\mathrm{x}} \rangle \,]$ ne vérifie donc pas les conditions de bonne formation sur les connexions puisque le nom x présente trois points d'ancrage différents (on retrouve la capacité du langage à représenter des objets dont les connexions sont décrites par des hypergraphes). La solution $\mathrm{M}(\!|\, \mathtt{b}\langle 1 + 2 \rangle \ , \ \mathtt{c}\langle 1 + 2 \rangle \,|\!)[\mathtt{a}\langle 1 + 2^{\mathrm{x}} + 3 \rangle \,]$ ne vérifie pas, quant à elle, la condition de bonne formation sur les cytoplasmes car $\mathtt{a}\langle 1 + 2^{\mathrm{x}} + 3 \rangle$ possède une connexion qui n'est pas rattachée à sa membrane. Dans ce qui suit, les solutions qui sont des membranes seront représentées par les lettres M, N, ...

Interactions biologiques. Les interactions biologiques qui sont considérées dans cette section sont de deux types : les *complexations*, qui créent une connexion entre deux proteines éventuellement disjointes, et les *décomplexations*, qui suppriment une connexion. Un exemple de complexation pourrait être (nous supposons que $\mathtt{n}(\mathtt{a}) = \mathtt{n}(\mathtt{b}) = 3$) :

$$\mathtt{a}\big\langle 1^{\mathrm{x}} + 2 + \overline{3} + 4 \big\rangle \ , \ \mathtt{b}\big\langle 1 + \overline{2} + \overline{3} \big\rangle \rightarrow \mathtt{a}\langle 1^{\mathrm{x}} + 2^{\mathrm{y}} + 3 + 4 \rangle \ , \ \mathtt{b}\big\langle 1^{\mathrm{y}} + 2 + \overline{3} \big\rangle$$

Celle-ci crée une connexion dénotée par y entre le site 2 de a et le site 1 de b. Ces deux sites, pour que l'interaction puisse effectivement avoir lieu, doivent être visibles. Cela signifie que l'application d'une complexation doit regarder au préalable si les sites concernés sont visibles ou non. L'interaction décrite ci-dessus ne peut donc pas s'appliquer sur le groupe $\mathtt{a}\big\langle 1^{\mathrm{x}} + \overline{2} + \overline{3} + 4 \big\rangle$, $\mathtt{b}\big\langle 1 + \overline{2} + \overline{3} \big\rangle$ car le site 2 de a est caché. Les règles d'interaction de bioκ-calcul peuvent aussi changer l'état des sites des protéines en permutant les sites cachés en sites visibles et réciproquement. Dans l'exemple ci-dessus c'est ce qui se produit pour le site 3 de a et le site 2 de b. Cette interface partielle sert donc à la fois de test pour savoir si l'interaction est réalisable mais aussi de description de l'effet qu'a l'interaction sur l'état des protéines. Ce sont pourtant a priori

deux missions différentes. Il semble donc naturel de rajouter la possibilité de décrire une partie de l'interface servant de test mais restant inchangée par l'interaction. C'est le cas du site 4 de $\mathtt{a}$ si l'on suppose que l'interaction est impossible quand ce site est caché. Une manière concise de représenter ces interactions consiste donc dans le schéma

$$\mathfrak{r} : \quad ((\mathtt{a}, 2, \overline{3}, 4), (\mathtt{b}, 1, 2, \emptyset))$$

qui rend explicite le nom des protéines qui interagissent – les deux premiers éléments des triplets – les sites d'ancrage du nouveau nom créé – les seconds éléments – et la partie de l'interface des protéines qui doit être testée, en différenciant la sous-partie qui est modifiée de celle qui reste inchangée – troisièmes et quatrièmes éléments. La règle $\mathfrak{r}$ peut donc aussi s'appliquer sur le groupe $\mathtt{a}\langle 1 + 2 + \overline{3} + 4\rangle$, $\mathtt{b}\langle 1 + \overline{2} + 3\rangle$ ou encore $\mathtt{a}\langle \overline{1} + 2 + \overline{3} + 4\rangle$, $\mathtt{b}\langle 1 + \overline{2} + 3^{\mathrm{x}}\rangle$ produisant les solutions $\mathtt{a}\langle 1 + 2^{\mathrm{y}} + 3 + 4\rangle$, $\mathtt{b}\langle 1^{\mathrm{y}} + \overline{2} + 3\rangle$ et $\mathtt{a}\langle \overline{1} + 2^{\mathrm{y}} + 3 + 4\rangle$, $\mathtt{b}\langle 1^{\mathrm{y}} + \overline{2} + 3^{\mathrm{x}}\rangle$, respectivement. De manière générale, la forme des interactions suit le schéma

$$\mathfrak{r} : \quad ((\mathtt{a}, i, \phi_1, \phi_2), (\mathtt{b}, j, \psi_1, \psi_2))$$

c'est-à-dire un nom de règle $\mathfrak{r}$ et deux quadruplets contenant un nom de protéine, un site et deux interfaces permutables. Une *application* générique de ce schéma peut se décrire par

$$\mathtt{a}\langle i + \phi_1 + \phi_2 + \phi'\rangle \ , \mathtt{b}\langle j + \psi_1 + \psi_2 + \psi''\rangle$$
$$\downarrow$$
$$\mathtt{a}\langle i^{\mathrm{x}} + \overline{\phi_1} + \phi_2 + \phi'\rangle \ , \mathtt{b}\langle j^{\mathrm{x}} + \overline{\psi_1} + \psi_2 + \psi'\rangle$$

où x est un nom frais et $\langle i + \phi_1 + \phi_2 + \phi'\rangle$ et $\langle j + \psi_1 + \psi_2 + \psi'\rangle$ sont des interfaces totales sur $\langle 1, \ldots, \mathfrak{n}(\mathtt{a})\rangle$ et $\langle 1, \ldots, \mathfrak{n}(\mathtt{b})\rangle$, respectivement. On peut remarquer que les interfaces ϕ' et ψ' ne sont pas mentionnées dans le schéma de la règle $\mathfrak{r}$. Ceci signifie que l'interaction peut avoir lieu *quel que soit* l'état des sites de ces interfaces.

Les décomplexations sont les interactions inverses des complexations. Par exemple, la décomplexation qui correspond à la version inverse de la complexation précédente est :

$$\mathtt{a}\langle 1^{\mathrm{x}} + 2^{\mathrm{y}} + 3 + 4\rangle \ , \mathtt{b}\langle 1^{\mathrm{y}} + 2 + \overline{3}\rangle \rightarrow \mathtt{a}\langle 1^{\mathrm{x}} + 2 + \overline{3} + 4\rangle \ , \mathtt{b}\langle 1 + \overline{2} + \overline{3}\rangle$$

qui retire le nom y de l'interface des protéines. Le schéma décrivant une décomplexation est similaire à celui décrivant une complexation :

$$\mathfrak{r}' : \quad ((\mathtt{a}, i, \phi_1, \phi_2), (\mathtt{b}, j, \psi_1, \psi_2))$$

L'application d'une règle de décomplexation est différente de celle d'une complexation : dans ce cas, les deux protéines qui interagissent doivent être

connectées par un arc entre le site i de $\mathtt{a}$ et le site j de $\mathtt{b}$. Donc une application générique de $\mathfrak{r}'$ consiste en

$$\mathtt{a}\langle i^{\mathrm{x}} + \phi_1 + \phi_2 + \phi' \rangle \ , \ \mathtt{b}\langle j^{\mathrm{x}} + \psi_1 + \psi_2 + \psi' \rangle$$
$$\downarrow$$
$$\mathtt{a}\big\langle i + \overline{\phi_1} + \phi_2 + \phi' \big\rangle \ , \mathtt{b}\big\langle j + \overline{\psi_1} + \psi_2 + \psi' \big\rangle$$

Afin de séparer les complexations des décomplexations, nous considérons deux fonctions, $\mathfrak{C}$ pour les complexations et $\mathfrak{D}$ pour les décomplexations, associant les couples de triplets $((\mathtt{a}, i, \phi_1, \phi_2), (\mathtt{b}, j, \psi_1, \psi_2))$ aux noms des règles $\mathfrak{r}$. Nous supposons que ces fonctions $\mathfrak{C}$ et $\mathfrak{D}$ ont des domaines disjoints. C'est pourquoi un nom de règle définira de manière unique s'il s'agit d'une complexation ou d'une décomplexation. Soit $\mathfrak{R}_{\mathfrak{s}}$ l'ensemble constitué de l'union de $\mathfrak{C}$ et $\mathfrak{D}$. Par abus de notation nous écrirons $(\mathtt{a}, i, \phi_1, \phi_2) \in \mathfrak{R}_{\mathfrak{s}}(\mathfrak{r})$ si $\mathfrak{R}_{\mathfrak{s}}(\mathfrak{r}) = ((\mathtt{a}, i, \phi_1, \phi_2), (\mathtt{b}, j, \psi_1, \psi_2))$ ou bien $\mathfrak{R}_{\mathfrak{s}}(\mathfrak{r}) = ((\mathtt{b}, j, \psi_1, \psi_2), (\mathtt{a}, i, \phi_1, \phi_2))$.

Nous pouvons désormais définir la sémantique opérationnelle du bioκ-calcul restreint à l'aide d'un système de transitions étiquetées. La forme des étiquettes est soit un triplet $(\mathtt{a}, \mathrm{x}, \mathfrak{r})$, encore noté $\mathtt{a}^x_{\mathfrak{r}}$, où $\mathtt{a}$ est un nom de protéine, x un nom de connexion et $\mathfrak{r}$ un nom de règle. Nous utiliserons μ pour dénoter à la fois $\mathtt{a}^x_{\mathfrak{r}}$ et τ et nous noterons $\mathtt{diff}(\mathsf{S}, \mathsf{S}')$ pour désigner l'ensemble $(\mathtt{nc}(\mathsf{S}')\backslash\mathtt{nc}(\mathsf{S})) \cup (\mathtt{nc}(\mathsf{S})\backslash\mathtt{nc}(\mathsf{S}'))$ contenant les noms de connexions apparaissant exclusivement dans l'une des deux solutions S ou S'.

Définition 35 (Système de transition du bioκ-calcul restreint) *La* relation de transition $\xrightarrow{\mu}$ *est la plus petite relation satisfaisant les réductions de la figure 6.1.*

Les règles (com) et (dec) définissent respectivement les complexations et les décomplexations que sont capables d'effectuer les protéines et, en même temps, le changement que ces interactions produisent sur les interfaces. Les règles (sol-g), (sol-d) et (mem) transposent ces transitions au cas des groupes et des membranes. Il est crucial ici que les noms créés ou effacés n'apparaissent nulle part ailleurs. La règle (cyto) transpose, quant à elle, les interactions *internes* à la cellule. Comme précédemment, les noms créés et supprimés ne doivent pas apparaître ailleurs. Notons que (cyto) exclut les complexations et les décomplexations entre le cytoplasme et la solution située à l'extérieur de la cellule. Les règles (react) et (ms-react) définissent les interactions (à la fois les complexations et les décomplexations) pour les groupes et les cellules. En particulier, (react) rend également compte des réactions ayant lieu entre des protéines situées dans les membranes de deux cellules différentes. Notons d'autre part que la forme de ces règles exclut de fait la possibilité que les deux points d'ancrage d'un nom fassent partie de l'interface d'une même protéine (*cf. l'auto-complexation* dans [DL04]). Il est tout à fait possible, comme nous

$$\text{(COM)}\quad \frac{(\mathtt{a}, i, \phi_1, \phi_2) \in \mathfrak{C}(\mathfrak{r}) \quad x \notin \mathtt{nc}(\sigma)}{\mathtt{a}\langle i + \phi_1 + \phi_2 + \sigma\rangle \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathtt{a}\langle i^x + \overline{\phi_1} + \phi_2 + \sigma\rangle}$$

$$\text{(DEC)}\quad \frac{(\mathtt{a}, i, \phi_1, \phi_2) \in \mathfrak{D}(\mathfrak{r})}{\mathtt{a}\langle i^x + \phi_1 + \phi_2 + \sigma\rangle \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathtt{a}\langle i + \overline{\phi_1} + \phi_2 + \sigma\rangle}$$

$$\text{(SOL-G)}\quad \frac{\mathsf{S} \xrightarrow{\mu} \mathsf{S}' \quad \mathtt{diff}(\mathsf{S}, \mathsf{S}') \cap \mathtt{nc}(\mathsf{T}) = \emptyset}{\mathsf{S}\,,\,\mathsf{T} \xrightarrow{\mu} \mathsf{S}'\,,\,\mathsf{T}} \qquad \text{(SOL-D)}\quad \frac{\mathsf{T} \xrightarrow{\mu} \mathsf{T}' \quad \mathtt{diff}(\mathsf{T}, \mathsf{T}') \cap \mathtt{nc}(\mathsf{S}) = \emptyset}{\mathsf{S}\,,\,\mathsf{T} \xrightarrow{\mu} \mathsf{S}\,,\,\mathsf{T}'}$$

$$\text{(MEM)}\quad \frac{\mathsf{M} \xrightarrow{\mu} \mathsf{M}' \quad \mathtt{diff}(\mathsf{M}, \mathsf{M}') \cap \mathtt{nc}(\mathsf{S}) = \emptyset}{\mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}] \xrightarrow{\mu} \mathrm{M}(\!|\,\mathsf{M}'\,|\!)[\mathsf{S}]} \qquad \text{(CYTO)}\quad \frac{\mathsf{S} \xrightarrow{\tau} \mathsf{S}' \quad \mathtt{diff}(\mathsf{S}, \mathsf{S}') \cap \mathtt{nc}(\mathsf{M}) = \emptyset}{\mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}] \xrightarrow{\tau} \mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}']}$$

$$\text{(REACT)}\quad \frac{\mathsf{S} \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathsf{S}' \quad \mathsf{T} \xrightarrow{\mathtt{b}^x_{\mathfrak{r}}} \mathsf{T}' \quad \mathtt{a} \neq \mathtt{b}}{\mathsf{S}\,,\,\mathsf{T} \xrightarrow{\tau} \mathsf{S}'\,,\,\mathsf{T}'} \qquad \text{(MS-REACT)}\quad \frac{\mathsf{M} \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathsf{M}' \quad \mathsf{S} \xrightarrow{\mathtt{b}^x_{\mathfrak{r}}} \mathsf{S}' \quad \mathtt{a} \neq \mathtt{b}}{\mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}] \xrightarrow{\tau} \mathrm{M}(\!|\,\mathsf{M}'\,|\!)[\mathsf{S}']}$$

FIG. 6.1 – Sémantique opérationnelle du $\mathtt{bio}\kappa$-calcul restreint

le verrons en conclusion, de rajouter des règles d'interaction modélisant un tel cas.

D'après la définition 35, la notation $\rightarrow$ que nous avons utilisée précédemment doit être lue comme $\xrightarrow{\tau}$. En réalité, dans [DL04], les transitions de $\mathtt{m}\kappa$-calcul sont définies à l'aide d'une relation de réduction simple, sans étiquette, qui correspond aux règles (com), (dec), (sol-g), (sol-d) et (react). Dans le cas présent, nous avons opté pour un système de transition étiqueté afin de réduire le nombre de règles qui aurait été plus grand que dans le cas de la définition 35 à cause de la présence des membranes.

Une première constation s'impose quant à ce système de transition : il s'avère que celui-ci préserve les conditions de bonne formation des solutions.

Proposition 7 *Si* S *est bien formée et si* $\mathsf{S} \xrightarrow{\mu} \mathsf{T}$ *alors* T *est aussi bien formée.*

Notons que pour l'instant le nom des membranes ne joue aucun rôle dans les réactions. Il ne deviendra utile que dans la description complète du système présentée dans la section 6.3.

6.2.2 Modélisation de la transduction du signal

La voie de transduction du signal RTK-MAPK a été intensément utilisée et étudiée dans beaucoup d'approches représentant et simulant ce système biologique [RSS01, DL04]. C'est pourquoi nous modélisons les premières étapes de cette voie de signalisation dans le bioκ-calcul, fournissant ainsi un référent pour comparer notre langage aux autres approches.

Les *récepteurs tyrosine kinase* (RTKs) sont des récepteurs situés à la surface membranaire qui, suite à la liaison d'un ligand (insuline, EGF, VEGF, etc...), sont activés et amènent une réponse cellulaire (croissance cellulaire, prolifération, etc...). Il existe en réalité plusieurs types de RTKs qui sont regroupés en différentes familles. Ils n'agissent pas de la même façon et la réponse cellulaire engendrée varie selon le type de RTK qui est activé. Suite à la liaison d'un ligand, les RTKs peuvent s'*autophosphoryler* (ajout d'un groupe phosphate sur un domaine de liaison). Cette modification est reconnue par certaines protéines qui vont se lier au RTK et recruter d'autres protéines qui seront activées ou inhibées et amèneront une réponse cellulaire. Les RTKs peuvent aussi médier leur action en phosphorylant d'autres protéines en plus de s'autophosphoryler. Ces protéines phosphorylées sont activées et engendrent elles aussi une cascade de signalisations. Ces cascades de signalisations mènent principalement à l'activation ou l'inhibition de facteurs de transcription qui modulent l'expression des gènes. Le schéma générique des cascades RTK-MAPK est donc le suivant : les RTKs, suite à la liaison d'un ligand (signal), vont amener des changements dans la cellule (ex. expression des gènes) pour médier un effet biologique (ex. entrée dans le cycle cellulaire).

L'exemple que nous prenons ici est celui de la réponse induite par le signal porté par une protéine de croisssance épidermique (`egf`). La forme dimérique (1) de `egf` peut se lier à son recepteur associé `egfr` (2), une protéine transmembranaire possédant un domaine de liaison extracellulaire situé sur la membrane plasmique de certaines cellules. Cette liaison active `egfr` en phosphorylisant le domaine intracellulaire de la protéine (3) et (4). Cette activation conduit à de multiples interactions avec des complexes protéiques situés dans le cytoplasme par le biais de liaisons/activations successives, en commençant avec la protéine adaptatrice `shc` (5). Cette cascade d'interactions prend fin avec l'activation de la kinase régulatrice du signal `erk`. Cette protéine, une fois phosphorylée, peut être transportée à l'intérieur du noyau par un mécanisme de translocation et va, par la suite, modifier l'expression d'un gêne ciblé, provocant l'entrée de la cellule en mitose. Ceci déclenche la division de la cellule et entraîne sa prolifération.

D'après la description biologique qui vient d'être faite, `egf`, `rtk`, and `shc` ont respectivement pour arité 3, 4, and 2. Nous donnons ici le rendu formel des cinq premières étapes de la description ci-dessus :

$$\begin{aligned}
\mathfrak{r}_1 &: ((\mathtt{egf}, 1, \overline{2}, \emptyset), (\mathtt{egf}, 1, \overline{2}, \emptyset)) \in \mathfrak{C} \\
\mathfrak{r}_2 &: ((\mathtt{egf}, 2, \emptyset, \emptyset), (\mathtt{egfr}, 1, \overline{4}, \emptyset)) \in \mathfrak{C} \\
\mathfrak{r}_3 &: ((\mathtt{egfr}, 2, \overline{3}+4, \emptyset), (\mathtt{egfr}, 2, \overline{3}+4, \emptyset)) \in \mathfrak{C} \\
\mathfrak{r}_4 &: ((\mathtt{egfr}, 2, \emptyset, \emptyset), (\mathtt{egfr}, 2, \emptyset, \emptyset)) \in \mathfrak{D} \\
\mathfrak{r}_5 &: ((\mathtt{egfr}, 3, \emptyset, \emptyset), (\mathtt{shc}, 1, \overline{2}, \emptyset)) \in \mathfrak{C}
\end{aligned}$$

FIG. 6.2 – Règles d'interaction modélisant la cascade RTK-MAPK

Une exécution possible de ces règles est présentée dans la figure 6.3, montrant que notre langage est suffisamment expressif pour témoigner de la causalité inhérente à la transduction du signal d'une manière à la fois précise et naturelle.

$$\begin{aligned}
&\mathtt{egf}\langle 1+\overline{2}\rangle \ , \mathtt{egf}\langle 1+\overline{2}\rangle \ , \\
&\qquad \mathrm{M}(\!| \ \mathtt{egfr}\langle 1+2^{+}\overline{3}+\overline{4}\rangle \ , \mathtt{egfr}\langle 1+2^{+}\overline{3}+\overline{4}\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}] \\
&\xrightarrow{\tau} \ \mathtt{egf}\langle 1^z+2\rangle \ , \mathtt{egf}\langle 1^z+2\rangle \ , \\
&\qquad \mathrm{M}(\!| \ \mathtt{egfr}\langle 1+2+\overline{3}+\overline{4}\rangle \ , \mathtt{egfr}\langle 1+2+\overline{3}+\overline{4}\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}] \qquad (\mathfrak{r}_1) \\
&\xrightarrow{\tau} \ \mathtt{egf}\langle 1^z+2^y\rangle \ , \mathtt{egf}\langle 1^z+2\rangle \ , \\
&\qquad \mathrm{M}(\!| \ \mathtt{egfr}\langle 1^y+2+\overline{3}+4\rangle \ , \mathtt{egfr}\langle 1+2+\overline{3}+\overline{4}\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}] \qquad (\mathfrak{r}_2) \\
&\xrightarrow{\tau} \ \mathtt{egf}\langle 1^z+2^y\rangle \ , \mathtt{egf}\langle 1^z+2^u\rangle \ , \\
&\qquad \mathrm{M}(\!| \ \mathtt{egfr}\langle 1^y+2+\overline{3}+4\rangle \ , \mathtt{egfr}\langle 1^u+2+\overline{3}+4\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}] \qquad (\mathfrak{r}_2) \\
&\xrightarrow{\tau} \ \mathtt{egf}\langle 1^z+2^y\rangle \ , \mathtt{egf}\langle 1^z+2^u\rangle \ , \\
&\qquad \mathrm{M}(\!| \ \mathtt{egfr}\langle 1^y+2^x+3+\overline{4}\rangle \ , \mathtt{egfr}\langle 1^u+2^x+3+\overline{4}\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}] \qquad (\mathfrak{r}_3) \\
&\xrightarrow{\tau} \ \mathtt{egf}\langle 1^z+2^y\rangle \ , \mathtt{egf}\langle 1^z+2^u\rangle \ , \\
&\qquad \mathrm{M}(\!| \ \mathtt{egfr}\langle 1^y+2+3+\overline{4}\rangle \ , \mathtt{egfr}\langle 1^u+2+3+\overline{4}\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}] \qquad (\mathfrak{r}_4) \\
&\xrightarrow{\tau} \ \mathtt{egf}\langle 1^z+2^y\rangle \ , \mathtt{egf}\langle 1^z+2^u\rangle \ , \\
&\qquad \mathrm{M}(\!| \ \mathtt{egfr}\langle 1^y+2+3^t+\overline{4}\rangle \ , \mathtt{egfr}\langle 1^u+2+3+\overline{4}\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1^t+2\rangle \ , \mathsf{S}] \qquad (\mathfrak{r}_5)
\end{aligned}$$

FIG. 6.3 – Déroulement des premières étapes de la cascade RTK-MAPK

Un certain nombre de problèmes dus à notre notation sont toutefois soulevés par cet exemple et méritent d'être discutés. Prenons cette fois-ci la solu-

tion initiale suivante :

$$\mathtt{egf}\langle 1+\overline{2}\rangle \ , \mathtt{egf}\langle 1+\overline{2}\rangle \ , \mathtt{egf}\langle 1+\overline{2}\rangle \ , \mathtt{egf}\langle 1+\overline{2}\rangle \ ,$$
$$\mathrm{M}(\!| \ \mathtt{egfr}\langle 1+2^x+\overline{3}+\overline{4}\rangle \ , \mathtt{egfr}\langle 1+2^x+\overline{3}+\overline{4}\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}]$$

Après deux applications de la règles $\mathfrak{r}_1$, nous obtenons la solution :

$$\mathtt{egf}\langle 1^x+2\rangle \ , \mathtt{egf}\langle 1^x+2\rangle \ , \mathtt{egf}\langle 1^y+2\rangle \ , \mathtt{egf}\langle 1^y+2\rangle \ ,$$
$$\mathrm{M}(\!| \ \mathtt{egfr}\langle 1+2^x+\overline{3}+\overline{4}\rangle \ , \mathtt{egfr}\langle 1+2^x+\overline{3}+\overline{4}\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}]$$

qui peut se réduire, après deux applications de la règle $\mathfrak{r}_2$ dans la « mauvaise » solution :

$$\mathtt{egf}\langle 1^x+2\rangle \ , \mathtt{egf}\langle 1^x+2^u\rangle \ , \mathtt{egf}\langle 1^y+2^v\rangle \ , \mathtt{egf}\langle 1^y+2\rangle \ ,$$
$$\mathrm{M}(\!| \ \mathtt{egfr}\langle 1^u+2^x+\overline{3}+4\rangle \ , \mathtt{egfr}\langle 1^v+2^x+\overline{3}+4\rangle \ , \mathsf{M} \ |\!)[\mathtt{shc}\langle 1+\overline{2}\rangle \ , \mathsf{S}]$$

dans laquelle deux formes dimériques différentes de `egf` se lient à une même paire de récepteurs `egfr`. Notre notation est trop simple pour exclure une telle configuration. Dans mκ-calcul, ce problème d'expressivité était résolu en utilisant des identifiants de réactions et l'utilisation de filtrage des règles en fonction de ces identifiants. Ce problème est par ailleurs directement lié à la question plus générale d'auto-assemblage que l'on a étudiée à part dans les sections précédentes.

Le second problème se manifeste à la fin de la cascade RTK-MAPK. Cette voie de signalisation termine par le transport d'une protéine particulière (`erk`) dans le noyau de la cellule. À cette étape de la description du langage, nous n'avons aucun mécanisme permettant de traduire cet événement. Un tel mécanisme sera discuté en détail dans la section 6.4.

6.2.3 Sémantique extensionnelle

Le relation de transition de la définition 35 fait correspondre à chaque solution un graphe où les nœuds réprésentent les solutions et les arcs sont les μ-étiquettes modélisant les transitions $\mathsf{S} \xrightarrow{\mu} \mathsf{S}'$. L'équivalence induite sur les solutions de bioκ-calcul correspond donc à l'isomorphisme de graphes : deux termes sont équivalents à condition que les graphes associés soient isomorphes. Cependant l'isomorphisme de graphe est trop fort du point de vue biologique puisqu'il distingue des solutions qui sont pourtant égales à des τ-transitions près. Supposons $\mathfrak{C}(\mathfrak{r}) = ((\mathtt{a}, 1, \emptyset), (\mathtt{b}, 1, \emptyset)) = \mathfrak{D}(\mathfrak{r}')$. Autrement dit, les règles $\mathfrak{r}$ et $\mathfrak{r}'$ sont réversibles. Alors les solutions $\mathtt{a}\langle 1^y+\sigma\rangle \ , \mathtt{b}\langle 1^y+\sigma'\rangle$ et $\mathtt{a}\langle 1+\sigma\rangle \ , \mathtt{b}\langle 1+\sigma'\rangle$ correspondent à des graphes qui ne sont pas isomorphes. Il n'y a pourtant, dans ce cas, aucune raison de séparer ces deux solutions : elles sont isomorphes modulo une τ-transition, ce qui est une réduction interne à la solution et ne devrait donc pas être observé.

L'équivalence ci-dessous est une adaptation à notre langage de la bisimulation faible des calculs de processus [Mil89b] qui corrige le problème que l'on vient de soulever. Posons $\mathsf{S} \stackrel{\tau}{\Longrightarrow} \mathsf{S}'$ si $\mathsf{S} \stackrel{\tau}{\rightarrow}^* \mathsf{S}'$ et $\mathsf{S} \stackrel{\mu}{\Longrightarrow} \mathsf{S}'$, avec $\mu \neq \tau$, si $\mathsf{S} \stackrel{\tau}{\rightarrow}^* \stackrel{\mu}{\rightarrow} \stackrel{\tau}{\rightarrow}^* \mathsf{S}'$.

Définition 36 *Une* bisimulation faible *est une relation binaire symétrique* $\mathfrak{R}$ *sur les solutions telles que* $\mathsf{S}\ \mathfrak{R}\ \mathsf{T}$ *implique :*

1. *si* $\mathsf{S} \stackrel{\tau}{\rightarrow} \mathsf{S}'$ *alors* $\mathsf{T} \stackrel{\tau}{\Longrightarrow} \mathsf{T}'$ *et* $\mathsf{S}'\ \mathfrak{R}\ \mathsf{T}'$
2. *si* $\mathsf{S} \stackrel{\mathtt{a}_\mathtt{r}^x}{\rightarrow} \mathsf{S}'$ *alors* $\mathsf{T} \stackrel{\mathtt{a}_\mathtt{r}^x}{\Longrightarrow} \mathsf{T}'$ *et* $\mathsf{S}'\ \mathfrak{R}\ \mathsf{T}'$

S *est bisimilaire à* T*, aussi noté* $\mathsf{S} \approx \mathsf{T}$*, si* $\mathsf{S}\ \mathfrak{R}\ \mathsf{T}$ *pour une relation de bisimulation* $\mathfrak{R}$*.*

À l'aide de cette notion d'équivalence, il est possible de montrer les propriétés suivantes du $\mathtt{bio}\kappa$-calcul restreint :

Proposition 8 *1.* $\mathsf{,}$ *est un opérateur modoïdal abélien ayant* $\emptyset$ *comme élément neutre. Autrement dit* $\mathsf{S}\mathsf{,}\mathsf{T} \approx \mathsf{T}\mathsf{,}\mathsf{S}$ *et* $(\mathsf{S}\mathsf{,}\mathsf{T})\mathsf{,}\mathsf{R} \approx \mathsf{S}\mathsf{,}(\mathsf{T}\mathsf{,}\mathsf{R})$ *et* $\mathsf{S}\mathsf{,}\emptyset \approx \mathsf{S}$*.*

2. $\approx$ *est préservé par tout renommage injectif identitaire sur les noms pendants. Énoncé plus formellement, soit* ι *un renommage injectif sur* $\mathtt{nc}(\mathsf{S})$ *tel que* ι *corresponde à l'identité sur* $\mathtt{np}(\mathsf{S})$*, alors* $\mathsf{S} \approx \iota(\mathsf{S})$*.*
3. $\approx$ *est préservé par les règles réversibles. Autrement dit, soit* $\mathfrak{C}(\mathfrak{r}) = ((\mathtt{a}, i, \psi), (\mathtt{b}, j, \phi))$ *et* $\mathfrak{D}(\mathfrak{r}') = ((\mathtt{a}, i, \overline{\psi}), (\mathtt{b}, j, \overline{\phi}))$ *alors*

$$\mathtt{a}\langle i + \psi + \sigma\rangle\ \mathsf{,}\ \mathtt{b}\big\langle j + \phi + \sigma'\big\rangle \approx \mathtt{a}\big\langle i^x + \overline{\psi} + \sigma\big\rangle\ \mathsf{,}\ \mathtt{b}\big\langle 1^x + \overline{\phi} + \sigma'\big\rangle$$

Une autre propriété plus intéressante de $\approx$ consiste en ce que deux systèmes bisimilaires se comportent de façon similaire s'ils sont plongés dans le même contexte.

Théoreme 5 $\approx$ *est une congruence.*

Démonstration : On doit prouver que, si $\mathsf{S} \approx \mathsf{T}$ alors

1. $\mathsf{S}\mathsf{,}\mathsf{R} \approx \mathsf{T}\mathsf{,}\mathsf{R}$
2. $\mathrm{M}(\!|\,\mathsf{S}\,|\!)[\mathsf{R}] \approx \mathrm{M}(\!|\,\mathsf{T}\,|\!)[\mathsf{R}]$
3. $\mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}] \approx \mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathsf{T}]$

si ces solutions respectent les conditions de bonne formation. Nous démontrons (1), les autres cas sont similaires. Soit $\mathfrak{R}$ une bisimulation contenant la paire (S, T) et soit $(\mathsf{S}'\mathsf{,}\mathsf{R})\ \mathfrak{R}'\ (\mathsf{T}'\mathsf{,}\mathsf{R})$ si

1. $\mathsf{S}'\,\mathfrak{R}\,\mathsf{T}'$,
2. $\mathsf{S}'\mathsf{,}\mathsf{R}$ et $\mathsf{T}'\mathsf{,}\mathsf{R}$ sont bien formées.

Pour démontrer que $\mathfrak{R}'$ est une bisimulation, nous devons prouver que si $\mathsf{S}', \mathsf{R} \xrightarrow{\mu} \mathsf{U}$ alors il existe U' tel que $\mathsf{T}', \mathsf{R} \stackrel{\mu}{\Longrightarrow} \mathsf{U}'$ et $\mathsf{U}\,\mathfrak{R}'\,\mathsf{U}'$. Deux cas sont possibles :

$\mu = \mathtt{a}^x_\mathtt{r}$. Alors soit $\mathsf{U} = \mathsf{S}', \mathsf{R}'$ avec $\mathsf{R} \xrightarrow{\mathtt{a}^x_\mathtt{r}} \mathsf{R}'$ ou bien $\mathsf{U} = \mathsf{S}'', \mathsf{R}$ avec $\mathsf{S}' \xrightarrow{\mathtt{a}^x_\mathtt{r}} \mathsf{S}''$. Le premier cas est immédiat. Dans le second cas, $\mathsf{T}' \stackrel{\mathtt{a}^x_\mathtt{r}}{\Longrightarrow} \mathsf{T}''$ et $\mathsf{S}''\,\mathfrak{R}\,\mathsf{T}''$. Par définition : $(\mathsf{S}'', \mathsf{R})\,\mathfrak{R}'\,(\mathsf{T}'', \mathsf{R})$.

$\mu = \tau$. Le cas intéressant intervient quand S' et R interagissent par la règle (react) (les autres cas étant traités comme pour le cas ci-dessus). Supposons $\mathsf{U} = \mathsf{S}'', \mathsf{R}'$ avec $\mathsf{S}' \xrightarrow{\mathtt{a}^x_\mathtt{r}} \mathsf{S}''$ et $\mathsf{R} \xrightarrow{\mathtt{b}^x_\mathtt{r}} \mathsf{R}'$. Puisque $\mathsf{S}'\,\mathfrak{R}\,\mathsf{T}'$, il existe T'' tel que $\mathsf{T}' \stackrel{\mathtt{a}^x_\mathtt{r}}{\Longrightarrow} \mathsf{T}''$ et $\mathsf{S}''\,\mathfrak{R}\,\mathsf{T}''$. Alors, par (react), $\mathsf{T}', \mathsf{R} \stackrel{\tau}{\Longrightarrow} \mathsf{T}'', \mathsf{R}'$ et $(\mathsf{S}'', \mathsf{R}')\,\mathfrak{R}'\,(\mathsf{T}'', \mathsf{R}')$ par définition de $\mathfrak{R}'$.

Du point de vue biologique, la propriété de substitution de la proposition 8 peut se révéler trop forte et par conséquent rendre la sémantique « insensible ». Dans ce contexte, on cherche en général à prouver que les deux systèmes qui sont comparés se comportent de façon équivalente lorsqu'ils sont plongés *dans un certain nombre de contextes*, et non pas dans tous les contextes possibles. C'est pourquoi une sémantique paramétrée par une propriété de congruence pourrait se révéler plus adaptée au contexte biologique. Cependant, la définition de ces paramètres et les propriétés relatives à une « bonne » sémantique étant pour l'instant discutables, nous laissons cette question ouverte dans le présent travail.

Nous terminons mantenant cette section par une série de remarques concernant $\approx$. Il est interessant de noter par exemple que $\mathsf{S} \approx \mathsf{T}$ n'implique pas $\mathtt{np}(\mathsf{S}) = \mathtt{np}(\mathsf{T})$. Cela pour deux raisons : tout d'abord, par bisimulation, $\mathsf{S} \xrightarrow{\mathtt{a}^x_\mathtt{r}} \mathsf{S}'$ peut être simulé par $\mathsf{T} \xrightarrow{\mathtt{a}^y_\mathtt{r}} \mathsf{T}'$ avec $x \neq y$. D'autre part, si nous prenons les ensembles de réactions vides – *i.e.* $\mathfrak{C} = \emptyset$ et $\mathfrak{D} = \emptyset$ –, alors $\mathtt{a}\langle 1^x\rangle \approx \emptyset$ mais leurs connexions pendantes sont différentes. Néanmoins, une relation peut être établie sur un sous-ensemble des noms pendants de deux solutions bisimilaires. Soit $\mathsf{oe}(\mathsf{S})$, que nous appellerons l'ensemble des *noms observables*, défini comme étant l'ensemble

$$\mathsf{oe} = \left\{ x \mid \mathsf{S} \xrightarrow{\mathtt{a}^x_\mathtt{r}} \mathsf{S}' \text{ et } \mathtt{r} \text{ est dans le domaine de } \mathfrak{D} \right\}$$

Nous pouvons facilement établir que si $\mathsf{S} \approx \mathsf{T}$ alors il existe un renommage injectif ι tel que $\mathsf{oe}(\mathsf{S}) = \iota(\mathsf{oe}(\mathsf{T}))$.

À titre d'illustration de la définition 36, on peut noter qu'une cellule dont la membrane ne peut pas interagir avec les éléments à l'extérieur est équivalente à la solution vide. Soit M un groupe de protéines. M est dit *inerte* si $\mathsf{M} \stackrel{\mathtt{a}^x_\mathtt{r}}{\not\Rightarrow} \mathsf{M}'$ pour tout $\mathtt{a}^x_\mathtt{r}$ et M'. Il est possible de montrer que, si M est inerte, alors $\mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}] \approx \emptyset$.

6.3 Interactions entre membranes

Le langage présenté en section 6.2 ne diffère pas vraiment de $\mathtt{m}\kappa$-calcul. Les cellules, en particulier, ne joue aucun rôle notable puisque leur structure est préservée par les transitions. Dans cette section, on explore une extension basée sur les primitives du Brane calcul, rendant possible la modélisation des fusions comme la *fusion d'endosomes* suivante :

$$\textsc{esm}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}]\ ,\ \textsc{esm}(\!|\,\mathsf{N}\,|\!)[\mathsf{T}] \xrightarrow{\tau} \textsc{esm}(\!|\,\mathsf{M}\ ,\ \mathsf{N}\,|\!)[\mathsf{S}\ ,\ \mathsf{T}]$$

Cette extension utilise des mécanismes d'ordre supérieur, proches du *π-calcul d'ordre supérieur*, une extension similaire déjà utilisée pour l'étude du π-calcul [San93].

6.3.1 Les mréagents

Nous commençons par étendre la syntaxe avec les *réagents membranaires*, appelés par la suite mréagents.

Définition 37 (La syntaxe du $\mathtt{bio}\kappa$-calcul) *La syntaxe du $\mathtt{bio}\kappa$-calcul définit les solutions S par :*

$$\begin{array}{lccccccc} \mathsf{S} ::= & \emptyset & | & \mathtt{a}\langle\sigma\rangle & | & \textsc{m}(\!|\,\mathsf{S}\,|\!)[\mathsf{S}] & | & \mathsf{S}\,,\,\mathsf{S} \quad | \quad \wr\mathsf{S}\,;\,\mathsf{S}\int\cdot\mathsf{S} \\ & \textit{(vide)} & & \textit{(protéine)} & & \textit{(cellule)} & & \textit{(groupe)} \quad \textit{(mréagent)} \end{array}$$

Un mréagent $\wr\mathsf{M}\,;\,\mathsf{S}\int\cdot\mathsf{T}$ est une solution intermédiaire et instable utilisée pour manifester la capacité d'une membrane M, isolant une solution S de l'environnement extérieur T, à procéder à une fusion avec une autre membrane. Cette définition appelle à considérer les éléments M et S comme les constituants d'une cellule $\textsc{m}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}]$ et, en tant que telle, suppose un certain nombre de contraintes sur leur structure, comme nous l'avons fait pour la propriété 3. En particulier, M doit être un multi-ensemble de protéines et les connexions pendantes de S doivent être connectées à des protéines situées dans la membrane M. D'autre part, nous contraignons les mréagents à ne pas contenir d'autres mréagents.

Propriété 4 (Bonne formation des mréagents) *Un mréagent $\wr\mathsf{M}\,;\,\mathsf{S}\int\cdot\mathsf{T}$ est dit* bien formé *s'il vérifie les conditions suivantes :*

1. $\mathsf{S} \equiv \prod_{i\in 1\ldots n} \mathtt{a_i}\langle\sigma_i\rangle$*, où les* $\mathtt{a_i}\langle\sigma_i\rangle$ *désignent des protéines*
2. $\mathtt{np}(\mathsf{S}) \subseteq \mathtt{np}(\mathsf{M})$.
3. S *et* T *sont des solutions restreintes.*[4]

[4] Au sens de la définition 34.

On définit ensuite deux opérations relatives aux mréagents : l'opération de *fusion* qui réalise la mise en commun, dans une même cellule, de deux cytoplasmes auparavant séparés par deux membranes ; l'opération d'*activation* qui se présente comme un effet successif à un événement élémentaire de type complexation.

Plus précisément, les fusions sont formalisées par une fonction $\mathfrak{F}$ associant à des noms de règles des triplets de la forme $((\mathrm{M}, \mathrm{M}'), \mathrm{N})$. Nous écrirons $(\mathrm{M} \otimes \mathrm{M}', \mathrm{N}) = \mathfrak{F}(\mathfrak{r})$ si ou bien $\mathfrak{F}(\mathfrak{r}) = ((\mathrm{M}, \mathrm{M}'), \mathrm{N})$ ou bien $\mathfrak{F}(\mathfrak{r}) = ((\mathrm{M}', \mathrm{M}), \mathrm{N})$. Nous écrirons également $\mathrm{M} \in \mathfrak{F}(\mathfrak{r})$ si $(\mathrm{M} \otimes \mathrm{M}', \mathrm{N}) = \mathfrak{F}(\mathfrak{r})$, pour un M' et un N quelconques. Nous supposons à nouveau que les domaines de $\mathfrak{F}$, $\mathfrak{C}$ et $\mathfrak{D}$ sont disjoints.

Afin de pouvoir contrôler les mécanismes de fusions, il nous faut décrire dans quelles conditions une membrane peut changer de type. Comme expliqué en introduction, nous relions cet événement à l'interaction entre deux protéines. Plus exactement, une complexation impliquant une protéine transmembranaire peut désormais activer la membrane et la rendre apte à effectuer une fusion. L'activation liée à une complexation est définie par une fonction d'activation $\mathfrak{A}$ qui associe aux paires $(\mathsf{a}_\mathfrak{r}, \mathrm{M})$ le nom de la membrane activée.

Par abus de notation, nous utiliserons μ pour dénoter aussi les étiquettes de la forme $\mathtt{m}_\mathfrak{r}$.

Définition 38 (Système de transition du bioκ-calcul restreint) *La* relation de transition $\xrightarrow{\mu}$ *est la plus petite relation incluant les règles de la définition 35 dans lesquelles (mem) et (ms-react) ont pour prémisse « $(\mathsf{a}_\mathfrak{r}, \mathrm{M})$ n'est pas dans le domaine de $\mathfrak{A}$ » et les réductions de la figure 6.4.*

La règle (mrea) prépare la membrane d'une cellule à être fusionnée avec une cellule qui est entourée par cette membrane (*cellule fille*) ou bien avec une cellule située au même niveau hiérarchique (*cellule sœur*). La précondition garantit que les cellules peuvent participer à une fusion. La règle (ctx) transpose la capacité de fusionner au groupe de solutions en figeant ce groupe dans le mréagent. Les règles (fuse-h) et (fuse-v) définissent la fusion proprement dite entre cellules enveloppées (mère/fille) et cellules situées au même niveau (cellules sœurs). La nouvelle cellule est créée avec le type de membrane renvoyé par la fonction $\mathfrak{F}$. La règle (mem-a) est un simple raffinement de (mem). Elle modélise la possibilité d'un effet de bord sur le nom de la membrane sur laquelle a lieu la complexation. De telles interactions peuvent activer les membranes en changeant leur capacité de fusionner. D'une manière similaire, la règle (ms-react) est un raffinement de (ms-react).

6.3.2 Infection virale

Un virus est un parasite intracellulaire qui utilise la machinerie de réplication transcriptionnelle de la cellule infectée pour dupliquer son propre matériel

(MREA)
$$\frac{\mathrm{M} \in \mathfrak{F}(\mathfrak{r})}{\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}] \xrightarrow{\mathtt{m}_{\mathfrak{r}}} (\!|\, \mathsf{M} \,;\, \mathsf{S} \,|\!) \cdot \emptyset}$$

(CTX)
$$\frac{\mathsf{S} \xrightarrow{\mu} (\!|\, \mathsf{M} \,;\, \mathsf{S}'' \,|\!) \cdot \mathsf{S}'}{\mathsf{S} \,,\, \mathsf{T} \xrightarrow{\mu} (\!|\, \mathsf{M} \,;\, \mathsf{S}'' \,|\!) \cdot (\mathsf{S}' \,,\, \mathsf{T})}$$

(FUSE-H)
$$\frac{\mathsf{S} \xrightarrow{\mathtt{m}_{\mathfrak{r}}} (\!|\, \mathsf{M} \,;\, \mathsf{S}'' \,|\!) \cdot \mathsf{S}' \quad \mathsf{T} \xrightarrow{\mathtt{n}_{\mathfrak{r}}} (\!|\, \mathsf{N} \,;\, \mathsf{T}'' \,|\!) \cdot \mathsf{T}' \quad \mathfrak{F}(\mathfrak{r}) = (\mathrm{M} \otimes \mathrm{N}, \mathrm{M}')}{\mathsf{S} \,,\, \mathsf{T} \xrightarrow{\tau} \mathsf{S}' \,,\, \mathsf{T}' \,,\, \mathrm{M}'(\!|\, \mathsf{M} \,,\, \mathsf{N} \,|\!)[\mathsf{S}'' \,,\, \mathsf{T}'']}$$

(FUSE-V)
$$\frac{\mathsf{S} \xrightarrow{\mathtt{n}_{\mathfrak{r}}} (\!|\, \mathsf{N} \,;\, \mathsf{T} \,|\!) \cdot \mathsf{S}' \quad \mathfrak{F}(\mathfrak{r}) = (\mathrm{M} \otimes \mathrm{N}, \mathrm{M}')}{\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}] \xrightarrow{\tau} \mathrm{M}'(\!|\, \mathsf{M} \,,\, \mathsf{N} \,|\!)[\mathsf{S}'] \,,\, \mathsf{T}}$$

(MEM-A)
$$\frac{\mathsf{M} \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathsf{M}' \quad \mathfrak{A}(\mathtt{a}_{\mathfrak{r}}, \mathrm{M}) = \mathrm{N} \quad \mathtt{diff}(\mathsf{M}, \mathsf{M}') \cap \mathtt{nc}(\mathsf{S}) = \emptyset}{\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}] \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathrm{N}(\!|\, \mathsf{M}' \,|\!)[\mathsf{S}]}$$

(MS-AREACT)
$$\frac{\mathsf{M} \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathsf{M}' \quad \mathsf{S} \xrightarrow{\mathtt{b}^x_{\mathfrak{r}}} \mathsf{S}' \quad \mathtt{a} \neq \mathtt{b} \quad \mathfrak{A}(\mathtt{a}_{\mathfrak{r}}, \mathrm{M}) = \mathrm{N}}{\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}] \xrightarrow{\tau} \mathrm{N}(\!|\, \mathsf{M}' \,|\!)[\mathsf{S}']}$$

FIG. 6.4 – Sémantique opérationnelle du bioκ-calcul

génétique. Habituellement, un virus consiste en un matériel génétique (ADN ou ARN), une structure protéique appelée *capside* protégeant ce matériel génétique – on utilise le terme de *nucléocapside* pour dénoter à la fois la capside et le matériel génétique – et une possible enveloppe (généralement extraite d'une cellule précédemment infectée et utilisée plus tard pour infecter d'autres cellules).

Ci-dessous, nous proposons une modélisation, dans le bioκ-calcul, d'un virus de type influenza, reposant sur des descriptions similaires faites dans [Car05, DP04]. On s'intéresse plus particulièrement à la partie infection puisque nous ne pouvons pas encore exprimer l'ajout de matériel nouveau dans le langage. Cette partie consiste en les étapes suivantes :

(1) Une interaction de type protéine-protéine entre une protéine faisant partie de l'enveloppe du virus – l'hemagglutinine `ha` – et un récepteur transmembranaire de la cellule – une glycoprotéine `gly`. Cette interaction active `gly` et prépare la membrane à laisser entrer le virus dans la cellule.

(2) Le virus pénètre dans la cellule et se retrouve enveloppé par une vésicule VES.

(3) Une fusion a lieu entre la nouvelle vésicule et la membrane d'un endosome (EDSM) dans le cytoplasme de la cellule. Ceci permet au virus d'entrer dans l'endosome.

(4) une seconde fusion a lieu entre l'endosome et le virus qui fait alors partie du cytoplsame de la cellule infectée. Ceci conduit à une exocytose qui relâche la nucleocapside du virus dans le cytoplasme.

Considérons les règles suivantes :

$$\begin{aligned} \mathfrak{r}_3 &: \quad ((\text{EDSM}, \text{VES}), \text{EDSM}) \quad \in \mathfrak{F} \\ \mathfrak{r}_4 &: \quad ((\text{EDSM}, \text{VS}), \text{EDSM}) \quad \in \mathfrak{F} \end{aligned}$$

Considérons également la solution initiale Virus , Cell où les différents composants sont décrits par :

$$\begin{aligned} \mathsf{Virus} &:= \text{VS}(\!| \, \mathtt{ha}\langle 1\rangle \, |\!)[\mathsf{Nucaps}] \\ \mathsf{Cell} &:= \text{CLL}(\!| \, \mathtt{gly}\langle 1\rangle \ , \mathsf{M}_c \, |\!)[\mathsf{Endosome} \ , \mathsf{Cytosol}] \\ \mathsf{Endosome} &:= \text{EDSM}(\!| \, \mathsf{M}_e \, |\!)[\mathsf{E}_s] \end{aligned}$$

Nous décrivons dans un premier temps la dernière partie de l'infection, en supposant que le virus est déjà entré dans la cellule hôte. Nous omettons donc les premières étapes car nous ne pouvons pas pour l'instant exprimer le mécanisme permettant au virus de pénétrer dans la cellule. Dans la section 6.4, nous analyserons cette partie. Supposons donc que

$$\text{CLL}(\!| \, \mathsf{M}_c \, |\!)[\mathsf{Endosome} \ , \text{VES}(\!| \, \mathtt{gly}\langle 1\rangle \, |\!)[\mathsf{Virus}] \ , \mathsf{Cytosol}]$$

soit la solution initiale. Nous présentons maintenant une évolution possible de ce système :

$$\begin{aligned} &\text{CLL}(\!| \, \mathsf{M}_c \, |\!)[\text{EDSM}(\!| \, \mathsf{M}_e \, |\!)[\mathsf{E}_s] \ , \text{VES}(\!| \, \mathtt{gly}\langle 1\rangle \, |\!)[\mathsf{Virus}] \ , \mathsf{Cytosol}] \\ &\quad \xrightarrow{\tau} \quad \text{CLL}(\!| \, \mathsf{M}_c \, |\!)[\text{EDSM}(\!| \, \mathsf{M}_e \ , \mathtt{gly}\langle 1\rangle \, |\!)[\mathsf{Virus} \ , \mathsf{E_s}] \ , \mathsf{Cytosol}] && (\mathfrak{r}_3) \\ &\quad \equiv \quad \text{CLL}(\!| \, \mathsf{M}_c \, |\!)[\text{EDSM}(\!| \, \mathsf{M}_e \ , \mathtt{gly}\langle 1\rangle \, |\!)[\text{VS}(\!| \, \mathtt{ha}\langle 1\rangle \, |\!)[\mathsf{Nucaps}] \ , \mathsf{E}_s] \ , \mathsf{Cytosol}] \\ &\quad \xrightarrow{\tau} \quad \text{CLL}(\!| \, \mathsf{M}_c \, |\!)[\text{EDSM}(\!| \, \mathsf{M}_e \ , \mathtt{gly}\langle 1\rangle \ , \mathtt{ha}\langle 1\rangle \, |\!)[\mathsf{E}_s] \ , \mathsf{Nucaps} \ , \mathsf{Cytosol}] && (\mathfrak{r}_4) \end{aligned}$$

6.3.3 Une sémantique extensionnelle pour les cellules

La sémantique extensionnelle de la définition 36 doit être raffinée afin de tenir compte des mréagents et de ces nouvelles transitions. Ce raffinement devrait en particulier égaliser les solutions telles que $\mathtt{a}\langle 1^x\rangle$, $\text{M}(\!| \, \mathtt{b}\langle 1^x\rangle \, |\!)[\mathsf{S}]$ et $\mathtt{a}\langle 1\rangle$, $\text{N}(\!| \, \mathtt{b}\langle 1\rangle \, |\!)[\mathsf{S}]$ quand $\mathfrak{C}(\mathfrak{r}) = ((\mathtt{a}, 1, \emptyset), (\mathtt{b}, 1, \emptyset))$ $\mathfrak{D}(\mathfrak{r}') = ((\mathtt{a}, 1, \emptyset), (\mathtt{b}, 1, \emptyset))$ et $\mathfrak{A}(\mathtt{b}_{\mathfrak{r}}, \text{N}) = \text{M}'$ et $\mathfrak{A}(\mathtt{b}_{\mathfrak{r}'}, \text{M}') = \text{N}$. Ce cas est très similaire au cas des règles réversibles que nous évoquions dans la section 6.2.

Les notations $\mathsf{S} \stackrel{\tau}{\Longrightarrow} \mathsf{T}$ et $\mathsf{S} \stackrel{\mu}{\Longrightarrow} \mathsf{T}$, $\mu \neq \tau$, sont définies comme dans la définition 35.

Définition 39 *Une* bisimulation contextuelle *est une relation binaire symétrique* $\mathfrak{R}$ *entre les solutions qui est une bisimulation et telle que si* $\mathsf{S}\,\mathfrak{R}\,\mathsf{T}$ *et si* $\mathsf{S} \xrightarrow{\mathtt{m}_\mathfrak{r}} \wr \mathsf{M} \,;\, \mathsf{S}'' \wr \cdot \mathsf{S}'$ *alors* $\mathsf{T} \xRightarrow{\mathtt{m}_\mathfrak{r}} \wr \mathsf{M}' \,;\, \mathsf{T}'' \wr \cdot \mathsf{T}'$ *et, pour tout* N, R, *et* N *tels que* $\mathfrak{F}(\mathfrak{r}) = (\mathrm{M} \otimes \mathrm{N}, \mathrm{N}')$, *on a :*

1. $\Big(\mathsf{S}'' \,,\, \mathrm{N}'(\!|\, \mathsf{M} \,,\, \mathsf{N} \,|\!)[\mathsf{S}']\Big)\ \mathfrak{R}\ \Big(\mathsf{T}'' \,,\, \mathrm{N}'(\!|\, \mathsf{M}' \,,\, \mathsf{N} \,|\!)[\mathsf{T}'']\Big)$
2. $\Big(\mathsf{S}' \,,\, \mathrm{N}'(\!|\, \mathsf{M} \,,\, \mathsf{N} \,|\!)[\mathsf{S}'' \,,\, \mathsf{R}]\Big)\ \mathfrak{R}\ \Big(\mathsf{T}' \,,\, \mathrm{N}'(\!|\, \mathsf{M}' \,,\, \mathsf{N} \,|\!)[\mathsf{T}'' \,,\, \mathsf{R}]\Big)$

S *est contextuellement bisimilaire à* T, *ce que l'on note également* $\mathsf{S} \approx_c \mathsf{T}$, *si* $\mathsf{S}\,\mathfrak{R}\,\mathsf{T}$ *pour une bisimulation contextuelle* $\mathfrak{R}$.

Les propriétés énoncées dans la proposition 8 sont aussi valables pour $\approx$. D'autre part, on retrouve la propriété suivante :

Théoreme 6 $\approx_c$ *est une congruence.*

Cette preuve est similaire à celle du théorème 5. Nous ne discutons par conséquent que le cas des contextes $[\cdot]\,,\,\mathsf{R}$. Soit $\mathfrak{R}$ une bisimulation contextuelle telle que $\mathsf{S}\,\mathfrak{R}\,\mathsf{T}$ et soit $(\mathsf{S}'\,,\,\mathsf{R})\ \mathfrak{R}'\ (\mathsf{T}'\,,\,\mathsf{R})$ si

1. $\mathsf{S}'\,\mathfrak{R}\,\mathsf{T}'$,
2. $\mathsf{S}'\,,\,\mathsf{R}$ et $\mathsf{T}'\,,\,\mathsf{R}$ vérifient les conditions de bonne formation.

Pour démontrer que $\mathfrak{R}'$ est une bisimulation, il nous faut prouver que si $\mathsf{S}'\,,\,\mathsf{R} \xrightarrow{\mu} \mathsf{U}$ alors $\mathsf{T}'\,,\,\mathsf{R} \xRightarrow{\mu} \mathsf{U}'$ et $\mathsf{U}\,\mathfrak{R}'\,\mathsf{U}'$. Parmi tous les cas possibles, nous ne présentons que le cas où $\mathsf{S}'\,,\,\mathsf{R} \xrightarrow{\tau} \mathsf{U}$ est dû à l'application d'une règle (fuse-h) entre un mréagent de S' et un autre de R.

Soit $\mathsf{S}' \xrightarrow{\mathtt{m}_\mathfrak{r}} \wr \mathsf{M} \,;\, \mathsf{S}''' \wr \cdot \mathsf{S}''$, $\mathsf{R} \xrightarrow{\mathtt{n}_\mathfrak{r}} \wr \mathsf{N} \,;\, \mathsf{R}'' \wr \cdot \mathsf{R}'$, et $\mathfrak{F}(\mathfrak{r}) = (\mathrm{M} \otimes \mathrm{N}, \mathrm{M}')$. Alors $\mathsf{U} = \mathsf{S}''\,,\,\mathrm{M}'(\!|\, \mathsf{M} \,,\, \mathsf{N} \,|\!)[\mathsf{S}'''\,,\,\mathsf{R}'']\,,\,\mathsf{R}'$. Puisque $\mathsf{S}'\,\mathfrak{R}\,\mathsf{T}'$ alors $\mathsf{T}' \xRightarrow{\mathtt{m}_\mathfrak{r}} \wr \mathsf{M}' \,;\, \mathsf{T}''' \wr \cdot \mathsf{T}''$ et $\mathsf{U}' = \mathsf{T}''\,,\,\mathrm{M}'(\!|\, \mathsf{M}' \,,\, \mathsf{N} \,|\!)[\mathsf{T}'''\,,\,\mathsf{R}'']\,,\,\mathsf{R}'$. Par définition de $\mathfrak{R}$, il vient que

$$(\mathsf{S}''\,,\,\mathrm{M}'(\!|\, \mathsf{M} \,,\, \mathsf{N} \,|\!)[\mathsf{S}'''\,,\,\mathsf{R}'']\,)\ \mathfrak{R}\ (\mathsf{T}''\,,\,\mathrm{M}'(\!|\, \mathsf{M}' \,,\, \mathsf{N} \,|\!)[\mathsf{T}'''\,,\,\mathsf{R}'']\,)$$

Nous pouvons en conclure que $\mathsf{U}\,\mathfrak{R}'\,\mathsf{U}'$ à l'aide du contexte $[\cdot]\,,\,\mathsf{R}'$.

Le problème des bisimulations contextuelles est qu'elles utilisent des quantifications universelles qui sont extrêmement difficiles à vérifier dans la pratique. On peut par conséquent se demander s'il ne serait pas possible de simplifier cette notion. Par exemple, au lieu de quantifier sur les cellules, on pourait demander simplement la bisimulation entre les mréagents, c'est-à-dire $\mathsf{M} \approx_c \mathsf{M}'$, $\mathsf{S}'' \approx_c \mathsf{T}''$, et $\mathsf{S}' \approx_c \mathsf{T}'$. Il est aisé de démontrer que l'équivalence qui est alors induite, et que nous noterons $\approx_c^+$, est aussi une congruence et que $\approx_c^+ \subseteq \approx_c$. Au moment de la rédaction de ce manuscrit, il n'est pas établi si cette inclusion est stricte ou non. Cette question requiert un travail plus approfondi.

6.4 Autres types d'interactions entre membranes

Le bioκ-calcul présenté en sections 6.2 et 6.3 est limité en terme d'expressivité. Des mécanismes tels que la translocation – où une molécule simple peut traverser la membrane d'une cellule – ou l'endocytose – qui permet à une cellule d'ingérer des éléments extérieurs à celle-ci en les enveloppant avec une partie de sa membrane – ne peuvent pas être représentés. C'est pour cette raison que nous n'avons pas modélisé les premières étapes d'infection du virus dans l'exemple de la section 6.3.2. De tels mécanismes ne sont pas élémentaires et requièrent par conséquent des choix en terme de représentation. Nous discutons dans cette section plusieurs approches possibles.

6.4.1 Translocations.

La translocation est un mécanisme permettant le transport de protéines à travers les membranes. Ce mécanisme est très ciblé et contrôlé par des protéines transmembranaires particulières, différentes pour chaque type de membranes.

La translocation ne permet pas de transporter la protéine en une seule étape, les protéines étant généralement trop grosses pour cela. Ce problème est résolu de deux façons. Soit l'ARN messager, comportant le code génétique de la protéine, interagit avec un ribosome – gros complexe protéique en charge de la traduction de ce code ; cette interaction traduit le code de l'ARN messager et, *en même temps*, crée la protéine de l'autre coté de la membrane. Soit, de manière alternative, sont utilisées des protéines spécifiques, appelées *chaperonnes*, qui déplient la structure de la protéine pendant le processus (c'est d'ailleurs le cas dans l'exemple de la cascade RTK-MAPK décrit en section 6.2.2).

Nous fournissons ici une abstraction de cette description de bas niveau du mécanisme et supposons que les protéines peuvent traverser les membranes. Une première approximation de la définition d'une translocation pourrait être le simple fait de vérifier que la protéine entrante est libre de connexion et possède une interface adéquate :

$$\mathtt{a}\langle\phi+\psi\rangle \ \mathtt{,}\ \mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathtt{S}] \xrightarrow{\tau} \mathrm{M}(\!|\,\mathsf{M}\,|\!)[\mathtt{a}\langle\overline{\phi}+\psi\rangle \ \mathtt{,}\ \mathsf{S}']$$

Comme le suggère la notation, les interfaces partielles ϕ et ψ sont des fonctions permutables. Il s'ensuit que $\mathtt{nc}(\phi+\psi)=\emptyset$. Cette description n'est pourtant pas satisfaisante car elle fait jouer un rôle passif à la membrane alors que ce processus est, au contraire, hautement spécifique. Comme tel, il est impossible de représenter les effets des protéines chaperonnes.

Une meilleure manière de procéder consiste à représenter le mécanisme de translocation comme le résultat d'une décomplexation entre une protéine appartenant à la membrane d'une cellule et une protéine extérieure à la cellule et liée uniquement à celle-ci. Pour éviter les confusions entre les deux phénomènes

de décomplexation – simple décomplexation et décomplexation avec translocation – nous utilisons une nouvelle fonction $\mathfrak{T}$ associant à chaque nom de règle le n-uplet $((\mathsf{a}, i, \phi, \phi'), (\mathsf{b}, j, \psi, \psi'), \mathrm{M}, \mathrm{N})$. Comme auparavant, nous présupposons que les domaines entre la fonction de translocation $\mathfrak{T}$ et les autres fonctions précédemment introduites sont disjoints. Deux règles régissent le cas des translocations :

$$\frac{\mathfrak{T}(\mathfrak{r}) = ((\mathsf{a}, i, \phi, \phi'), (\mathsf{b}, j, \psi, \psi'), \mathrm{M}, \mathrm{N})}{\mathsf{a}\langle i^x + \phi + \phi' \rangle \xrightarrow{\mathsf{a}^x_{\mathfrak{r}}} \emptyset} \text{(TRS-P)}$$

$$\frac{\mathsf{M} \xrightarrow{\mathsf{b}^x_{\mathfrak{r}}} \mathsf{M}' \qquad \mathfrak{T}(\mathfrak{r}) = ((\mathsf{a}, i, \phi, \phi'), (\mathsf{b}, j, \psi, \psi'), \mathrm{M}, \mathrm{N})}{\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}] \xrightarrow{\mathsf{b}^x_{\mathfrak{r}}} \mathrm{N}(\!|\, \mathsf{M}' \,|\!)[\mathsf{a}\langle i + \overline{\phi} + \phi' \rangle \ , \mathsf{S}']} \text{(TRS-M)}$$

FIG. 6.5 – Réductions modélisant la translocation

Dans la règle (trs-p), l'interface de a possède exactement un site lié car nous voulons que le reste de l'interface, ϕ et ϕ', corresponde à des fonctions permutables, dénotant le fait que la protéine n'est pas liée à d'autres protéines. L'interface ϕ est alors permutée en $\overline{\phi}$ durant la translocation, tandis que ϕ' reste inchangée. La protéine a « disparaît » de la solution pendant l'application de (trs-p). De façon duale, cette protéine « réapparaît » pendant l'application de (trs-m). La translocation correspond à l'effet de l'application de la règle (react), synchronisant les règles (trs-p) et (trs-m).

6.4.2 Phagocytose

Comme précisé en introduction, l'endocytose est un mécanisme permettant à une cellule d'ingérer du nouveau matériel. Il existe bien évidemment une grande variété de types d'endocytose. Celui qui nous intéresse tout particulièrement correspond à l'*endocytose active* pour laquelle ce phénomène apparaît comme une réponse à une activation ou, plus généralement, un signal extérieur. Afin de la distinguer de l'endocytose générale, et comme nous destinons l'utilisation de ce phénomère à l'ingérence d'une structure complexe (comme celle du virus), nous utiliserons désormais le terme de *phagocytose*, reprenant le nom de la primitive du Brane calcul qui lui correspond.

La phagocytose de $\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}]$ – la plupart du temps une petite cellule – par $\mathrm{N}(\!|\, \mathsf{N} \,|\!)[\mathsf{T}]$ – généralement une grosse cellule – réalise le transport de la première dans le cytoplasme de la seconde en *enveloppant* $\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}]$ *avec une*

nouvelle membrane qui est extraite de N. Ce mécanisme permettant d'englober toute une cellule est problématique car cela demande de détacher une partie de la membrane de la cellule hôte, ce qui semble ne pas être un processus élémentaire. Par exemple, la transition

$$\textsc{m}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}]\;,\;\textsc{n}(\!|\,\mathsf{N}\,|\!)[\mathsf{T}] \xrightarrow{\tau} \textsc{n}(\!|\,\mathsf{N}\,|\!)[\textsc{n}'(\!|\,\emptyset\,|\!)[\textsc{m}(\!|\,\mathsf{M}\,|\!)[\mathsf{S}]]\;,\;\mathsf{T}]$$

n'est pas satisfaisante puisque la nouvelle membrane $\textsc{n}'$ est vide. Étant donné que nous ne possédons pas encore de mécanisme permettant d'ajouter des protéines dans une membrane, cette transition exclut de fait de futures complexations avec la nouvelle membrane.

En réalité, la phagocytose est un procédé qui est rendu possible lorsque la membrane de la cellule hôte a suffisamment de matériel pour créer la nouvelle membrane. C'est pourquoi nous proposons de modéliser la phagocytose comme une décomplexation de deux protéines appartenant aux membranes des cellules interagissant, avec l'effet de bord consistant en la séparation en deux de la cellule hôte en fonction de motifs prédéfinis. Comme pour la translocation, nous utilisons une nouvelle fonction $\mathfrak{P}$ associant à chaque nom de règle le n-uplet de la forme $((\mathtt{a}, i, \phi), (\mathtt{b}, j, \psi), \textsc{m}, \textsc{n}, \textsc{n}', \textsc{n}'', \mathsf{N}')$, où $\mathtt{np}(\mathsf{N}') = \emptyset$. La signification de ce n-uplet est la suivante : $(\mathtt{a}, i, \psi)$ et $(\mathtt{b}, j, \phi)$ sont les deux protéines qui prennent part à la décomplexation et sont localisées dans les deux membranes $\textsc{m}$ et $\textsc{n}$, respectivement. Le nom $\textsc{n}'$ est celui donné à la nouvelle membrane entourant la cellule ingérée ; N' est le multi-ensemble de protéines faisant partie de la nouvelle cellule.

Dans les règles qui suivent, nous étendons l'ensemble des étiquettes de transition avec $\mathtt{m}^x_{\mathfrak{t}}$ et, comme précédemment, nous utilisons la notation μ lorsque nous ne voulons pas spécifier la forme de l'étiquette. Nous dénotons par $\uplus$ l'union disjointe sur les ensembles. Le mécanisme de phagocytose est régi par les règles définies dans la figure 6.6.

La règle (pgo-e) définit la transition relative à une cellule ingérée. L'étiquette $\mathtt{a}^x_{\mathfrak{t}}$ de la transition relative à la membrane devient $\mathtt{m}^x_{\mathfrak{t}}$ dans la transition relative à la cellule. Ceci permet de spécifier la phagocytose dans l'étiquette même. On procède de manière similaire pour la règle (pgo-i). Dans (pgo-i), la partie du multi-ensemble de protéines qui va servir de membrane à la nouvelle cellule englobant la cellule ingérée est effacée de la cellule hôte. Ce multi-ensemble est restitué dans la règle (phago).

Il n'est toujours pas clair pour nous si la règle présentée ci-dessus est proche ou non de son implémentation biologique. On peut cependant noter que (pgo-i) est, d'un point de vue calculatoire, extrêmement extensive, du moins comparée aux autres opérations décrites dans le présent chapitre. D'après (pgo-i), extraire de la membrane un multi-ensemble de protéines, dont le motif est décrit par la fonction $\mathfrak{P}$, requiert une vérification complexe de la membrane en terme de nombre de protéines. Nous laissons le soin à un travail futur de déterminer s'il

$$\text{(PGO-E)}\quad \frac{\mathsf{M} \xrightarrow{\mathtt{a}_{\mathfrak{r}}^{x}} \mathsf{M}' \quad \mathfrak{P}(\mathfrak{r}) = ((\mathtt{a}, i, \phi, \phi'), (\mathtt{b}, j, \psi, \psi'), \mathrm{M}, \mathrm{N}, \mathrm{N}', \mathrm{N}'', \mathsf{N}')}{\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}] \xrightarrow{\mathtt{m}_{\mathfrak{r}}^{x}} \wr \mathsf{M}' \,;\, \mathsf{S} \int \cdot \emptyset}$$

$$\text{(PGO-I)}\quad \frac{\mathsf{M} \xrightarrow{\mathtt{b}_{\mathfrak{r}}^{x}} \prod_{k \in I \uplus J} \mathtt{b}_k \langle \sigma_k \rangle \quad \mathfrak{P}(\mathfrak{r}) = ((\mathtt{a}, i, \phi, \phi'), (\mathtt{b}, j, \psi, \psi'), \mathrm{M}, \mathrm{N}, \mathrm{N}', \mathrm{N}'', \prod_{j \in J} \mathtt{b}_j \langle \sigma_j \rangle)}{\mathrm{N}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}] \xrightarrow{\mathtt{n}_{\mathfrak{r}}^{x}} \wr \prod_{i \in I} \mathtt{b}_i \langle \sigma_i \rangle \;;\, \mathsf{S} \int \cdot \emptyset}$$

$$\text{(PHAGO)}\quad \frac{\begin{array}{c}\mathsf{S} \xrightarrow{\mathtt{m}_{\mathfrak{r}}^{x}} \wr \mathsf{M} \,;\, \mathsf{S}'' \int \cdot \mathsf{S}' \qquad \mathsf{T} \xrightarrow{\mathtt{n}_{\mathfrak{r}}^{x}} \wr \mathsf{N} \,;\, \mathsf{T}'' \int \cdot \mathsf{T}' \\ \mathfrak{P}(\mathfrak{r}) = ((\mathtt{a}, i, \phi, \phi'), (\mathtt{b}, j, \psi, \psi'), \mathrm{M}, \mathrm{N}, \mathrm{N}', \mathrm{N}'', \mathsf{N}')\end{array}}{\mathsf{S} \,,\, \mathsf{T} \xrightarrow{\tau} \mathsf{S}' \,,\, \mathsf{T}' \,,\, \mathrm{N}'(\!|\, \mathsf{N} \,|\!)[\mathrm{N}''(\!|\, \mathsf{N}' \,|\!)[\mathrm{M}(\!|\, \mathsf{M} \,|\!)[\mathsf{S}'']] \,,\, \mathsf{T}'']}$$

FIG. 6.6 – Réductions modélisant la phagocytose

est possible ou non de définir une règle d'interaction plus élémentaire de la phagocytose.

Nous pouvons enfin terminer la modélisation de l'exemple de la section 6.3.2. Nous présentons dans la figure 6.7 l'ensemble des règles nécessaires pour faire évoluer la solution initiale $\mathsf{Virus} \,,\, \mathsf{Cell}$ jusqu'au moment où la nucléocapside est relachée dans le cytoplasme de la cellule infectée.

$$\begin{array}{lcrl}
\mathfrak{r}_1 & : & ((\mathtt{ha}, 1, \emptyset), (\mathtt{gly}, 1, \emptyset)) & \in \mathfrak{C} \\
\mathfrak{r}_1{}' & : & (\mathtt{gly}_{\mathfrak{r}_1}, \mathrm{CLL}) \mapsto \mathrm{ACLL} & \in \mathfrak{A} \\
\mathfrak{r}_2 & : & ((\mathtt{ha}, 1, \emptyset), (\mathtt{gly}, 1, \emptyset)), \mathrm{VS}, \mathrm{ACLL}, \mathrm{CLL}, \mathrm{VES}, (\mathtt{gly}\langle 1 \rangle) & \in \mathfrak{P} \\
\mathfrak{r}_3 & : & ((\mathrm{EDSM}, \mathrm{VES}), \mathrm{EDSM}) & \in \mathfrak{F} \\
\mathfrak{r}_4 & : & ((\mathrm{EDSM}, \mathrm{VS}), \mathrm{EDSM}) & \in \mathfrak{F}
\end{array}$$

FIG. 6.7 – Règles d'interaction modélisant l'infection d'un virus de type influenza

Le rendu formel de l'exemple de la secion 6.3.2 peut désormais être présenté à partir de la solution initiale, c'est-à-dire avant que la phagocytose n'ait lieu. C'est ce qui est présenté dans la figure 6.8.

$$
\begin{array}{lll}
 & \textsc{vs}(\!|\, \mathtt{ha}\langle 1\rangle \,|\!)[\mathsf{Nucaps}] \;,\; \textsc{cll}(\!|\, \mathtt{gly}\langle 1\rangle \;,\; \mathsf{M_c} \,|\!)[\mathsf{Endosome}\;,\;\mathsf{Cytosol}] & \\
\xrightarrow{\tau} & \textsc{vs}(\!|\, \mathtt{ha}\langle 1^x\rangle \,|\!)[\mathsf{Nucaps}] \;,\; \textsc{acll}(\!|\, \mathtt{gly}\langle 1^{\mathsf{x}}\rangle \;,\; \mathsf{M_c} \,|\!)[\mathsf{Endosome}\;,\;\mathsf{Cytosol}] & (\mathfrak{r}_1 - \mathfrak{r}_1{}') \\
\xrightarrow{\tau} & \textsc{cll}(\!|\, \mathsf{M}_c \,|\!)[\textsc{ves}(\!|\, \mathtt{gly}\langle 1\rangle \,|\!)[\textsc{vs}(\!|\, \mathtt{ha}\langle 1\rangle \,|\!)[\mathsf{Nucaps}]]\;,\;\mathsf{Endosome}\;,\;\mathsf{Cytosol}] & (\mathfrak{r}_2) \\
\xrightarrow{\tau} & \textsc{cll}(\!|\, \mathsf{M}_c \,|\!)[\textsc{edsm}(\!|\, \mathsf{M}_e \;,\; \mathtt{gly}\langle 1\rangle \,|\!)[\mathsf{Virus}\;,\;\mathsf{E_s}]\;,\;\mathsf{Cytosol}] & (\mathfrak{r}_3) \\
\equiv & \textsc{cll}(\!|\, \mathsf{M}_c \,|\!)[\textsc{edsm}(\!|\, \mathsf{M}_e \;,\; \mathtt{gly}\langle 1\rangle \,|\!)[\textsc{vs}(\!|\, \mathtt{ha}\langle 1\rangle \,|\!)[\mathsf{Nucaps}]\;,\;\mathsf{E}_s]\;,\;\mathsf{Cytosol}] & \\
\xrightarrow{\tau} & \textsc{cll}(\!|\, \mathsf{M}_c \,|\!)[\textsc{edsm}(\!|\, \mathsf{M}_e \;,\; \mathtt{gly}\langle 1\rangle \;,\; \mathtt{ha}\langle 1\rangle \,|\!)[\mathsf{E}_s]\;,\;\mathsf{Nucaps}\;,\;\mathsf{Cytosol}] & (\mathfrak{r}_4)
\end{array}
$$

FIG. 6.8 – Déroulement des premières étapes de l'infection virale

6.5 Conclusions

Nous avons présenté dans ce chapitre un cadre formel pour la modélisation des protéines et des cellules. Les interactions entre protéines sont soit des complexations soit des décomplexations. Les interactions entre cellules dans bioκ-calcul permettent de décrire les fusions de membranes. Toutes les interactions sont élémentaires, en ce sens qu'elles impliquent toujours deux protéines au plus. Les fusions ont été définies à l'aide d'une sémantique d'ordre supérieur basée sur le travail présenté dans [San93]. Nous avons étudié la sémantique opérationnelle du bioκ-calcul et une sémantique extensionnelle de ce même langage – la bisimulation. L'expressivité a été analysée en modélisant deux systèmes biologiques importants.

Quelques extensions des règles de bioκ-calcul peuvent être définies sans difficultés. Nous avons discuté dans la précédente section le cas de la translocation et de la phagocytose, même si cette dernière ne nous semble pas encore satisfaisante. D'autres extensions n'ont pas été présentées mais il serait facile de les ajouter. Nous pourrions, par exemple, autoriser une protéine à établir une connexion entre deux de ses sites, ce que l'on nomme *auto-complexation*. Il suffirait pour cela de définir une nouvelle fonction $\mathcal{A}$ retournant des n-uplets de la forme $(\mathtt{a}, i, j, \phi_1, \phi_2)$ et de rajouter les règles de réduction

$$\text{(AUTO-1)}\quad \frac{(\mathtt{a}, i, j, \phi_1, \phi_2) \in \mathcal{A}(\mathfrak{r}) \quad x \notin \mathtt{nc}(\sigma)}{\mathtt{a}\langle i + \phi_1 + \phi_2 + \sigma\rangle \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathtt{a}\langle i^x + \overline{\phi_1} + \phi_2 + \sigma\rangle}$$

$$\text{(AUTO-2)}\quad \frac{(\mathtt{a}, i, j, \phi_1, \phi_2) \in \mathcal{A}(\mathfrak{r}) \quad x \notin \mathtt{nc}(\sigma)}{\mathtt{a}\langle j + \phi_1 + \phi_2 + \sigma\rangle \xrightarrow{\mathtt{a}^x_{\mathfrak{r}}} \mathtt{a}\langle j^x + \overline{\phi_1} + \phi_2 + \sigma\rangle}$$

qui permettent d'observer une possible auto-complexation sur les deux sites d'une protéine. L'interaction proprement dite est alors décrite par la τ-reduction suivante :

$$\text{(AUTO)}\quad \frac{(\mathtt{a}, i, j, \phi_1, \phi_2) \in \mathcal{A}(\mathfrak{r}) \quad x \notin \mathtt{nc}(\sigma)}{\mathtt{a}\langle i + j + \phi_1 + \phi_2 + \sigma\rangle \xrightarrow{\tau} \mathtt{a}\langle i^x + j^x + \overline{\phi_1} + \phi_2 + \sigma\rangle}$$

De même nous avons vu que tous les changements relatifs aux états internes des protéines étaient consécutifs à une interaction de type complexation ou décomplexation. Or nous pourrions relacher cette contrainte en permettant à deux protéines connectées de modifier leurs interfaces respectives sans pour autant réaliser une décomplexation. Ceci traduirait assez élégamment les phénomènes de propagation de modifications dus à une interaction ayant lieu entre deux protéines n'étant pas directement en contact avec les protéines concernées. On sait, par exemple, qu'une interaction impliquant une protéine faisant partie d'un complexe protéique a des répercussions sur les capacités d'interaction de ce dernier, notamment en reconfigurant sa structure tridimensionnelle et en révelant ainsi de nouveaux sites jusque là cachés par le repliement du complexe. Cette propagation des conséquences de l'interaction pourtant très localisée peut se voir comme une *mise à jour* des capacités d'interaction globales du complexe et il semble naturel d'utiliser les connexions déjà établies entre les protéines du complexe pour représenter cette propagation. Ainsi, en définissant une nouvelle fonction $\mathfrak{U}$ similaire aux fonctions $\mathfrak{C}$ et $\mathfrak{D}$, à savoir retournant des n-uplets de la forme $((\mathtt{a}, i, \phi_1, \phi_2), (\mathtt{b}, j, \psi_1, \psi_2))$ et en ajoutant la règle suivante dès que la condition $((\mathtt{a}, i, \phi_1, \phi_2), (\mathtt{b}, j, \psi_1, \psi_2)) \in \mathfrak{U}$

est vérifiée :

$$\begin{gathered}\mathtt{a}\langle i^x + \phi_1 + \phi_2 + \sigma\rangle \ , \ \mathtt{b}\langle j^x + \psi_1 + \psi_2 + \sigma'\rangle \\ \downarrow \tau \\ \mathtt{a}\big\langle i^x + \overline{\phi_1} + \phi_2 + \sigma\big\rangle \ , \ \mathtt{b}\big\langle j^x + \overline{\psi_1} + \psi_2 + \sigma'\big\rangle\end{gathered}$$

ainsi que les règles similaires tenant compte de la possible position des protéines dans des membranes cellulaires. À la différence de l'extension concernant l'auto-complexation des protéines, il est naturel que les interactions relatives à des mises à jour d'un complexe protéique ne soient pas observables. C'est pourquoi seule la τ-transition est définie.

D'autres interactions biologiques, telles que celles présentées dans [Car05] et [DP04], n'ont pas encore été considérées et sont laissées à un travail ultérieur.

Les sémantiques extensionnelles de $\mathtt{bio}\kappa$-calcul posent des questions intéressantes qui seront étudiées dans de prochains travaux. En particulier les outils mathématiques et les techniques qui permettent d'établir des propriétés sur les solutions biologiques. De tels outils pourraient par exemple être utilisés pour prédire les résultats d'expériences *in vitro*.

Conclusions et perspectives

Dans cette thèse, nous avons présenté un langage, le $g\kappa$-calcul, proche de la famille des calculs de processus. Une des particularités de ce langage est de rompre avec la dissymétrie habituelle entre émetteur et récepteur et de fournir un cadre général pour l'étude des phénomènes d'auto-assemblage. Ceci nous a permis de proposer des algorithmes distribués réalisant l'assemblage décentralisé de systèmes d'agents correspondant à des spécifications exprimées sous forme de réécritures d'arbres aussi bien que de graphes. La capacité du langage à pouvoir exprimer à la fois la spécification et sa réalisation dans un même cadre théorique a rendu la notion de correction plus claire que dans les approches habituelles. Dans une seconde partie, nous avons présenté une variante de ce langage, à même de modéliser une partie importante de la biologie décrite au niveau moléculaire.

Des questions émergent naturellement de ce travail. En particulier, la notion de réversibilité abordée lors des problèmes d'auto-assemblage énoncés dans les chapitres 3 et 4 ouvre la voie à une étude théorique plus approfondie sur l'intégration éventuelle de mécanismes de retour arrière au sein même du langage[5]. Le traitement qui est fait des situations d'impasse dans le cas des assemblages de graphes indique en effet qu'une approche plus générale est nécessaire afin d'obtenir une gestion fine de ces situations. On imagine en réalité deux manières de procéder sur ce point. Une première solution serait de mimer l'alternative proposée en conclusion du chapitre 3, en plongeant l'algorithme décrit dans $g\kappa$-calcul dans un formalisme plus traditionnel, de type π-calcul, mais équipé d'une infrastructure de retour arrière. Une étude réalisée par Danos, Krivine et Tarissan a déjà permis de montrer comment cette approche apportait une solution légèrement différente de celle proposée dans la section 4.3, répondant avec plus de souplesse à la spécification de l'assemblage de graphes. Le π-calcul est en effet nécessaire dans ce cas, puisque, à la différence de l'assemblage d'arbres qui ne requiert que des synchronisations, il est indispensable dans le cas des graphes de transmettre un certain nombre d'informations (image de la composante, identifiant de groupe, fonction de mise à jour) afin de maintenir la cohérence des structures assemblées.

[5] perspective qu'il faut rapprocher de travaux tels que ceux de [DK04] ou [PU06].

Dans le cas des arbres, les deux codes – celui gérant directement les impasses par l'ajout de règles d'interaction dans le système d'agents et celui utilisant une version réversible de CCS – se trouvaient être totalement équivalents. Le comportement en arrière décrit par les règles REV dans le $g\kappa$-calcul produit en effet la même dynamique que celle fournie par le mécanisme de backtrack du langage RCCS. Dans le cas des graphes, par contre, le comportement s'avère totalement différent puisque la solution proposée, en utilisant seulement le cadre fourni par le $g\kappa$-calcul, oblige les composantes bloquées à supprimer toutes les connexions pour repartir de l'état initial. Le traitement suggéré par le plongement de l'algorithme dans un π-calcul réversible permet en revanche une gestion plus fine du retour arrière.

Il semble cependant évident que la facilité à proposer de tels algorithmes et à prouver leur adéquation avec la spécification dépend fortement de l'adaptation du cadre utilisé pour les décrire. Or, et c'est un argument qui justifie à la base le développement du $g\kappa$-calcul, les algèbres de processus du type π-calcul ne sont pas particulièrement adaptés à de tels cas. À l'inverse de celui relativement simple des assemblages d'arbres pour lesquels il était possible de fournir directement un algorithme réécrit en CCS, un tel algorithme dans le cas des graphes ne semble pas envisageable sans la description au préalable de celui-ci dans un formalisme de type $g\kappa$-calcul. C'est pourquoi il paraît légitime de se pencher sur la seconde alternative en matière de gestion des impasses. Elle consisterait à équiper directement le $g\kappa$-calcul d'un mécanisme de backtrack similaire à ceux proposés pour CCS ou le π-calcul. Ceci constitue une voie prometteuse pour les questions relatives à l'auto-assemblage.

Cette extension trouverait par ailleurs un écho naturel dans une perspective biologique, comme l'a montré [DK03], puisque les liaisons entre protéines au sein d'un complexe sont des liaisons généralement faibles et que celles-ci ne s'établissent qu'après différents tâtonnements, formant des structures partiellement instables. Un formalisme équipé de retour arrière permettrait alors de rendre directement compte de ce type de phénomènes, sans pour autant encombrer la description des systèmes étudiés avec des règles d'interaction explicitant ces tâtonnements. D'autre part, en reprenant la perspective d'utilisation de la bisimulation en tant qu'outil de validation de modèle, il devient possible d'apporter une preuve supplémentaire que de telles interactions observées ne régissent que la réalité de ces interactions instables et que les modèles peuvent s'en abstraire sans perdre toute leur puissance descriptive du point de vue de leur comportement.

Un autre champ d'exploration ouvert par cette étude est porté par l'implémentation proposée au chapitre 5. Dans celle-ci, une variante de l'algorithme présenté au chapitre 4 est implémentée en Ocaml et produit un programme qui permet la synthèse des règles réalisant l'assemblage hautement distribué

de graphes, à partir de la seule description de scénarios de construction. Dans cette implémentation, le système a conscience de l'espace dans lequel les agents sont plongés. Le programme raffine donc en quelque sorte les règles du calcul d'agents en les assujettissant à ne s'appliquer qu'entre agents capables de se percevoir. Ce sont là des problèmes d'auto-organisation dans un véritable espace euclidien qui sont des phénomènes intéressants et qui pourraient constituer une direction de recherche future.

Bibliographie

[Car05] Luca Cardelli. Brane calculi. In V.Danos and V.Schachter, editors, *Computational Methods in Systems Biology*, volume 3082 of *Lecture Notes in Computer Science*, pages 257–278. Springer, 2005.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoritical Computer Science*, 240(1) :177–213, 2000.

[DK03] Vincent Danos and Jean Krivine. Formal molecular biology done in CCS-R. In *Bio-Concur'03, satellite workshop of CONCUR'03*, 2003.

[DK04] Vincent Danos and Jean Krivine. Reversible communicating systems. In Gardner and Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307. Springer-Verlag, Sep 2004.

[DK05] Vincent Danos and Jean Krivine. Transactions in rccs. In *Sixteenth International Conference on Concurrency Theory, CONCUR'05*, volume 3653 of *LNCS*. Springer-Verlag, August 2005.

[DKT06] Vincent Danos, Jean Krivine, and Fabien Tarissan. Self assembling trees. In *Proceedings of the Workshop on Structural Operational Semantics (SOS'06)*. Elsevier, 2006. To appear.

[DL03] Vincent Danos and Cosimo Laneve. Core formal molecular biology. In *Proceedings of the 12th European Symposium on Programming (ESOP'03)*, volume 2618 of *Lecture Notes in Computer Science*, pages 302–318, Warsaw, Poland, April 2003. Springer-Verlag.

[DL04] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoritical Computer Science*, 325(1) :69–110, 2004.

[DP04] Vincent Danos and Sylvain Pradalier. Projective brane calculus. In *Computational Methods in Systems Biology*, pages 134–148, 2004.

[DS03] Eric Drexler and Richard Smalley. Controversy about molecular assemblers. Available at `www.foresight.org/NanoRev/Letter.html`, 2003.

[DT05] Vincent Danos and Fabien Tarissan. Self-assembling graphs. In José Mira and José R. Álvarez, editors, *Mechanisms, Symbols, and*

Models Underlying Cognition, volume 3561 of *Lecture Notes in Computer Science*, pages 498–507. Springer, June 2005.

[FB96] Walter Fontana and Leo W. Buss. The barrier of objects : From dynamical systems to bounded organizations. In J.Casti and A.Karlqvis, editors, *Boundaries and Barriers*, pages 55–116. Addison-Wesley, 1996.

[Gil76] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22 :403–434, 1976.

[Gil77] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem*, 81 :2340–2361, 1977.

[HMC02] Jeff Hasty, David McMillen, and James J. Collins. Engineered gene circuits. *Nature*, 420 :224–230, November 2002.

[Kla02] Eric Klavins. Automatic synthesis of controllers for assembly and formation forming. In *Proceedings of the International Conference on Robotics and Automation*, 2002.

[LT06] Cosimo Laneve and Fabien Tarissan. A simple calculus for proteins and cells. In *Proceedings of Workshop on Membrane Computing and Biologically Inspired Process Calculi (MECBIC'06)*, 2006. To appear.

[Mil89a] Robin Milner. *A Calculus of Communicating Systems*. Prentice Hall, 1989.

[Mil89b] Robin Milner. *Communication and Concurrency*. International Series on Computer Science. Prentice Hall, 1989.

[Mil99] Robin Milner. *Communicating and mobile systems : the π-calculus*. Cambridge University Press, Cambridge, 1999.

[Nag02] Radhika Nagpal. Programmable self-assembly using biologically-inspired multiagent control. In *Autonomous Agents and Multiagent Systems Conference (AAMAS)*, July 2002.

[Par81] David Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proceedings of the 5th GI-Conference on Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer-Verlag, Berlin/New York, 1981.

[Pau02] Gh. Paun. Membrane computing. an introduction. Berlin, 2002. Springer-Verlag.

[Pri95] Corrado Priami. Stochastic pi-calculus. *The Computer Journal*, 38(7) :578–589, 1995.

[PRSS01] Corrado Priami, Aviv Regev, Ehud Shapiro, and William Silverman. Application of a stochastic name-passing calculus to representation and simulation of molecular processes. *Information Processing Letters*, 80 :25–31, 2001.

[PU06] Iain Phillips and Irek Ulidowski. Reversing algebraic process calculi. In *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS 2006)*, volume 3921 of *Lecture Notes in Computer Science*, pages 246–260, Vienna/Austria, March 2006.

[RPS+04] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. Bioambients : an abstraction for biological compartments. *Theoretical Computer Science*, 325(1) :141–167, 2004.

[RSS01] Aviv Regev, William Silverman, and Ehud Shapiro. Representation and simulation of biochemical processes using the π-calculus process algebra. In R. B. Altman, A. K. Dunker, L. Hunter, and T. E. Klein, editors, *Pacific Symposium on Biocomputing*, volume 6, pages 459–470, Singapore, 2001. World Scientific Press.

[San93] Davide Sangiorgi. From π-calculus to Higher-Order π-calculus — and back. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. TAPSOFT'93*, volume 668 of *Lecture Notes in Computer Science*, pages 151–166, 1993.

[San98] Davide Sangiorgi. On the bisimulation proof method. *Mathematical Structures in Computer Science*, 8(5) :447–479, 1998.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus. A Theory of Mobile Processes*. Cambridge University Press, 2001.

Annexe A

Code source partiel

```
(*************************************************************
 *                Auto-assemblage de graphes                 *
 *************************************************************
   Auteur  : Fabien TARISSAN
   Fichier : agent.ml
   Date    : 09/06/05
 *************************************************************
   Description : Syntaxe abstraite pour les agents
 *************************************************************)


EXCEPTION Agent_Error OF string;;

(* Type des connexions *)
TYPE com = string
;;

(* Création d'un nouvelle connexion *)
LET create_com =
  LET str = REF "x" AND c = REF 0 IN
    FUNCTION () ->
      IF !c = max_int
      THEN (
        str := (!str)^"x";
        c := 0
      );
      c := !c+1;
      (!str)^(string_of_int(!c))
;;

(* Type de la représentation des graphes *)
TYPE graph = int * int
;;
```

```
(* Type utiliser pour propager les informations *)
(* sur la structure de la nouvelle composante   *)
TYPE shift = int array
;;

(* Type de la représentation du role d'un agent *)
TYPE role = int
;;

(* Type de l'état d'un agent *)
TYPE state =
    Passive
  | Active OF graph
  | Update OF com list * shift * bool * graph option
  | Alarme OF float
  | Final
;;

(* Type des agents *)
TYPE agent = {
  MUTABLE ident : com;       (* identifiant de groupe          *)
  MUTABLE span  : com list;  (* voisins dans l'arbre de recouvrement *)
  MUTABLE cycle : com list;  (* autres voisins                 *)
  MUTABLE role  : role;      (* rôle joué dans la composante   *)
  MUTABLE state : state;     (* état (type) de l'agent         *)

  MUTABLE pos_x : float;     (* positions de l'agent *)
  MUTABLE pos_y : float;

  MUTABLE dx : float;        (* dernier déplacement de l'agent *)
  MUTABLE dy : float
}
;;

(* Pour récupérer l'image de la composante  *)
(* connexe ainsi que les attributs relatifs *)
(* à l'alarme par le biai d'un agent actif  *)

LET recup_image a =
  MATCH a.state WITH
      Active(g) -> g
    | _ -> raise (Agent_Error "recup_image : Agent non actif")
;;
LET recup_alarme a =
  MATCH a.state WITH
      Alarme(t) -> t
    | _ -> raise( Agent_Error "Agent not in alarme mode")
;;
```

```
(*************************************************************
 *                Auto-assemblage de graphes                 *
 *************************************************************
   Auteur  : Fabien TARISSAN
   Fichier : system.ml
   Date    : 09/06/05
 *************************************************************
   Description : Syntaxe abstraite pour le système
 *************************************************************)

OPEN Agent;;
OPEN Assembly;;

(* Ce que peut faire un système pour évoluer *)
TYPE choice = BinJoin | UnJoin | Upd | Switch | Break
;;


(* Un système est controlé par six paramètres *)
TYPE system = {
  MUTABLE active  : agent list; (* agents actifs  *)
  MUTABLE update  : agent list; (* agents en phase de mise à
                                   jour *)
  MUTABLE passive : agent list; (* agents en mode passif *)
  MUTABLE break   : agent list; (* agents en mode alarme *)
  MUTABLE final   : agent list; (* agents ayant atteint une
                                   formation stable *)

  assembly : assembly           (* scénario de construction *)
}
;;



(* Choix aléatoire de la prochaine action *)
LET choose_attempt () =
  LET n = Random.int 5 IN
    IF n=0 THEN UnJoin
    ELSE IF n = 1 THEN BinJoin
    ELSE IF n = 2 THEN Upd
    ELSE IF n = 3 THEN Switch
    ELSE Break
;;
```

```
(*************************************************************
 *                Auto-assemblage de graphes                 *
 *************************************************************
   Auteur  : Fabien TARISSAN
   Fichier : assembly.ml
   Date    : 09/06/05
 *************************************************************
   Description : Syntaxe abstraite et opérations de bases
                 sur les scénarios de construction
 *************************************************************)

OPEN Agent;;

(* REPRESENTATION DU GRAPHE D'ASSEMBLAGE *)
(*****************************************)

TYPE assembly = {
  init     : graph * role ;   (* graphe initial *)
  final    : graph list;      (* liste de graphes finaux *)

  (* Liste des jonctions binaires autorisées *)
  binaire : ((graph * role *  graph * role) *
               (graph * (shift * shift))) list;

  (* Liste des jonctions internes autorisées *)
  interne : ((graph * role * role)
              * (graph * shift)) list
}
;;

(* TESTS DE JONCTIONS *)
(**********************)

(* Teste si une jonction interne est autorisée *)
LET self_join g r1 r2 assem =
  LET REC recherche_cycle l =
    MATCH l WITH
        [] -> None
      | ( (a,b,c),res )::tl ->
          IF (a,b,c) = (g,r1,r2)
          THEN Some(res,List.mem (fst res) assem.final)
          ELSE IF a < g ||
            (a = g &&
                (b < r1
                  || (b = r1 && c < r2)))
          THEN recherche_cycle tl
          ELSE None
  IN
    recherche_cycle assem.interne
```

```
;;
(* Teste si une connexion entre deux composantes *)
(* disjointes est autorisée                      *)
flet binary_join rev g1 r1 g2 r2 assem =
  LET REC recherche_binaire l =
    MATCH l WITH
        [] -> None
      | ( (a,b,c,d),(g,(s1,s2)) )::tl ->
          IF (a,b,c,d) = (g1,r1,g2,r2)
          THEN (
            IF rev
            THEN Some((g,(s1,s2)),List.mem g assem.final)
            ELSE Some((g,(s2,s1)),List.mem g assem.final)
          )
          ELSE IF a < g1 ||
            (a = g1 &&
                (b < r1  ||
                    (b = r1 &&
                        (c < g2 ||
                            (c = g2 && d < r2)))))
          THEN recherche_binaire tl
          ELSE None
  IN
    recherche_binaire assem.binaire
;;
```

Table des figures

Résumé

Dans cette thèse, nous proposons un langage formel, le $g\kappa$-calcul, issu de la famille des algèbres de processus. Ce langage se distingue notamment des langages concurrents habituels par la rupture de la dissymétrie inhérente à la notion d'émetteur et de récepteur traditionnellement considérée. Cette rupture permet alors de voir les interactions entre les éléments du langage comme des phénomènes de collisions, approche bien adaptée aux questions d'auto-organisation qui font l'objet de la première partie de cette thèse.

La question qui se pose est celle de la construction concurrente et décentralisée de formes géométriques abstraites (arbres et graphes) ainsi que de phénomènes plus génériques décrits sous forme de transfert d'information dans des systèmes à base de réécriture de graphes, éventuellement hiérarchisés dans l'optique d'une application à la biologie moléculaire. Cette première partie s'accompagne notamment d'une implémentation en Ocaml simulant un algorithme d'auto-assemblage de graphes.

Dans un second temps, nous développons un sous-ensemble du langage présenté, en enrichissant une version restreinte aux interactions binaires avec une notion de membrane et d'interactions entre membranes. Ce nouveau langage se montre à même de décrire une biologie moléculaire simplifiée, qualitative, basée sur les interactions entre protéines et membranes. Cette partie de la thèse s'attache alors à montrer la valeur descriptive de ce langage sur quelques exemples et à explorer des définitions pertinentes d'équivalence entre solutions biologiques.

Abstract

In this thesis, we propose a formal language, the $g\kappa$-calcul, stemming from the family of process algebra. This language contrasts in particular with the usual concurrent languages by the lack of dissymmetry traditionally considered between senders and receivers. This point of view entitles to see interactions between elements of the language as collision phenomena. This approach is convenient for the study of self-organisation which the first part of the thesis is devoted to.

The problem is that of the concurrent and distributed construction of abstract geometrical forms (trees and graphs) as well as more generic phenomena described as transfer of information in systems based on graph rewriting rules. Those systems may come along with a hierarchical structure in the aim of applying those techniques in the context of molecular biology. This first part is presented together with an implementation in Ocaml which simulates an algorithm for self-assembling graphs.

In a second step, we develop a subset of the former language by adding a notion of membranes and interactions between membranes to a version restricted to binary interactions. This new language turns out to be suitable for describing a simplified qualitative version of the molecular biology based on interactions between proteins and membranes. This part focuses on showing the descriptive value of this new language by means of some examples and explores relevant definitions of equivalences between biological solutions.