Un calcul de réécriture de graphes : applications à la biologie et aux systèmes autonomes

Oana-Maria Andrei

Nancy-Université INPL

Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAEM Lorraine

Un calcul de réécriture de graphes : applications à la biologie et aux systèmes autonomes

THÈSE

présentée et soutenue publiquement le 5 Novembre 2008

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine

(spécialité informatique)

par

Oana Andrei

Composition du jury

Rapporteurs :	Jean-Pierre Banâtre	Professeur, Université de Rennes 1, France
	Jean-Louis Giavitto	Directeur de Recherche, IBISC, CNRS, France
Examinateurs :	Paolo Baldan	Professeur, Université de Padova, Italie
	Horatiu Cirstea	Maître de Conférences, Université Nancy 2, France
	Marie-Dominique Devignes	Chargée de Recherche CNRS, Habilitée, Nancy, France
	Hélène Kirchner	Directeur de Recherche, INRIA Bordeaux, France
	Dorel Lucanu	Professeur, Université "Al.I.Cuza", Iaşi, Roumanie
	Jean-Yves Marion	Professeur, École des Mines de Nancy, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Mis en page avec LaTeX

Table des matières

Introduction 1

I Un Calcul Biochimique Abstrait 13

1 Notions Préliminaires 15
1.1 Relations binaires et leurs propriétés . . . 15
1.2 Graphes . . . 16
1.3 Systèmes de Réduction Abstraits . . . 17
1.4 Réécriture de Termes . . . 18
1.4.1 Algèbre de termes . . . 18
1.4.2 Théories Équationelles . . . 19
1.4.3 Réécriture de Termes . . . 20
1.5 Éléments de Théorie des Catégories . . . 22
1.6 Transformation de Graphe . . . 23
1.7 Réécriture Stratégique . . . 25

2 Motivation et Vue d'Ensemble du Calcul 29
2.1 Le γ-calcul et HOCL . . . 29
2.2 Le ρ-calcul . . . 30
2.3 Vers un Calcul Biochimique Abstrait . . . 32
2.4 Structure de la Partie . . . 33

3 Syntaxe 35
3.1 Objets structurés . . . 35
3.2 Abstractions . . . 36
3.3 Molécules Abstraites . . . 37
3.4 Sous-objets, Sous-molécules, Substitutions, Filtrage . . . 38
3.5 Mondes . . . 40
3.6 Structures de Mondes ou Multivers . . . 41
3.7 Vue d'Ensemble de la Syntaxe de Base du Calcul . . . 41

4 Sémantique à Petit Pas 43
4.1 Sémantique de Base . . . 43
4.2 Rendre l'Application Explicite . . . 44
4.3 Au sujet de la Confluence Locale . . . 45

4.4 D'abord refroidir, puis réchauffer . . . 45
4.5 Vue d'ensemble de la Sémantique du Calcul avec Application Explicite . . 46

5 Ajouter des Stratégies au Calcul 47
5.1 Des Stratégies sous Formes d'Abstractions . . . 47
5.2 Appel-par-Nom dans le Calcul avec Stratégies . . . 49
5.3 Correction de l' Encodage des Stratégies sous Forme d'Abstractions . . . 49
5.4 Étendre la Sémantique avec des Stratégies et une Récupération d'Échec . 51
5.5 Stratégies Persistentes . . . 52
5.6 Vue d'ensemble de la Syntaxe et de la Sémantique du Calcul avec Stratégies 53

6 Rélation de Reduction à Gros Grain 55

7 Stratégies Possibles pour le Calcul 57

8 Comparaison avec le γ-Calcul et HOCL 59

9 Conclusion 61

II Une Instance du Calcul Biochimique Abstrait Basée sur des Graphes 63

10 Réécriture de Graphes avec Ports 65
10.1 Graphes avec Ports . . . 66
10.2 Morphismes de Graphes avec Ports et Morphismes de Nœuds . . . 67
10.3 Filtrage . . . 69
10.3.1 Définition Générale . . . 69
10.3.2 Un algorithme de sous-filtrage . . . 70
10.4 Règles de Réécriture . . . 74
10.5 Relation de Réécriture . . . 75
10.6 Réécriture Stratégique de Graphe avec Ports . . . 77
10.7 Graphes avec Ports Faibles . . . 78
10.8 Sur la Confluence de la Réécriture de Graphe avec Ports . . . 80
10.9 Comparaison avec les Systèmes Réactifs de Bigraphes . . . 83
10.10 Conclusions . . . 84

11 Le ρ_{pg}-Calcul : Un Calcul Biochimique Basé sur la Réécriture Stratégique de Graphe avec Ports 87
11.1 Syntaxe . . . 87
11.2 Sémantique . . . 92
11.2.1 Règles d'Évaluation sous forme de Règles de Réécriture de Graphes avec Ports . . . 92
11.2.2 Mécanisme d'Application sous forme de Règle de Réécriture de Graphe avec Ports . . . 93
11.3 Conclusions . . . 94

Conclusion de la Thèse et Travaux Futurs 95

III Applications. Implantation. Extensions 101

12 Case Studies for the ρ_{pg}-calcul 103
12.1 Autonomic Computing . . . 103
12.1.1 Strategy-Based Modeling of Self-Management . . . 104
12.1.2 Towards Embedding Runtime Verification in the Model . . . 108
12.2 Molecular Graphs. Biochemical Networks . . . 110
12.2.1 Modeling Molecular Complexes as Port Graphs . . . 111
12.2.2 Biochemical Network Generation by Strategic Rewriting . . . 113
12.2.3 Comparisons with Related Formalisms . . . 115
12.3 Conclusions . . . 116

13 Term Rewriting Semantics for Port Graph Rewriting 119
13.1 Term Encoding of Port Graphs . . . 120
13.1.1 An Algebraic Signature for Port Graphs . . . 120
13.1.2 A Term Algebra for Port Graphs . . . 120
13.2 pg-Rewrite Rules . . . 122
13.3 Extending the pg-Rewrite Rules . . . 122
13.4 Auxiliary Operations and Reduction Relations . . . 123
13.5 The pg-Rewriting Relation . . . 126
13.6 Operational Correspondence . . . 127
13.7 Relation to the ρ-Calculus . . . 128
13.8 Conclusions . . . 129

14 Runtime Verification in the ρ_{pg}-Calculus 131
14.1 CTL for Port Graphs and Port Graph Rewriting . . . 132
14.1.1 Port Graph Expressions . . . 133
14.1.2 Structural Formulas . . . 133
14.1.3 State and Path Formulas . . . 135
14.2 Embedding Verification in the ρ_{pg}-Calculus : the ρ^{v}_{pg}-Calculus . . . 138
14.2.1 Syntax . . . 138
14.2.2 Semantics . . . 141
14.2.3 Application in Modeling Autonomous Systems . . . 148
14.3 Conclusions . . . 149

A Implementation of the EGFR Signaling Pathway Fragment using TOM 151

Bibliographie 159

Introduction

La Motivation : des Calculs Chimiques aux Calculs Biochimiques

Depuis les prémisses de l'informatique, des chercheurs se sont intéressés aux modèles informatiques inspirés par la nature, ces travaux ont mené, par exemple, aux réseaux neuronaux [MP43], aux automates cellulaires [Neu66], et aux systèmes de Lindenmayer [Lin68]. Tandis que le développement de l'informatique théorique a accéléré, la simplicité des principes de base de la chimie ont poussé les chercheurs à abstraire un paradigme informatique pour programmer en termes de molécules, solutions de molécules et réactions, ce qui a été appelé modèle de programmation chimique ou métaphore chimique. Dans ce qui suit, nous passons en revue ce paradigme informatique, puis nous présentons une manière de déplacer le modèle à une dimension biologique, en considérant des molécules structurées. Le résultat est un calcul biochimique abstrait qui peut être instancié pour différentes structures et qui peut être étendu avec des éléments de vérification.

La Métaphore Chimique

La métaphore calculatoire chimique est utilisée dans la conception de paradigmes de calcul depuis près de 30 ans. Cette métaphore décrit le calcul en termes de solution chimique dans laquelle les molécules représentant des données agissent l'une sur l'autre librement selon des règles de réaction. Des solutions chimiques sont représentées par des multi-ensembles et le calcul procède par réécriture, qui consomme des éléments et en produit des nouveaux selon des règles précises. Plusieurs réactions se produisent en parallèle si elles ne concourent pas pour les mêmes données. Par conséquent les multi-ensembles représentent la structure fondamentale des modèles de calcul chimiques. Le modèle calculatoire chimique a été proposé par [BM86] en utilisant le formalisme Gamma. Le but de ce travail était de capturer l'intuition du calcul comme évolution globale d'une collection de valeurs atomiques agissant l'une sur l'autre librement. La généralité des règles assure une grande puissance expressive et, d'une façon directe, une universalité calculatoire. Plus généralement, les multi-ensembles structurés définis dans [FM98] peuvent être vus comme moyens syntaxiques permettant l'organisation des données explicites, et fournissant une notation menant aux programmes de plus haut niveau manœuvrant des structures de données plus complexes.

La machine chimique abstraite (CHAM) [BB92] étend le formalisme Gamma en introduisant la notion de sous-solution incluse dans une membrane, ainsi qu'une classification des règles comme des règles de chauffage (pour réarranger une solution de telle sorte que la réaction puisse avoir lieu), des règles de refroidissement (pour enlever les molécules inutiles après qu'une réaction ait eu lieu), ou des règles ordinaires de réaction. Ce forma-

lisme a été conçu comme modèle concurrent et comme manière spécifique pour définir la sémantique opérationnelle des systèmes concurrents.

Dans AlChemy [FB96], les molécules sont des λ-termes normalisés [Bar84] et une réaction entre deux molécules correspond à une β-réduction. La motivation fondamentale de ce système calculatoire était de développer un modèle formel pour des systèmes auto-organisants inspirés par les systèmes biologiques.

Le γ-calcul [BFR04, Rad07] a été conçu comme un calcul d'ordre supérieur développé sur les éléments de base du paradigme chimique. Il généralise le modèle chimique en considérant également les réactions comme des molécules. Le langage de programmation chimique d'ordre supérieur (HOCL) [BFR06c, BFR06a, BFR07] étend le γ-calcul par des éléments de programmation. On a démontré que ces formalismes sont appropriés pour modéliser des systèmes autonomes et pour la programmation des grilles.

Les systèmes de membrane ou les P-systèmes [Pau02] sont un autre exemple de modèle chimique. Ils représentent un modèle abstrait du calcul parallèle et distribué, inspirés par des compartiments de cellules et des membranes moléculaires. Une cellule est divisée en divers compartiments, chaque compartiment ayant une tâche différente, et coopérant pour accomplir simultanément une tâche plus générale pour le système entier. Les membranes d'un P-système déterminent des régions où des multi-ensembles des objets et des règles d'évolution peuvent être placés. Les objets évoluent selon les règles liées à chaque région, et les régions coopèrent afin de maintenir un comportement approprié du système entier. Les P-systèmes fournissent une abstraction convenable pour les systèmes parallèles, et un cadre approprié pour des algorithmes distribués et parallèles. Le calcul de membranes est directement inspiré par la biologie des cellules et emploie des idées nouvelles et utiles : localisation, structures hiérarchisées, distribution et communication. Les P-systèmes fournissent un modèle de calcul élégant et puissant, capable de résoudre des problèmes calculatoires durs dans un temps raisonnable et utile pour modéliser des phénomènes biologiques divers [PRC08].

MGS est un autre formalisme basé sur le modèle chimique [GM01, Gia03, Spi06]. Il a été conçu pour représenter et diriger des transformations locales sur des entités structurées par des topologies abstraites [GM01]. Un ensemble d'entités organisées par une topologie abstraite s'appelle une collection topologique. Dans MGS, la collection varie des ensembles et des multi-ensembles à des types plus structurés. MGS a la capacité d'imbriquer des topologies afin de décrire les systèmes biologiques. Utilisant la transformation sur des multi-ensembles, MGS est un formalisme unifiant les modèles calculatoires biologiquement inspirés comme Gamma, les P-systèmes, les systèmes de Lindenmayer.

La réécriture de multi-ensembles se trouve au noyau de tous ces formalismes. C'est un cas particulier de réécriture de termes où les symboles de fonction sont associatifs et commutatifs. Plusieurs cadres fournissent des environnements efficaces pour appliquer des règles de réécriture de multi-ensembles, pouvant suivre des stratégies d'évaluation. Tous les formalismes mentionnés ci-dessus sont des exemples particuliers de chimie artificielle basée sur le mécanisme de réécriture. Une chimie artificielle est "un système créé par l'homme qui est semblable à un réel système chimique" [DZB01]. Formellement, une chimie artificielle est définie par l'ensemble de toutes les molécules possibles, l'ensemble des règles de collision (ou règles de réaction) représentent des interactions parmi les

molécules, et l'algorithme décrivant comment les règles sont appliquées sur un ensemble fixe de molécules.

Vers un Calcul Biochimique

Une extension naturelle de la métaphore chimique consiste à ajouter un aspect biologique en fournissant aux molécules une structure particulière et des possibilités d'association (complexation) et de dissociation (décomplexation) entre elles. Dans les cellules vivantes, les molécules comme les acides nucléiques, protéines, lipides, hydrates de carbone peuvent se combiner, selon leurs propriétés structurelles, pour former des entités plus complexes. La biochimie comme science se concentre fortement sur le rôle, la fonction, et la structure de telles molécules. Dans une représentation informatique, les structures de données qui décrivent le mieux ces molécules varient des listes à des structures plus complexes, en passant par les arbres et graphes [Car05a].

Cardelli et Zavattaro ont montré, dans [CZ08], que le passage de la chimie vers la biochimie, en employant pour les molécules une structure adéquate, capable d'exprimer des connexions entre elles, était très intéressant d'un point de vue calculatoire. Les auteurs ont montré qu'en ajoutant des possibilités de base d'association et de dissociation entre les entités (pour la complexation et la décomplexation respectivement) à un formalisme basé sur une algèbre de processus minimale pour modéliser la chimie, on augmente la puissance informatique de telle sorte que le modèle calculatoire obtenu est Turing-complet. Ce résultat nous a encouragés à croire que l'ajout des possibilités d'association et de dissociation pour des molécules représente un dispositif essentiel pour passer d'un modèle chimique minimal à un modèle biochimique. En outre, il justifie notre but de définir un calcul biochimique en étendant le modèle chimique minimal proposé par le γ-calcul avec (i) une structure pour des molécules qui permette l'expression de connexions entre les molécules, et (ii) des opérations sur ces connexions.

Un Calcul Biochimique Basé sur la Réécriture de Graphes avec Ports

Un Calcul Biochimique Abstrait

Le passage d'un modèle chimique à un modèle biochimique et le gain dans l'expressivité qu'il peut nous fournir, nous a motivé dans le travail présenté dans cette thèse. Nous proposons un calcul qui étend le γ-calcul par des possibilités plus puissantes d'abstraction qui considèrent pour le filtrage non pas seulement une variable mais une molécule structurée. Nous supposons que la structure considérée pour des molécules, en général dénotée par Σ, leur permet également de se relier. Cette approche est semblable à la définition du ρ-calcul [CK01] comme extension du λ-calcul et de la réécriture de premier ordre. Le résultat est un calcul de réécriture avec des possibilités d'ordre supérieur basées sur la métaphore chimique avec les molécules structurées ayant des possibilités de connexion et des règles de réaction sur de telles molécules ; nous l'avons appelé le $\rho_{\langle\Sigma\rangle}$-calcul. Basé sur les capacités de connectivité des molécules Σ-structurées, nous considérons le $\rho_{\langle\Sigma\rangle}$-calcul comme étant une extension biochimique du γ-calcul, d'où le nom de Calcul Biochimique Abstrait.

Les éléments de première classe du $\rho_{\langle\Sigma\rangle}$-calcul sont des objets structurés représentant les molécules, des abstractions représentant les règles de réécriture sur les molécules ou sur d'autres abstractions, et des applications d'abstraction. Les objets structurés et les abstractions sont définis au même niveau en tant que molécules. D'après les principes du modèle chimique, une juxtaposition des molécules dans un multi-ensemble représente également une molécule. Nous abstrayons l'environnement où les molécules flottent en utilisant un opérateur qui les groupe dans un monde. Une interaction entre une abstraction et une molécule peut avoir lieu de manière multiple, cela étant du à toutes les solutions de filtrage possibles entre l'abstraction et la molécule. Par conséquent un monde peut avoir plusieurs possibilités d'évolution que nous rassemblons dans une structure des mondes alternatifs appelée multiverse.

La puissance expressive élevée du $\rho_{\langle\Sigma\rangle}$-calcul nous permet de modéliser un certain contrôle sur la composition ou le choix de l'ordre d'application des règles à l'aide des notions de la stratégie et de la réécriture stratégique. Nous encodons des stratégies en tant qu'abstractions particulières et les incluons dans le calcul au même niveau que les autres molécules. En outre, les stratégies nous permettent d'exploiter l'information d'échec.

La Structure de Graphes avec Ports pour les Molécules Biologiques

Dans [AIK06], nous avons étudié l'implantation des modèles de graphes dans TOM, en nous inspirant du travail poursuivi dans le cadre du projet GasEl [BCC+03, BIK06, Iba04], cela afin de simuler un réacteur chimique. Ce projet a été développé en utilisant les systèmes basés sur des règles et des stratégies, appliqué au problème de la génération automatique des mécanismes de cinétique suivant l'approche de la chimie artificielle. Dans les deux applications, la modélisation d'un réacteur chimique [AIK06] et la modélisation des interactions de protéines [AK07], les molécules sont représentées comme des graphes où les nœuds correspondent aux atomes ou aux protéines, et les réactions sont des règles de réécriture de graphes qui créent ou cassent des liens entre les nœuds. À l'issue de ces travaux, nous avons identifié une structure de graphes où les noeuds ont plusieurs points d'accès, appelés des ports, permettant d'attacher les arêtes du graphe. Ces ports fournissent une division explicite de la connectivité des noeuds. Dans cette thèse, nous identifions une classe générale de graphes orientés avec des arêtes et des boucles multiples, et où une étiquette de noeud est un triplet composé d'un identificateur de noeud, d'un nom de noeud et d'un ensemble de ports. Une étiquette d'arête est une paire (port source, port cible). Nous appelons un tel graphe un graphe avec ports (ou multigraphes avec ports) et nous définissons une relation de réécriture stratégique appropriée sur ces graphes [AK08c]. Nous fournissons également une axiomatisation (i) des graphes avec ports et (ii) de la réécriture de graphes avec ports en utilisant une algèbre de termes de premier ordre appropriée, et une relation correspondante de réécriture de termes.

Le concept de graphes avec ports n'est pas nouveau. Il peut se voir comme un raffinement de l'information de connectivité des noeuds. Notons qu'une source d'inspiration pour notre travail sur les graphes avec ports, a été le formalisme basé sur les graphes présenté dans [BYFH06]. Ce formalisme a été défini pour modéliser les réseaux biochimiques, les complexes de protéines y sont représentées par des graphes attribués typés et les classes de réactions par des règles de transformation de graphes. Dans le même esprit, un autre formalisme d'inspiration a été le κ-calcul [DL04] ; c'est un langage de protéines formelles qui modélise les complexes moléculaires par des graphes avec sites et leurs interactions par une opération particulière de réécriture de graphes. Elle emploie une notation algébrique proche du π-calcul [Mil99] et les liens entre les protéines dans les complexes moléculaires y sont représentés par des noms partagés.

Des protéines sont abstraites par des boîtes avec des sites d'interaction sur la surface, ayant des états particuliers. Par conséquent, en raffinant l'information des ports pour contenir l'état, et en les appelant des sites avec au plus une arête attachée à chaque port, la réécriture de graphes avec ports devient appropriée à la modélisation des interactions de complexes moléculaires. Chaque site a un état indiquant la disponibilité pour une connexion. Nous appelons cette variante des graphes avec ports, utilisée pour modéliser les complexes moléculaires, des graphes moléculaires [AK07]. En Figure 0.1 nous illustrons au milieu un modèle de réaction qui, appliqué sur le graphe moléculaire de gauche, crée une arête (appelée lien dans le cadre biochimique) comme nous pouvons le voir dans le graphe moléculaire de droite. Cet exemple est extrait d'un exemple de taille plus importante, développé en Section 12.2. Dans cette section, on modélise un fragment de

la voie épidermique de signalisation du récepteur de facteur de croissance (EGFR). Les protagonistes de l'exemple sont quatre protéines de signal dénotées par S (S.S étant leur forme dimerisée), deux protéines récepteur R, et une protéine adapteur A. Les sites sont représentés différemment selon leur état : cercles remplis pour les sites attachés et cercles vides pour les sites libres.

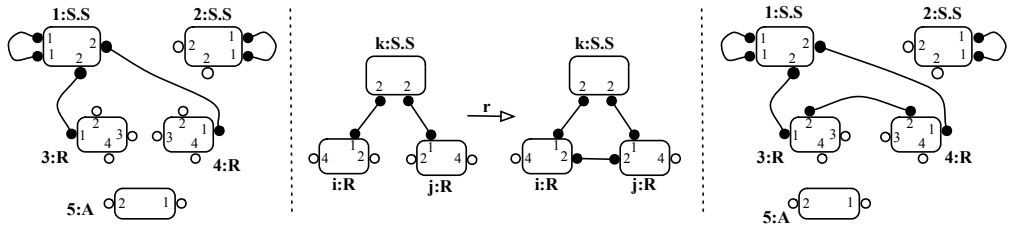


Fig. 0.1: Deux graphes moléculaires reliés pas une réaction de complexation

Comme on l'a déjà vu dans l'exemple ci-dessus, la modélisation des molécules en employant la structure des graphes avec ports les dote de capacités de raccordement. Ceci nous motive à instancier la structure abstraite Σ dans le ρ_Σ-calcul avec les graphes avec ports. En conséquence, nous obtenons un calcul biochimique basé sur la réécriture stratégique de graphes avec ports, le ρ_{pg}-calcul. Les graphes avec ports représentent une structure unificatrice pour représenter toutes sortes de molécules abstraites dans le ρ_{pg}-calcul. En outre, les opérations sous-jacentes au mécanisme d'application, le filtrage et le remplacement, habituellement définis à l'extérieur du calcul de réécriture, sont exprimables en utilisant des noeuds appropriés et des transformations de graphes avec ports. En limitant les graphes avec ports aux graphes moléculaires, nous obtenons un calcul pour modéliser les réseaux biochimiques [AK08a].

Etant donné que le γ-calcul et le HOCL se sont avérés bien adaptés à la modélisation des systèmes autonomes [Rad07], nous avons également enquêté sur les propriétés du ρ_{pg}-calcul pour cette modélisation [AK08d, AK08b]. En particulier, l'usage de stratégies ayant le même statut que les objets (molécules) dans le calcul, aide le système à s'auto-organiser et à coordonner les comportements de ses composants. Cette étude est également pertinente pour la modélisation de systèmes biologiques en raison de leur comportement très complexe et autonome. Ainsi nous utilisons le ρ_{pg}-calcul pour la modélisation d'un fragment de la voie de signalisation de l'EGFR. Toujours dans le cadre de la modélisation des systèmes autonomes, nous analysons la possibilité d'intégrer des éléments de vérification dans le calcul sur la base de ses capacités d'ordre supérieur.

Au-delà de la Simulation : Ajouter de la Vérification à l'Exécution dans le Calcul Biochimique

Dans le cadre de la modélisation des systèmes autonomes, la vérification à l'exécution est utile pour rattraper des situations problématiques, on parle en ce cas de propriété

d'auto-réparation. Les conditions typiques qu'on veut qu'un système satisfasse, sont l'occurrence, la conséquence et l'invariance de propriétés structurales ou comportementales particulières. De tels types de conditions sont également intéressantes pour vérifier des modèles biochimiques [CRCFS04, MRM+08].

Grâce à la possibilité d'encoder les stratégies comme objets du calcul et à la construction de multiverses qui comprennent toutes les interactions possibles entre une abstraction et une molécule, nous dotons le ρ_{pg}-calcul d'une méthode automatisée pour valider le comportement du système en ce qui concerne quelques conditions ou propriétés de conception initiales. Nous exprimons les conditions sous forme de formules de logique temporelle standard – logique des arbres computationnels (Computational Tree Logic, CTL) [CGP00] – , celle-ci étant bien adaptée pour raisonner sur la réduction de graphes avec ports. Les propositions atomiques sont des formules structurelles basées sur les expressions de graphes avec ports que nous encodons avec des stratégies. Dans ce contexte, nous vérifions que le système modélisé satisfait une proposition atomique utilisant le mécanisme d'évaluation des stratégies de réécriture. Nous avons mis les formules temporelles au même niveau que la description du système dans le ρ_{pg}-calcul et nous obtenons une technique de vérification à l'exécution qui permet au système modélisé de détecter ses propres échecs. En outre, le système modélisé peut être équipé de stratégies de récupération pour manipuler l'échec des conditions initiales.

En conclusion, nous proposons un formalisme biochimique d'ordre supérieur basé sur la réécriture stratégique de structures spécifiques, qui est conçu non seulement pour simuler l'évolution d'un système dans le temps, mais également pour vérifier la structure et l'évolution du système par rapport à des propriétés données.

Plan de la Thèse

La thèse est organisée comme suit :

Partie I En Chapitre 1 nous passons en revue les notions et les concepts de base sur la réécriture et les stratégies que nous employons dans la thèse. Du Chapitre 2 au Chapitre 9 nous proposons un calcul abstrait appelé le $\rho_{\langle\Sigma\rangle}$-calcul, où Σ dénote la structure des molécules. Nous présentons sa syntaxe et sa sémantique par étapes, à partir de l'intuition de base, rendant explicite l'application d'une abstraction à une molécule. Nous définissons alors les stratégies comme des abstractions dans le calcul.

Partie II Nous étudions une instance du calcul abstrait biochimique pour une structure particulière de graphe.

Chapitre 10 Nous définissons la structure de graphes avec ports, puis présentons un algorithme de filtrage pour les graphes avec ports, des règles de réécriture de graphes avec ports et une relation de réécriture sur les graphes avec ports. Nous étudions également la propriété de confluence pour la réécriture de graphes avec ports.

Chapitre 11 A partir de la structure des graphes avec ports, nous instancions le $\rho_{\langle\Sigma\rangle}$-calcul pour obtenir un calcul biochimique basé sur la réécriture stratégique de graphes avec ports.

Partie III Nous étudions des applications et des extensions pour le ρ_{pg}-calcul.

Chapitre 12 Nous illustrons la pertinence du ρ_{pg}-calcul pour modéliser des systèmes autonomes grâce aux stratégies encodées sous forme de molécules dans le calcul. Nous instancions également le ρ_{pg}-calcul par la structure de graphes moléculaires pour modéliser un fragment de la voie de signalisation du récepteur au facteur de croissance de l'épiderme (EGFR). Enfin, nous donnons les idées principales de l'implantation en TOM, qui est détaillée dans l'Annexe A.

Chapitre 13 Nous donnons une sémantique opérationnelle pour la réécriture de graphes avec ports, basée sur des termes algébriques définis sur une signature avec sortes ordonnées appropriée. Cet encodage des graphes avec ports et leur relation de réécriture nous permettent d'instancier le ρ-calcul pour obtenir un calcul de réécriture pour des termes encodant les graphes avec ports.

Chapitre 14 Nous étendons la syntaxe du calcul au moyen d'une classe de formules temporelles et sa sémantique pour vérifier la satisfaction de ces formules. Nous obtenons de cette façon un calcul biochimique avec des possibilités de vérification à l'exécution.

Pour finir, nous donnons une conclusion générale, ainsi que des perspectives pour des travaux futurs.

La Partie I, la Partie II ainsi que la conclusion de la thèse ont été traduits en français. La Partie III provient de la thèse en anglais. Une version complète en anglais de cette thèse est disponible sur le serveur de thèses-en-ligne du Centre pour la Communication Scientifique Directe – CNRS, via le lien suivant :
http://tel.archives-ouvertes.fr/tel-00337558/en/

Première partie

Un Calcul Biochimique Abstrait

1 Notions Préliminaires

Dans cette partie nous présentons les notions utiles concernant la réécriture de termes, la réécriture de graphes et la réécriture sous stratégies.

1.1 Relations binaires et leurs propriétés

Dans ce qui suit nous passons en revue des définitions de base et des notations, ainsi que des propriétés habituelles des relations binaires [BN98].

Définition 1 (Relations binaires). Étant données deux relations binaires $R \subseteq A \times B$ et $S \subseteq B \times C$, leur composition est définie par

$$R \circ S = \{(a, c) \mid \exists b \in B.(a, b) \in R \wedge (b, c) \in S\}$$

Soit $\rightarrow$ une relation binaire définie sur l'ensemble A. On dénote par :
- $\rightarrow^0$ l'identité sur A,
- $\rightarrow^n$ la n-ième composition de $\rightarrow$, $\rightarrow^n = \rightarrow \circ \rightarrow^{n-1}$, pour $n > 0$,
- $\rightarrow^=$ la clôture réflexive de $\rightarrow$, $\rightarrow^= = \rightarrow \cup \rightarrow^0$,
- $\leftarrow$ l'inverse de $\rightarrow$, $\leftarrow = \{(y, x) \mid x \rightarrow y\}$,
- $\leftrightarrow$ la clôture symétrique de $\rightarrow$, $\leftrightarrow = \rightarrow \cup \leftarrow$
- $\rightarrow^+$ la clôture transitive de $\rightarrow$, $\rightarrow^+ = \cup_{n>0} \rightarrow^n$,
- $\rightarrow^*$ la clôture réflexive et transitive de $\rightarrow$, $\rightarrow^* = \rightarrow^0 \cup \rightarrow^+$,
- $\leftrightarrow^*$ la clôture réflexive, transitive et symétrique de $\rightarrow$.

Définition 2 (Réductibilité). Soit $\rightarrow$ une relation sur A. Un élément x dans A est réductible s'il y a un élément y dans A tel que $x \rightarrow y$; x est irréductible (ou sous forme normale) s'il n'est pas réductible. Une forme normale de x est un élément irréductible y tel que $x \rightarrow^* y$. Deux éléments x et y dans A sont joignables s'il existe z dans A tel que $x \rightarrow^* z$ et $y \rightarrow^* z$, et on le dénote par $x \downarrow y$.

Définition 3 (Propriétés des relations binaires). Soit $\rightarrow$ une relation sur A. La relation $\rightarrow$ est appelée :
- localement confluente si $x \rightarrow y_1$ et $x \rightarrow y_2$ implique $y_1 \downarrow y_2$;
- confluente if $x \rightarrow^* y_1$ et $x \rightarrow^* y_2$ implique $y_1 \downarrow y_2$;
- fortement normalisante (or elle termine) si il n'existe pas une séquence infinie $x_0 \rightarrow x_1 \rightarrow \ldots$;
- normalisante si chaque élément de A a une forme normale ;
- convergente si elle est confluente et termine.

La preuve de la confluence d'une relation est en général difficile. Mais si la relation se termine, il suffit de prouver que la relation est localement confluente.

Théorème 1 (Lemme de Newman [New42]). Une relation fortement normalisante est confluente si elle est localement confluente.

1.2 Graphes

Définition 4 (Graphes étiquetés). Un alphabet d'étiquettes $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ est une paire d'ensembles d'étiquettes de noeuds et d'étiquettes d'arêtes. Un graphe fini sous $\mathcal{L}$ est un tuple $G = (V, E, s^G, t^G, l^G)$ ou :

- V est un ensemble $\{v_1, \ldots, v_k\}$ d'éléments appelés noeuds ou sommets,
- E est un ensemble d'éléments du produit cartésien $V \times V$ appelés arêtes,
- $s^G, t^G : E \to V$ sont les fonctions qui donne la source et la cible d'une arête respectivement, et
- $l^G = (l_V^G, l_E^G)$ est la fonction d'étiquetage pour les noeuds ($l_V^G : V \to \mathcal{L}_V$) et arêtes ($l_E^G : E \to \mathcal{L}_E$).

Si G est un graphe, on dénote généralement par V_G son ensemble de nœuds et par E_G son ensemble d'arêtes.

Une arête de la forme (v, v) s'appelle boucle. Pour une arête (u, v), u et v s'appellent des nœuds de fin avec u source et v cible ; d'ailleurs nous disons que u et v sont des nœuds adjacents, avec v étant le voisin de u. Une arête est incidente à un nœud si le nœud est l'un de ses nœuds de fin. Une arête est multiple s'il y a une autre arête avec la même source et la même cible ; autrement elle est simple. Un multigraphe est un graphe où les arêtes multiples et les boucles sont permises, c.-à-d., E est un multi-ensemble des paires en $V \times V$. Un chemin est une séquence de nœuds $\{v_1, \ldots, v_n\}$ tels que (v_1, v_2), …, (v_{n-1}, v_n) sont des arêtes du graphes.

Une liste d'adjacence pour un nœud est donné par une liste de paires se composant d'un voisin et de l'étiquette correspondante de l'arête. Si un nœud n'a aucun voisin, sa liste d'adjacence est vide.

Un sous-graphe d'un graphe G est un graphe dont les ensembles de nœuds et d'arêtes sont des sous-ensembles de ceux de G. Un sous-graphe H d'un graphes G est induit si, pour n'importe quelles paires de sommets v et u de H, (v, u) est une arête de H si et seulement si (v, u) est une arête de G. Autrement dit, H est un sous-graphe induit de G s'il a tous les arêtes qui apparaissent dans G au-dessus du même ensemble de sommets.

Un morphisme de graphes $f : G \to H$ est une paire de fonctions $f_V : V_G \to V_H$ et $f_E : E_G \to E_H$ qui préserve les sources, les cibles et les étiquettes tout en préservant l'adjacence, i.e., qui satisfait $f_V \circ t^G = t^H \circ f_E$, $f_V \circ s^G = s^H \circ f_E$, $l_V^H \circ f_V = l_V^G$, $l_E^H \circ f_E = l_E^G$.

Un morphisme partiel de graphes $f : G \to H$ est un morphisme total de graphes d'un sous-graphe $dom(f)$ de G vers H, avec $dom(f)$ appelé le domaine de f.

La composition de deux morphismes (partiels) de graphes est définie par la composition des composants, et les identités comme paires d'identités sur les composantes.

1.3 Systèmes de Réduction Abstraits

Habituellement un système de réduction abstrait est décrit par un ensemble et une relation binaire définie sur cet ensemble. Afin de raisonner plus tard sur la notion des stratégies, dans cette thèse nous adoptons les définitions générales de [KKK08] basées sur la notion de graphe. Ces définitions nous permettent de décrire les différentes manières possibles pour un objet d'être atteint à partir d'un autre objet par réduction.

Définition 5 (Système de réduction abstrait). Un système de réduction abstrait (ARS) est un graphe étiqueté orienté $(\mathcal{O}, \mathcal{S})$. Les nœuds de $\mathcal{O}$ sont appelés des objets et les arêtes orientées de $\mathcal{S}$ sont appelées des pas.

Définition 6 (Dérivation). Pour un ARS donné $\mathcal{A}$:

1. Un pas de réduction est une arête étiquetée ϕ ensemble avec sa source a et sa cible b. On l'écrit $a \to_{\mathcal{A}}^{\phi} b$, ou simplement $a \to^{\phi} b$ s'il n'y a pas d'ambiguïté.
2. Une $\mathcal{A}$-dérivation ou $\mathcal{A}$-séquence de réduction est un chemin π dans le graphe $\mathcal{A}$.
3. Quand le chemin π est fini, il peut être écrit $a_0 \to^{\phi_0} a_1 \to^{\phi_1} a_2 \ldots \to^{\phi_{n-1}} a_n$ et on dit que a_0 se réduit en a_n par la dérivation $\pi = \phi_0\phi_1 \ldots \phi_{n-1}$; on le dénote aussi par $a_0 \to^{\pi} a_n$. La source de π est le singleton $\{a_0\}$ dénoté par $\mathrm{dom}(\pi)$. La cible de π est le singleton $\{a_n\}$ dénoté par $[\pi](a_0)$.
4. Une dérivation est vide si sa longueur est zéro. La dérivation vide avec la source a est dénoté par id_a.
5. La concaténation de deux dérivations $\pi_1; \pi_2$ est définie quand π_1 est fini et $\mathrm{dom}(\pi_2) = [\pi_1](\mathrm{dom}(\pi_1))$ comme suit :

$$\pi_1; \pi_2 : \mathrm{dom}(\pi_1) \to_{\mathcal{A}}^{\pi_1} \mathrm{dom}(\pi_2) \to_{\mathcal{A}}^{\pi_2} [\pi_2]([\pi_1](\mathrm{dom}(\pi_1)))$$

On note qu'une $\mathcal{A}$-dérivation est la concaténation de ses pas de réduction. La concaténation de π_1 et π_2, si elle est possible, est une nouvelle $\mathcal{A}$-dérivation.

Les définitions qui suivent généralisent les propriétés classiques d'une relation binaire à un système de réduction abstrait.

Définition 7 (Terminaison). Pour $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ un système de réduction abstrait, on dit que :

- $\mathcal{A}$ termine (ou est fortement normalisant) si toutes les dérivations ont une longueur finie ;
- un objet a en $\mathcal{O}$ est normalisé quand la dérivation vide est la seule dérivation ayant la source a ;
- une dérivation est normalisante si sa cible est normalisée ;
- un système de réduction abstrait est faiblement normalisant si chaque objet a est la source d'une dérivation normalisante.

Définition 8 (Confluence). Un système de réduction abstrait $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ est confluent si pour tous les objets a, b, c dans $\mathcal{O}$, et pour toutes les $\mathcal{A}$-dérivations π_1 et π_2, quand $a \to^{\pi_1} b$ et $a \to^{\pi_2} c$, il existe d dans $\mathcal{O}$ et deux $\mathcal{A}$-dérivations π_3, π_4 tels que $c \to^{\pi_3} d$ et $b \to^{\pi_4} d$.

1.4 Réécriture de Termes

Cette section contient les notions de base sur l'algèbre de termes de premier ordre et la réécriture de termes [BN98, GM92].

1.4.1 Algèbre de termes

Une signature multi-sortée est une paire $(\mathcal{S}, \mathcal{F})$ où $\mathcal{S}$ est un ensemble de sortes et $\mathcal{F}$ un ensemble de symboles de fonctions sortés, $\mathcal{F} = \{\mathcal{F}_{S_1 \dots S_n, S} \mid S_1, \dots S_n, S \in \mathcal{S}\}$. Pour $f \in \mathcal{F}_{S_1 \dots S_n, S}$ on utilise la notation $f : S_1 \ \dots \ S_n \to S$. Une signature ordo-sortée est un triplet $(\mathcal{S}, \leq, \mathcal{F})$ tel que $(\mathcal{S}, \mathcal{F})$ est une signature multi-sortée et $(\mathcal{S}, \leq)$ est un ensemble partiellement ordonné, et les symboles de fonction satisfont la condition de monotonicité suivante : si $f \in \mathcal{F}_{S_1 \dots S_n, S} \cap \mathcal{F}_{S'_1 \dots S'_n, S'}$ et $S_i \leq S'_i$ pour tous i, $1 \leq i \leq n$, alors $S \leq S'$. Dans ce qui suit, pour représenter la réécriture de termes, on considère seulement des signatures multi-sortées ; une introduction complète sur les algèbres ordonées-sortées est faite dans [GM92].

Quand $f \in \mathcal{F}_{S_1 \dots S_n, S}$, on dit que f a le rang $\langle S_1 \dots S_n, S \rangle$, l'arité $S_1 \dots S_n$, et la sorte S. Si $n = 0$, alors f est appelé constante. Si l'arité de f est d'une longueur variable, alors f est variadique. En général, quand $\mathcal{S}$ est un singleton, l'arité d'un symbole de fonction est réduite à un entier naturel.

Soit $(\mathcal{S}, \mathcal{F})$ une signature multi-sortée et $\mathcal{X} = \{\mathcal{X}_S\}_{S \in \mathcal{S}}$ une famille $\mathcal{S}$-sortée d'ensembles disjoints de variables.

Définition 9. L'ensemble des termes de sorte S sur la signature $(\mathcal{S}, \mathcal{F})$ et l'ensemble de variables $\mathcal{X}$, dénoté $\mathcal{T}(\mathcal{F}, \mathcal{X})_S$, est le plus petit ensemble qui contient $\mathcal{X}_S$ tel que $f(t_1, \dots, t_n)$ est dans $\mathcal{T}(\mathcal{F}, \mathcal{X})_S$ chaque fois que $f : S_1 \dots S_n \to S$ et $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})_{S_i}$ pour $1 \leq i \leq n$, $n \geq 0$. Alors $\mathcal{T}(\mathcal{F}, \mathcal{X}) = \bigcup \mathcal{T}(\mathcal{F}, \mathcal{X})_{S \in \mathcal{S}}$ est l'algèbre de termes généré par la signature $(\mathcal{S}, \mathcal{F})$ et l'ensemble de variables $\mathcal{X}$.

Le symbole top d'un terme est dénoté $\mathrm{Head}(t)$. L'ensemble de variables apparaissant dans un terme t est dénoté par $\mathcal{V}ar(t)$. Si $\mathcal{V}ar(t)$ est vide, t est appelé un terme clos. $\mathcal{T}(\mathcal{F})$ est l'ensemble de tous les termes clos. On omet les noms de sortes quand ils sont donnés par le contexte. Un terme $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ est linéaire si chaque variable dans t apparaît un seule fois.

Soient $\mathbb{N}$ l'ensemble des entiers naturels et $\mathbb{N}_+$ l'ensemble de entiers naturels distincts de zéro. L'ensemble des séquences finies d'entiers naturels distincts de zéro $\mathbb{N}_+^*$ est défini par $p = \epsilon \mid n \mid p.p$, où ϵ représente une séquence vide et $n \in \mathbb{N}_+$. Pour tous $p, q \in \mathbb{N}_+^*$, p est un préfixe de q s'il existe $r \in \mathbb{N}_+^*$ tel que $q = p.r$.

L'ensemble des positions $\mathcal{P}os(t)$ d'un terme t est défini récursivement comme suit :

- $\epsilon \in \mathcal{P}os(t)$ est la position top de t.
- Pour tous $p \in \mathcal{P}os(t)$ et tous $i \in \mathbb{N}_+^*$, $p.i \in \mathcal{P}os(t)$ si et seulement si $1 \leq i \leq |arity(f)|$ où $f \in \mathcal{F}$ est le symbole situé à la position p de t.

On appelle un sous-terme de t situé à la position $p \in \mathcal{P}os(t)$ le terme dénoté par $t_{|p}$ qui satisfait la condition suivante :

$$\forall p.r \in \mathcal{P}os(t), r \in \mathcal{P}os(t_{|p}) \text{ et } \mathrm{Head}(t_{|p.r}) = \mathrm{Head}((t_{|p})_{|r})$$

On dénote par $t[s]_p$ le terme t où le sous-terme à la position p a été remplacé par le terme s.

Une substitution σ est une association de chaque variable d'un sous-ensemble fini $\{x_1, \ldots, x_k\}$ de $\mathcal{X}$ vers un terme de même sorte dans $\mathcal{T}(\mathcal{F}, \mathcal{X})$, et on l'écrit $\sigma = \{x_1 \mapsto t_1, \ldots, x_k \mapsto t_k\}$. On défini le domaine de σ comme $dom(\sigma) = \{x_1, \ldots, x_k\}$. L'application d'une substitution σ à un terme t, dénoté par $\sigma(t)$, remplace de manière simultané toutes les apparitions des variables par leurs σ-images correspondantes. La composition de deux substitutions σ et μ est dénoté par $\sigma\mu$ et $(\sigma\mu)(t) = \sigma(\mu(t))$ pour tout terme t. On dit que σ instancie x si $x \in dom(\sigma)$.

Une substitution σ est plus générale qu'une substitution σ' s'il existe une substitution δ telle que $\sigma' = \delta\sigma$. Dans ce cas on écrit $\sigma \lesssim \sigma'$. On dit que σ' est une instance de σ.

Deux termes sont unifiables si il existe une substitution σ telle que $\sigma(s) = \sigma(t)$. Puis σ est le plus général unificateur pour s et t si pour n'importe quel autre unificateur σ' de s et t, $\sigma \lesssim \sigma'$.

Définition 10 (Filtrage). On dit qu'un terme t filtre un terme t', ou que t' est une instance de t, si il existe une substitution σ telle que $t' = \sigma(t)$.

Généralement on appelle t le motif et t' le sujet du filtrage. Ce type de filtrage est connu comme filtrage syntaxique. Le filtrage syntaxique est décidable. Il est linéaire dans la dimension du motif, si le motif est linéaire. Sinon, le filtrage est linéaire dans la dimension du sujet.

1.4.2 Théories Équationelles

Une égalité ou axiome dans une algèbre de termes $\mathcal{T}(\mathcal{F}, \mathcal{X})$ est une paire de termes $\langle l, r \rangle$, dénotée par $l = r$, où l et r sont des termes de même sorte. Pour E un ensemble d'axiomes, on dénote par $\longleftrightarrow_E$ la relation binaire symétrique sur $\mathcal{T}(\mathcal{F}, \mathcal{X})$ définie par $s \longleftrightarrow_E t$ si il y a un axiome $l = r$ dans E, une position p dans s et une substitution σ tel que $s|_p = \sigma(l)$ et $t = s[\sigma(r)]_p$. La clôture réflexive et transitive de $\longleftrightarrow_E$, dénotée par $\overset{*}{\longleftrightarrow}_E$, est la théorie équationelle généré par E, ou, en plus court, la théorie équationelle E.

Certaines des théories qu'on utilise dans cette thèse sont définies dans ce qui suit pour f un symbole de fonction binaire :

(A)	Associativité	$f(f(x, y), z) = f(x, f(y, z))$
(C)	Commutativité	$f(x, y) = f(y, z)$
(I)	Idempotence	$f(x, x) = x$
(U_e)	Unité	$f(x, e) = f(e, x) = x$

On peut combiner ces théories pour obtenir par exemple des théories associatives avec unité (AU), associatives et commutatives (AC), ou associatives et commutatives avec unité (ACU). De plus, une théorie équationelle E est une théorie permutative si pour chaque équation $s \longleftrightarrow_E t$, le nombre d'apparitions de chaque symbole dans s est le même que dans t.

Le problème de décider si deux termes arbitraires sont égaux dans une théorie équationelle est connu comme le problème des mots.

La notion de filtrage peut être généralisé pour prendre en compte le fait que des termes peuvent être égaux modulo une théorie équationelle donnée. On dit qu'un terme t filtre modulo E un terme s si il existe une substitution σ telle que $s \xleftrightarrow{*}_E \sigma(t)$.

Contrairement au problème de filtrage syntaxique, le problème de filtrage modulo une théorie équationelle est indécidable en général [BS01]. Quand il existe des algorithmes, ils peuvent avoir une complexité assez grande.

1.4.3 Réécriture de Termes

Soient $(\mathcal{S}, \mathcal{F})$ et $\mathcal{X}$ une signature multi-sortée et un ensemble de variable tels que définis précédemment.

Définition 11 (Règle de réécriture). Une règle de réécriture pour l'algèbre de termes $\mathcal{T}(\mathcal{F}, \mathcal{X})$ est une paire de termes orientée, dénotée $l \to r$, où l et r sont des termes de $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Nous appelons l et r respectivement membre de gauche et de droite d'une règle.

Un système de réécriture de termes est un ensemble R de règles de réécriture pour $\mathcal{T}(\mathcal{F}, \mathcal{X})$.

Parfois, nous ajoutons des étiquettes aux règles pour les identifier. Une règle de réécriture étiquetée a la forme $id \ : \ l \to r$.

Quelques restrictions sont généralement imposées sur une règle de réécriture $l \to r$:

- $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ (l'ensemble des variables du membre de droite est un sous-ensemble de l'ensemble des variables du membre de gauche),
- $l \notin \mathcal{X}$ (le membre de gauche n'est pas une variable),
- l et r sont de la même sorte.

Définition 12 (Relation de réécriture). Soit R un système de réécriture sur $\mathcal{T}(\mathcal{F}, \mathcal{X})$. La relation de réécriture associée à R sur $\mathcal{T}(\mathcal{F}, \mathcal{X})$ est dénotée $\to_R$ et est définie comme suit : $t \to_R s$ si il existe une position p dans t, une règle de réécriture $l \to r$ dans R et une substitution σ telle que $t|_p = \sigma(l)$ et $s = t[\sigma(r)]_p$. Le sous-terme $t|_p$ est une instance du membre de gauche l et est appelé un redex.

Les propriétés d'un système de réécriture de termes R sont celles d'une relation $\to_R$. Tous ces propriétés, en particulier la terminaison et la confluence sont indécidables en général. Ce n'est pas surprenant car la réécriture de termes est au moins aussi expressive que les machines de Turing. En effet, les machines de Turing peuvent être exprimées sous forme d'une simple règle de réécriture [Dau92].

Cependant, il existe des méthodes pour rendre ces propriétés décidables pour des classes spécifiques de systèmes de réécriture de termes. Par exemple, la terminaison d'un système de réécriture de termes peut être prouvée en utilisant un ordre de simplification approprié, grâce au théorème ci-dessous. Un ordre de réécriture est un ordre compatible sur un ensemble de termes. Un ordre de simplification est un ordre de réécriture qui contient la relation de sous-terme strict.

Théorème 2. [Der82] Soit $\mathcal{F}$ une signature avec un ensemble fini de symboles. Un système de réécriture de termes R sur $\mathcal{T}(\mathcal{F},\mathcal{X})$ termine s'il existe un ordre de simplification $\succ$ tel que $l \succ r$ pour chaque règle $l \to r \in R$.

La confluence peut être décidable pour les systèmes de réécriture de termes terminants en appliquant le lemme de Newman qui assure que la confluence locale implique la confluence pour de tels systèmes. La confluence locale peut être décidée en testant la joignabilité de paires critiques [BN98].

Définition 13 (Paire critique). Soient $l \to r$ et $g \to d$ deux règles avec des ensembles de variables disjoints. Nous appelons paire critique dans la règle $g \to d$ sur $l \to r$ à la position non variable $p \in \mathcal{P}os(l)$, la paire $(\sigma(r), \sigma(l)[\sigma(d)]_p)$ telle que σ est un plus général unificateur de g et $l|_p$.

Si chaque paire critique est joignable, le système de réécriture de termes est confluent localement. Puisque le nombre de paires critiques dans un système de réécriture de termes fini est également finie, la confluence locale est décidable.

Les systèmes de réécriture conditionnels apparaissent naturellement dans certaines des spécifications adoptées dans cette thèse.

Définition 14 (Réécriture conditionnelle). Un système de réécriture de termes conditionnel est un ensemble de règles de réécriture conditionnelles R sur un ensemble de termes $\mathcal{T}(\mathcal{F},\mathcal{X})$. Chaque règle de réécriture est de la forme $l \to r$ if $s_1 \to t_1, \ldots, s_k \to t_k$ avec $l, r, s_1, \ldots, s_k, t_1, \ldots t_k \in \mathcal{T}(\mathcal{F},\mathcal{X})$.

- Pour chaque règle dans R système de réécriture de termes $\mathcal{V}ar(r) \cup \mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$, où c est une abréviation pour la partie conditionnelle de la règle, $s_1 \to t_1, \ldots, s_k \to t_k$.
- Chaque t_j dans c est une forme normale fondée par rapport à R_u, qui contient toutes les règles dans R sans leurs parties conditionnelles.

Définition 15. Étant donné un système de réécriture conditionnel R, un terme t se réécrit dans un terme t', qui est dénoté comme d'habitude $t \to_R t'$ si il existe une règle de réécriture conditionnelle $l \to r$ if c, une position ω dans t, et une substitution σ satisfaisant $t_{|\omega} = \sigma(l)$, et $\sigma(s_1) \to_{R_u} t_1, \ldots, \sigma(s_k) \to_{R_u} t_k$.

Nous introduisons à présent la notion de réécriture modulo un ensemble d'équations. Quand les axiomes d'une théorie équationnelle peuvent être orientés dans un système de réécriture de termes canonique, les règles de réécriture peuvent être utilisées pour résoudre le problème du mot dans une telle théorie. Cependant, il y a des égalités qui ne peuvent pas être orientées sans perdre la propriété de terminaison. Un exemple typique est l'axiome de commutativité. Dans ce cas, le raisonnement équationnel a besoin d'une relation de réécriture différente qui fonctionne sur des classes d'équivalence de termes modulo ces égalités non orientables.

Définition 16 (Réécriture modulo classes d'équivalence). Soient un système de réécriture de termes R et un ensemble d'axiomes $\mathcal{E}$, le terme t se réécrit dans le terme s par R modulo $\mathcal{E}$, ce qui est dénoté $t \longrightarrow_{R/\mathcal{E}} s$, s'il existe une règle $l \to r \in R$, un terme u, une position p dans u et une substitution σ, tels que $t \overset{*}{\longleftrightarrow}_{\mathcal{E}} u[\sigma(l)]_p$ et $s \overset{*}{\longleftrightarrow}_{\mathcal{E}} u[\sigma(r)]_p$.

La relation $\longrightarrow_{R/\mathcal{E}}$ n'est pas satisfaisante en termes d'efficacité car pour réécrire un terme, il est nécessaire de cherche dans toute la classe d'équivalence $\mathcal{E}$. Une telle recherche est encore plus difficile dans le cas de classes d'équivalence infinies. Afin de résoudre ce problème, une relation plus faible a été proposée par [PS81], et généralisée par [JK86], dans laquelle le filtrage est remplacé par le filtrage modulo une théorie équationnelle. Cette relation est appelée réécriture modulo une théorie équationnelle et est dénotée $\rightarrow_{R,\mathcal{E}}$.

En pratique, la théorie équationnelle la plus utilisée est l'associativité et la commutativité. La relation $\rightarrow_{R,\mathcal{E}}$ est appelée dans ce cas réécriture modulo associativité et commutativité (AC). L'efficacité du filtrage modulo AC est essentielle pour les performances de la réécriture modulo AC. Cependant, le filtrage modulo AC est connu comme étant un problème NP-difficile [BKN87] et peut avoir un nombre exponentiel de solutions.

1.5 Éléments de Théorie des Catégories

Nous passons en revue quelques éléments de la théorie des catégories [Mac98] utilisés dans cette thèse. Nous rappelons les définitions de catégorie, foncteur, pushout, et catégorie strictement monoïdale et strictement symétrique.

Définition 17 (Catégorie). Une catégorie C est donnée par :

- Une classe d'objets dénotée par $Obj(\mathrm{C})$.
- Une classe de morphismes (ou flèches) dénotée par $Arr(\mathrm{C})$, où chaque morphisme f a un unique objet source A et objet cible B, avec A et B des objets de C. Nous dénotons par $\mathrm{C}(A, B)$ la classe de tous les morphismes de l'objet A vers l'objet B.
- Une loi de composition $\circ : \mathrm{C}(A, B) \times \mathrm{C}(B, C) \rightarrow \mathrm{C}(A, C)$ qui est associative, c'est-à-dire

$$\text{si } f \in \mathrm{C}(A, B), g \in \mathrm{C}(B, C), h \in \mathrm{C}(C, D) \text{ alors } h \circ (g \circ f) = (h \circ g) \circ f.$$

- Un morphisme identité $\mathrm{id}_A \in \mathrm{C}(A, A)$ pour tous les objets A, qui est un élément neutre pour $\circ$, c'est-à-dire

$$\forall f \in \mathrm{C}(A, B) \qquad f \circ id_A = f = id_B \circ f.$$

Un foncteur est un morphisme de catégories.

Définition 18 (Foncteur). Un foncteur F d'une catégorie C vers une catégorie D, noté $F : \mathrm{C} \rightarrow \mathrm{D}$, consiste en deux fonctions :

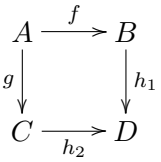
- la fonction objet qui assigne à chaque objet A dans C un objet $F(A)$ dans D, et
- la fonction flèche qui assigne à chaque flèche $f : A \rightarrow B$ de C une flèche $F(f) : F(A) \rightarrow F(B)$ dans D,

tels que

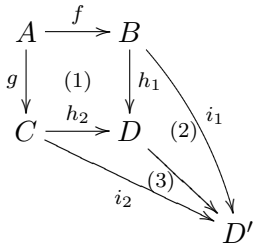
$$F(id_A) = id_{F(A)}, \qquad F(g \circ f) = F(g) \circ F(f)$$

Définition 19 (Pushout). Étant donné une catégorie C, une paire de flèches $f : A \rightarrow B$ et $g : A \rightarrow C$, un pushout de f et g consiste en un objet D et deux flèches $h_1 : C \rightarrow D$ et $h_1 : B \rightarrow D$ pour lesquelles les deux conditions suivantes sont satisfaites :

(commutativité) Le diagramme ci-dessous commute :



(universalité) Pour tout objet D' et flèches $i_1 : B \to D'$ et $i_2 : C \to D'$ telles que $i_1 \circ f = i_2 \circ g$, il existe un morphisme unique $D \to D'$ tel que les diagrammes (2) et (3) ci-dessous commutent.



Définition 20 (Catégorie strictement monoïdale et strictement symétrique). Une catégorie strictement monoïdale et strictement symétrique (Strict symmetric strict monoidal category, sssmc) $(B, +, e)$ est une catégorie B équipée d'un bifoncteur $+ : B \times B \to B$ appelé le produit tensoriel ou somme disjointe, et un objet e appelé l'objet unit ou objet identité qui satisfait :

1. la classe des objets $Ob(B)$ est un monoïde commutatif par rapport à $+$ avec l'élément neutre e, (i.e., $+$ est associatif, commutatif, avec un élément neutre),
2. pour chaque $f \in B(a, b)$ et $g \in B(a', b')$ la somme $f + g$ est dans $B(a + a', b + b')$ et satisfait les axiomes suivants :
   (C1) $(f + g) + h = f + (g + h)$,
   (C2) $\mathrm{id}_e + f = f = f + \mathrm{id}_e$,
   (C3) $\mathrm{id}_a + \mathrm{id}_b = \mathrm{id}_{a+b}$,
   (C4) $(u + v) \circ (f + g) = u \circ f + v \circ g$,
   (C5) $f + g = g + f$ pour tout $f, g \in \mathrm{Arr}(B)$.

Exemple 1. La catégorie ayant des graphes étiquetés comme objets et des morphismes de graphes comme flèches s'appelle Graphe. En limitant les flèches aux morphismes partiels, une nouvelle catégorie est obtenue, appelée $\mathrm{Graphe}^{\mathrm{P}}$.

1.6 Transformation de Graphe

Dans cette section, nous décrivons brièvement l'intuition se cachant derrière la transformation de graphe et l'approche algébrique, aussi appelée "approche du double pushout" (Double PushOut approach, DPO) [Roz97, EEKR97, EKMR97].

Une règle de transformation de graphe $L \rightsquigarrow R$ consiste en deux graphes L et R appelés membre de gauche et de droite respectivement, et un morphisme de graphe partiel entre eux. Le morphisme de graphe partiel fournit une correspondance entre les éléments du

membre de gauche et ceux du membre de droite. Graphiquement, cette correspondance est fournie par l'association d'identifiants uniques aux nœuds. Le sous-graphe K de L qui est associé à un sous-graphe de R est généralement mis en relief par la représentation de la règle de transformation de graphe sous forme de triplet $L \leftarrow K \rightarrow R$.

Définition 21 (Règle de transformation de graphe). Dans l'approche DPO, une production $p : L \leftarrow K \rightarrow R$ est une paire d'homomorphismes de graphe $l : K \rightarrow L$ et $r : K \rightarrow R$ où L, K, R sont des graphes finis et sont appelés le membre de gauche, l'interface et le membre de droite respectivement. Un système de transformation de graphe est un ensemble fini de règles de transformation de graphe.

Un dérivation directe de graphe peut être formalisée dans le cadre catégorique sous forme de pushout simple *SPO* [EHK+97] ou double pushout *DPO* [CMR+97]. Nous présentons ici l'approche du double pushout, qui a été la première approche proposée.

Définition 22 (Dérivation directe). Une dérivation directe de G vers H utilisant une production $p : L \leftarrow K \rightarrow R$ existe si et seulement si le diagramme ci-dessous peut être construit :

$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ \downarrow & & \downarrow & & \downarrow \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

où les deux carrés doivent être des pushouts dans la catégorie Graph. Dans ce cas, D est appelé le graphe de contexte.

Étant donné une production, le graphe K peut être vu comme une "interface", dans le sens ou il n'est pas affecté par l'étape de réécriture elle-même, mais il est nécessaire pour spécifier comment le membre de droite R est intégré au graphe D. Intuitivement, le graphe de contexte D est obtenu à partir du graphe G en effaçant tous les éléments de G qui ont une pré-image dans L mais pas dans K. Le second diagramme de pushout représente l'insertion dans H de tous les éléments de R qui n'ont pas de pré-image dans K.

En général, comme présenté dans [Sch97], l'application d'une règle de transformation de graphe $L \rightsquigarrow R$ à un graphe G, appelé graphe hôte, produit un nouveau graphe G' selon les étapes suivantes :

1. Trouve un morphisme filtrant m pour L dans G (ainsi $m(L)$ est un sous-graphe de G).
2. Retire le sous-graphe $m(L)$ de G résultant en le graphe de contexte G^-.
3. Ajoute $m(R)$ au graphe de contexte G^-.
4. Reconnecte $m(R)$ et G^-.

Le problème de trouver un motif pour une règle donnée est une instance du problème d'homomorphisme de sous-graphe, qui est connu comme étant NP-complet [Meh84]. En fait, il existe de nombreuses définitions de graphe et de transformations de graphe, qui diffèrent suivant les motivations et le contexte d'application. Il y a plusieurs algorithmes

disponibles pour le filtrage de motifs de graphe, nous pouvons mentionner certains de ces travaux [Ull76, LV02, Zün96, Val02]. Parmi les outils existants pour la transformation de graphe, nous pouvons mentionner AGG [ERT97], PROGRES [SWZ97], et Clean [PE93]. Plus d'information sur les domaines d'application et les outils pour les transformations de graphe peuvent être trouvées dans [EEKR97].

1.7 Réécriture Stratégique

La notion de stratégie utilisée dans cette thèse est fondamentale dans la réécriture. Nous donnons ici une présentation générale des idées principales en suivant l'approche basée sur les systèmes de réduction abstraits [KKK08].

Un système de réécriture de terme $\mathcal{R}$ ou système de transformation de graphe $\mathcal{G}$ génère un système de réduction abstrait dont les nœuds sont des termes ou des graphes respectivement, et dont les arcs orientés sont des étapes de réécriture ou dérivations directes. Alors, une dérivation dans $\mathcal{R}$ ou dans $\mathcal{G}$ est un chemin dans le graphe sous-jacent au système de réduction abstrait associé.

Définition 23 (Stratégie abstraite). Pour un ARS $\mathcal{A}$ donné :

1. Une stratégie abstraite ζ est un sous-ensemble de l'ensemble de toutes les dérivations (finies ou non) de $\mathcal{A}$.
2. Appliquer la stratégie ζ à un objet a, dénoté par $[\zeta](a)$, consiste en l'ensemble de tous les objets qui peuvent être atteints à partir de a en utilisant une dérivation dans ζ :
   $[\zeta](a) = \{b \mid \exists \pi \in \zeta \text{ tel que } a \twoheadrightarrow^{\pi} b\} = \bigcup_{\pi \in \zeta} \pi a$.
   Lorsqu'aucune dérivation dans ζ n'a pour source a, nous disons que l'application de stratégie sur a échoue.
3. Le domaine d'une stratégie est l'ensemble des objets qui sont source d'une dérivation dans ζ :
   $$\mathrm{dom}(\zeta) = \bigcup_{\delta \in \zeta} \mathrm{dom}(\delta)$$
4. La stratégie qui contient uniquement toutes les dérivations vides est $Id = \{id_a \mid a \in \mathcal{O}\}$.

Remarquons que, suivant la définition ci-dessus, une stratégie n'est pas définie pour tous les objets de l'ARS, ainsi c'est une fonction partielle. Une stratégie qui contient uniquement des dérivations infinies depuis une source $\{a\}$ s'applique à l'objet a et retourne l'ensemble vide. L'ensemble vide de dérivations est une stratégie appelée $Fail$; son application échoue systématiquement.

Il est maintenant possible de donner une définition de la réécriture stratégique :

Définition 24 (Réécriture stratégique). Étant donné un système de réduction abstrait $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ et une stratégie ζ, une dérivation de réécriture stratégique (ou dérivation de réécriture sous la stratégie ζ) est un élément de ζ.

Si le système de réduction abstrait $\mathcal{A}$ est généré par un système de réécriture de termes R, une étape de réécriture stratégique sous ζ est une étape de réécriture $t \to_\omega^R t'$ qui a lieu dans une dérivation de ζ.

Si $\mathcal{A}$ est généré par un système de transformation de graphe, une étape de réécriture de graphe stratégique ζ est une dérivation directe $G \to H$.

Une stratégie peut être décrite en enumérant tous ses éléments ou par un langage de stratégie. Différentes approches ont été suivies, produisant des langages de stratégie légèrement différents tels que ELAN[1] [KKV95, BKKR01b], Stratego[2] [Vis01], TOM[3] [BBK+07a, BBK+07c] ou Maude[4] [MOMV05]. Tous ces langages fournissent des langages de stratégie flexibles et expressifs ou des stratégies de haut-niveau sont définies en combinant des primitives de bas-niveau. Nous décrivons ci-dessous les éléments principaux du langage de stratégie TOM qui sont pertinents pour cette thèse.

En suivant [KKK08], nous pouvons distinguer deux classes de constructions dans le langage de stratégie : la première classe permet de construire des dérivations à partir d'éléments de base, à savoir les règles de réécriture. La seconde classe correspond aux constructions qui expriment le contrôle, en particulier le choix biaisé à gauche. De plus, la capacité d'exprimer la récursion dans le langage apporte encore plus d'expressivité.

Constructeur de stratégies élémentaires. Une stratégie élémentaire est soit l'Identité qui correspond à l'ensemble Id de toutes les dérivations vides, Fail qui dénote l'ensemble vide des dérivations $Fail$, ou un ensemble de règles de réécriture $\mathcal{R}$ qui représente dérivations simples (une étape) avec les règles de $\mathcal{R}$. $Sequence(\zeta_1, \zeta_2)$, aussi dénoté par $\zeta_1; \zeta_2$, est la concaténation de ζ_1 et ζ_2 chaque fois qu'elle existe : pour un objet a donné dans un ARS $\mathcal{A}$, $[\zeta_1; \zeta_2](a) = [\zeta_2]([\zeta_1](a))$.

Constructeurs de stratégies de contrôle. Quelques constructions sont nécessaires pour construire des dérivations, branchements et pour prendre en compte la structure des objets.

first $First(\zeta_1, \zeta_2)$ applique la première stratégie si elle n'échoue pas, sinon elle applique la seconde stratégie ; elle échoue si les deux stratégies échouent. Remarquons qu'il s'agit d'un choix déterministe, plus précisément, le choix biaisé à gauche.
$[First(\zeta_1, \zeta_2)](a) =$ si $[\zeta_1](a)$ n'échoue pas alors $[\zeta_1](a)$ sinon $[\zeta_2](a)$.

not $Not(\zeta)$ échoue si la stratégie ζ n'échoue pas, et elle ne fait rien si ζ échoue :
$[Not(\zeta)](a) =$ si $[\zeta](a)$ échoue alors Id sinon $Fail$.

if then else IfThenElse$(\zeta_1, \zeta_2, \zeta_3)$ applique la première stratégie : si elle n'échoue pas, elle applique la seconde stratégie, sinon elle applique la troisième stratégie, elle échoue si les deux stratégies ζ_2 et ζ_3 échouent.
$[\text{IfThenElse}(\zeta_1, \zeta_2, \zeta_3)](a) =$ si $[\zeta_1](a)$ n'échoue pas alors $[\zeta_2](a)$ sinon $[\zeta_3](a)$.

[1] http://elan.loria.fr
[2] http://www.program-transformation.org/Stratego
[3] http://tom.loria.fr
[4] http://maude.cs.uiuc.edu

fixpoint L'opérateur de récursion μ (comparable à *rec* dans OCaml) est introduit pour permettre la définition récursive de stratégies. $\mu x.\zeta$ applique la dérivation dans ζ avec la variable x instanciée à $\mu x.\zeta$, i.e., $\mu x.\zeta = \zeta[x \leftarrow \mu x.\zeta]$.

Les stratégies *Sequence* et $First$ s'étendent naturellement pour être applicables à une liste de stratégies.

Toutes ces stratégies sont alors composées pour construire d'autre stratégies utiles. Une stratégie composée est par exemple $Try(\zeta) = First(\zeta, Id)$ qui applique ζ si elle peut, et applique la stratégie Id sinon. De manière similaire, l'opération de combinaison *Repeat* est utilisée avec l'opérateur de point fixe pour itérer l'application d'une stratégie : $Repeat(\zeta) = \mu x.Try(\zeta; x)$.

Si les objets dans l'ARS ont une structure hiérarchique (ou sous forme de termes), alors les stratégies sont appliquées uniquement sur les positions hautes et les stratégies telles que bottom-up, top-down ou leftmost-innermost sont des caractéristiques d'ordre supérieur qui décrivent comment les règles de réécriture devraient être appliquées. Les constructeurs de contrôle de base pour de telles stratégies sur les termes sont les suivants :

tous les sous-termes $All(\zeta)$ dénote l'ensemble de toutes les dérivations dans ζ. Sur un terme t, $All(\zeta)$ applique la stratégie ζ sur tous les sous-termes immédiat : $[All(\zeta)](f(t_1, ..., t_n)) = f(t'_1, ..., t'_n)$ si $[\zeta](t_i) = \{t'_i\}$ pour tout i, $1 \leq i \leq n$, et échoue s'il existe i tel que $[\zeta](t_i)$ échoue.

un sous-terme $One(\zeta)$ donne un moyen de sélectionner une dérivation dans ζ qui n'échoue pas si elle existe. Sur un terme t, $One(\zeta)$ applique la stratégie ζ sur le premier sous-terme immédiat de t où ζ n'échoue pas : $[One(\zeta)](f(t_1, ..., t_n)) = f(t_1, ..., t'_i, ..., t_n)$ si $[\zeta](t_i) = \{t'_i\}$ et pour tout j, $1 \leq j < i$, $[\zeta](t_j) = \emptyset$, et échoue si pour tout i, $[\zeta](t_i)$ échoue.

Les opérateurs de combinaison *All* et *One* sont utilisés en combinaison avec l'opérateur de point fixe pour définir des parcours d'arbre. Par exemple, nous avons $TopDown(\zeta) = \mu x.Sequence(\zeta, All(x))$: la stratégie ζ est d'abord appliquée en haut du terme considéré, puis la stratégie $TopDown(\zeta)$ est appelée récursivement sur tous les sous-termes immédiats du terme.

En addition à la stratégie $First$, ELAN a deux autres stratégies pour décrire le choix de stratégies à appliquer à partir d'un ensemble de stratégies :

don't know $dk(\zeta_1, \ldots, \zeta_n)$ dénote l'ensemble de toutes les dérivations dans ζ_1, ..., ζ_n. Elle échoue si toutes les stratégies échouent.

don't care $dc(\zeta_1, \ldots, \zeta_n)$ dénote l'ensemble de toutes les dérivations d'une stratégie n'échouant pas ζ_i. Elle échoue si toutes les stratégies échouent.

D'autres stratégies de haut-niveau implantent le parcours de terme et la normalisation de termes et sont bien connues dans la littérature système de réécriture de termes :

$$\begin{array}{lcl} TopDown(\zeta) & = & \mu x.\zeta; All(x) \\ BottomUp(\zeta) & = & \mu x.All(x;\zeta) \\ OnceTopDown(\zeta) & = & \mu x.First(\zeta, One(x)) \\ OnceBottomUp(\zeta) & = & \mu x.First(One(x), \zeta) \\ Innermost(\zeta) & = & Repeat(OnceBottomUp(\zeta)) \\ Outermost(\zeta) & = & Repeat(OnceTopDown(\zeta)) \end{array}$$

Exemple 2. Quelques exemples d'application de stratégie :

$$\begin{array}{lcl} [First(a \to b, a \to c)](a) & = & \{b\} \\ [First(a \to c, a \to b)](b) & = & \emptyset \\ [Try(b \to c)](a) & = & \{a\} \\ [Repeat(First(b \to c, a \to b))](a) & = & \{c\} \\ [TopDown(a \to b)](f(g(a,d), h(h(a)))) & = & \{f(g(b,d), h(h(b)))\} \end{array}$$

2 Motivation et Vue d'Ensemble du Calcul

Dans cette partie, nous introduisons un modèle abstrait de calcul biochimique fondé sur les principes de la métaphore chimique, tels qu'implantés dans le γ-calcul, et sur les principes du calcul par réécriture.

La métaphore chimique a été proposée comme paradigme computationnel dans le langage Γ de [BM86]. Celui-ci décrit des calculs en termes d'une solution chimique dans laquelle les molécules représentant des données interagissent au moyen de règles de réaction. Les solutions chimiques sont représentées par des multi-ensembles. Le calcul procède alors par applications de règles de réécriture qui consomment et produisent de nouveaux éléments en fonction des conditions et transformations spécifiés par les règles de réaction. Ce modèle a été utilisé comme fondement pour la définition de la machine abstraite chimique (CHemical Abstract Machine, CHAM) [BB92].

Afin de comprendre les caractéristiques particulières du calcul biochimique abstrait, nous proposons dans ce chapitre de d'abord présenter la métaphore chimique telle qu'implantée dans le γ-calcul et HOCL, puis le calcul par réécriture. Dans un 3^{e} temps, nous parlerons du passage au modèle biochimique.

2.1 Le γ-calcul et HOCL

Le γ-calcul [BFR04, Rad07] a été conçu afin de proposer un calcul d'ordre supérieur de base, ayant les caractéristiques essentielles du paradigme chimique. Celui-ci généralise le modèle chimique en représentant les réactions comme des molécules. La structure fondamentale est le multi-ensemble. La syntaxe du calcul définit une molécule comme étant soit une variable $x \in \mathcal{X}$, soit une abstraction $\gamma\langle p\rangle.M$, soit une solution $\langle M\rangle$, ou encore un multi-ensemble de solutions. Une γ-abstraction est une fonction ayant un seul argument, une variable d'un ensemble $\mathcal{X}$. Nous appelons ces arguments motifs afin de motiver la définition de notre calcul par la suite.

Molécules	$\mathcal{M} ::= \mathcal{X} \mid \gamma\langle\mathcal{P}\rangle.\mathcal{M} \mid \mathcal{M},\mathcal{M} \mid \langle\mathcal{M}\rangle$
Motifs	$\mathcal{P} ::= \mathcal{X}$

Fig. 2.1: Syntaxe du γ-calcul

Les propriétés d'associativité et de commutativité de la construction de multi-ensembles aide à simuler le mouvement brownien des molécules dans une solution. Lorsqu'une abstraction entre en contact avec une solution inerte, une réaction prend place selon la règle

de calcul suivante, nommée γ-réduction :

$$(\gamma\langle x\rangle.M), \langle N\rangle \rightarrow_\gamma M[x := N] \text{ si } \langle N\rangle \text{ est inerte}$$

où une solution est inerte si elle contient uniquement des abstractions et des variables ou uniquement des solutions et des variables. La sémantique du γ-calcul contient, en plus de la règle ci-dessus, deux autres règles pour représenter des réactions dans un contexte. Les règles de localité et de solution indiquent que toute opération sur une molécule donnée, opère également sur un multi-ensemble plus large et une solution resectivement.

La nature d'ordre supérieur du γ-calcul permet d'encoder certains éléments de programmation du calcul sous forme d'expressions complexes, tel qu'on peut le faire dans le λ-calcul, comme par exemple les opérateurs booléens, les entiers, les identifiants de molécule, les paires de molécules, ou la récursivité. Cependant, ces constructions ne sont pas toujours assez expressives pour modéliser des programmes plus complexes. Un des buts du modèle de programmation biochimique tel que proposé par les auteurs du γ-calcul est de modéliser les systèmes autonomes et complexes. Le langage chimique d'ordre supérieur (Higher-Order Chemical Language, HOCL) [BFR06c, BFR06a, BFR07] a été proposé afin de satisfaire cet objectif. Celui-ci étend le γ-calcul avec les éléments de programmation énumérés ci-dessus, mais également des réactions conditionnelles et la capture atomique, et ce qui est plus important, avec le langage de motifs plus riche suivant :

$$\mathcal{P} ::= \mathcal{X} \mid \omega \mid \mathit{ident} = \mathcal{P} \mid \mathcal{P}, \mathcal{P} \mid \langle \mathcal{P} \rangle$$

Soient des motifs $P, P_1, P_2 \in \mathcal{P}$, et des molécules $M, M_1, M_2 \in \mathcal{M}$, ω filtre toute molécule nommée *ident* qui filtre P, le motif P_1, P_2 filtre toute molécule M_1, M_2 telle que P_i filtre M_i, pour $i = 1, 2$, et $\langle P\rangle$ filtre toute solution $\langle M\rangle$ telle que P filtre M. Une abstraction prend alors la forme $\gamma\lfloor C\rfloor P.M$ avec C une condition ; une condition est une molécule qui devrait être évaluée comme une constante true (représentant la γ-abstraction $\gamma\langle x\rangle\lfloor x\rfloor.x$) pour que la réaction ait lieu. Une abstraction est consommée par une γ-réduction, d'où son nom de règle de réaction à usage unique. Ensuite, suivant le mécanisme de récursion encodé par une constante let rec, des règles de réaction à usage multiple sont définies comme des règles de réaction qui ne sont pas consommées par une γ-réduction.

Exemple 3. Considérons quelques exemples de molécules dans le γ-calcul ainsi que des réductions possibles. Nous utilisons des entiers et l'opération d'addition sur les entiers $\_ + \_$ qui peut être encodée par des molécules adéquates en suivant les mêmes idées que le λ-calcul.

1. $\langle\gamma\langle x\rangle.(x+1),\ \langle 1\rangle\rangle,\ \gamma\langle r\rangle.\langle r+1\rangle,\ \langle 1\rangle \rightarrow_\gamma \langle 2\rangle,\ \gamma\langle r\rangle.\langle r+1\rangle,\ \langle 1\rangle \rightarrow_\gamma \langle 2\rangle,\ \langle 2\rangle$
2. $\gamma\langle y\rangle.(\gamma\langle x\rangle.(x+y),\ \langle b\rangle),\ \langle\gamma\langle x\rangle.x,\ \langle a\rangle\rangle \rightarrow_\gamma \gamma\langle y\rangle.(\gamma\langle x\rangle.(x+y),\ \langle b\rangle),\ \langle a\rangle \rightarrow_\gamma \gamma\langle x\rangle.(x+a),\ \langle b\rangle \rightarrow_\gamma b+a$

2.2 Le ρ-calcul

La première version du calcul de réécriture (ou ρ-calcul) a été introduit par H. Cirstea et C. Kirchner [CK01], afin de donner une sémantique au langage ELAN basé sur la

réécriture [CK98]. Dans [CKL02], une version simplifiée du calcul de réécriture a été proposée. Le ρ-calcul étend la réécriture de termes du premier ordre et le λ-calcul. Le ρ-calcul hérite du λ-calcul sa puissance d'ordre supérieur et le traitement explicite des fonctions et de leurs applications. Il a été introduit pour rendre explicite les ingrédients de la réécriture, en particulier les notions de règle de réécriture (ou abstraction) "$\_ \twoheadrightarrow \_$", l'application de règle "$\_\ \_$", et la structure de résultat "$\_ \wr \_$". Dans le ρ-calcul, les λ-abstractions usuelles $\lambda x.t$ sont remplacées par des abstractions de règle $p \twoheadrightarrow t$, où p et t sont deux termes arbitraires, avec p étant appelé un motif, et les variables libres de p étant liées à t. Le ρ-calcul généralise le λ-calcul en abstrayant sur un motif, plutôt que sur une simple variable. Ce genre de généralisation est une caractéristique clé de notre approche.

Sa syntaxe est définie en Fig. 2.2 où $\mathcal{X}$ représente l'ensemble des variables et $\mathcal{K}$ l'ensemble des symboles de fonction. L'opérateur "$\_ \wr \_$" regroupe des termes dans des structures, et, suivant la théorie choisie pour cet opérateur, il fournit des listes, des ensembles ou des multi-ensembles pour représenter des résultats multiples.

Termes	$\mathcal{T} ::= \mathcal{X} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid \mathcal{T} \wr \mathcal{T}$
Motifs	$\mathcal{P} \subseteq \mathcal{T}$

Fig. 2.2: Syntaxe de ρ-calcul

La sémantique de réduction à petit pas du ρ-calcul est définie par les deux règles d'évaluation de la Fig. 2.3. Si σ est une solution de substitution du problème de filtrage $p \ll t_3$, alors l'application de la règle de réécriture sur t_3 s'évalue en $\sigma(t_2)$. L'ensemble des motifs $\mathcal{P}$ n'est pas a priori fixé ($\mathcal{P}$ est un paramètre du calcul), et la puissance de filtrage du ρ-calcul peut être réfutée en utilisant des théories arbitraires. Ainsi, la sémantique du calcul et ses propriétés dépendent essentiellement de ces paramètres.

(ρ)	$(p \twoheadrightarrow t_2)t_3$	$\rightarrow_\rho$	$\sigma_1(t_2) \wr \ldots \wr \sigma_n(t_2) \wr \ldots$ où $\sigma_i \in \mathcal{S}ol(p \ll t_3)$
(δ)	$(t_1 \wr t_2)t_3$	$\rightarrow_\delta$	$t_1 t_3 \wr t_2 t_3$

Fig. 2.3: Sémantique du ρ-calcul

Une caractéristique importante du ρ-calcul est sa capacité à encoder des stratégies de réécriture comme cela est montré dans [CKLW03]. Les stratégies de base sont des règles de réécriture. Une application immédiate de l'utilisation de stratégies de réécriture dans le ρ-calcul est l'encodage de la réécriture conditionnelle [CK01]. La confluence du ρ-calcul a été prouvée pour les motifs algébriques linéaires [CF07].

Exemple 4. Considérons quelques ρ-termes sur un ensemble de constantes a, b, variables x, y, l'opération d'addition $\_ + \_$, et ses réductions [CK01].

1. $(y \twoheadrightarrow ((x \twoheadrightarrow x + y)\ b))\ ((x \twoheadrightarrow x)\ a) \rightarrow_\rho (y \twoheadrightarrow (b + y))\ ((x \twoheadrightarrow x)\ a) \rightarrow_\rho$
   $(y \twoheadrightarrow (b + y))\ a \rightarrow_\rho b + a$
   Le ρ-terme $(y \twoheadrightarrow ((x \twoheadrightarrow x + y)\ b))\ ((x \twoheadrightarrow x)\ a)$ correspond à la seconde γ-molécule de l'exemple 3.
2. $(((x \twoheadrightarrow x + 1) \twoheadrightarrow (1 \twoheadrightarrow x))\ (a \twoheadrightarrow a + 1))\ 1 \rightarrow_\rho (1 \twoheadrightarrow a)\ 1 \rightarrow_\rho a$
   Le ρ-terme $(((x \twoheadrightarrow x + 1) \twoheadrightarrow (1 \twoheadrightarrow x))\ (a \twoheadrightarrow a + 1))\ 1$ n'a pas de règle de réécriture de λ-termes standard correspondante.

2.3 Vers un Calcul Biochimique Abstrait

Une différence entre les modèles chimiques et biochimiques réside dans le fait que la représentation habituelle des molécules dans un modèle biochimique utilise des entités à états, qui peuvent se joindre dans un processus appelé complexation ou association [CZ08]. En conséquence de quoi, dans un modèle biochimique, une molécule peut être vue comme un objet abstrait avec une structure interne décrivant, entre autres, toutes les connexions possibles avec d'autres molécules.

Nous étendons le modèle chimique en considérant une structure abstraite Σ pour les molécules, et pour les règles de calcul. La structure Σ devrait permettre de modéliser aussi bien des connexions entre objets que des actions de création et de suppression de telles connexions. Le résultat est un calcul basé sur des molécules structurées de réécriture, ce que nous appelons $\rho_{\langle\Sigma\rangle}$-calcul. Les objets de première classe du $\rho_{\langle\Sigma\rangle}$-calcul sont des objets Σ-structurés, des règles de réécriture entre objets structurés, et des applications de règles. Ce calcul généralise le λ-calcul, le γ-calcul et le HOCL, via un mécanisme d'abstraction plus puissant, qui considère le filtrage non seulement sur des variables ou des motifs construit sur un langage de motif restreint, mais aussi sur des objets plus génériques construit sur une structure algébrique et un ensemble de variables. Le $\rho_{\langle\Sigma\rangle}$-calcul englobe également le calcul de réécriture [CK01] et le calcul de réécriture de graphes de termes [BBCK05] en considérant que les structures arborescentes de termes et les structures de graphes de graphes de termes respectivement sont des structures spéciales.

Le $\rho_{\langle\Sigma\rangle}$-calcul est conçu comme un formalisme de modélisation de systèmes complexes avec une topologie dynamique, où les entités dans un certain état ont une structure particulière et interagissent de manière concurrente en suivant un comportement décrit par des règles de réécriture. Grâce à la nature parallèle intrinsèque de la réécriture sur des sous-termes disjoints et à l'application de règles décentralisée, nous modélisons un mouvement Brownien, un principe de base du modèle chimique consistant en « une distribution libre et une réaction sous-spécifiée entre molécules » [BRF04], en considérant les molécules comme étant des objets structurés.

En considérant le $\rho_{\langle\Sigma\rangle}$-calcul comme un cadre pour la modélisation de systèmes, nous obtenons un gain en expressivité en choisissant les descriptions adéquates des états, et une grande flexibilité dans la modélisation de systèmes dynamiques. Par exemple, des règles peuvent être consommées par application, et de nouvelles règles peuvent être créées par l'application d'autres règles. Alors, à la place d'un comportement non déterministe

(et potentiellement sans terminaison) de l'application des abstractions, on peut vouloir introduire un contrôle sur la composition ou le choix des règles à appliquer. La notion d'abstraction se révèle assez puissante pour exprimer un tel contrôle, grâce aux concepts de stratégie et de réécriture stratégique. De plus, les stratégies permettent d'exploiter l'information d'échec.

2.4 Structure de la Partie

Dans le reste de cette partie, nous présentons la syntaxe et la sémantique d'un calcul biochimique abstrait. En Chapitre 3, nous définissons la syntaxe des entités de base du calcul. Nous continuons en Chapitre 4 avec la définition de la sémantique de réduction à petits pas du calcul, en nous basant sur trois étapes : le réchauffement, l'application, et le refroidissement. En Chapitre 5, nous définissons des stratégies sous forme d'objets du calcul et prouvons la correction de leur définition. Ensuite, nous raffinons le calcul de manière à récupérer un échec d'application avec la stratégie d'interaction initiale et des objets structurés. Comme application des stratégies, nous définissons des stratégies persistantes qui ne sont pas consommées par des interactions. En Chapitre 6, nous définissons une sémantique de réduction à gros grain, de manière à rendre les invisibles les interactions échouées dans l'évolution du système. Nous terminons par (i) une discussion sur d'autres stratégies possibles dans le calcul (Chapitre 7), (ii) une comparaison avec le γ-calcul et HOCL (Chapitre 8), et (iii) remarques de conclusion et idées pour des travaux futurs (Chapitre 9).

3 Syntaxe

Dans ce qui suit, nous décrivons, étape par étape, tous les éléments définissant la syntaxe du calcul, ainsi que l'application de substitution et le filtrage.

3.1 Objets structurés

Nous modélisons les états d'un système via des objets structurés décrits par des objets d'une catégorie fixée Σ. En accord avec l'intuition biochimique, les objets sont dans un mouvement Brownien, i.e. flottant dans une soupe biochimique, formant ainsi un objet structuré. Nous simulons le mouvement Brownien en considérant une opération de juxtaposition sur deux objets structurés qui construit un objet également structuré. Formellement, nous considérons une catégorie strictement monoïdale, et strictement symétrique (appelée *sssmc*, pour strict symmetric strict monoidal category) : $SSSMC_\Sigma = (\Sigma, \_\ \_, \varepsilon)$ avec $\_\ \_ : \Sigma \times \Sigma \to \Sigma$ un bifoncteur permettant de juxtaposer deux objets structurés, et ε l'objet vide avec la condition que la flèche de Σ satisfasse l'axiome d'un *sssmc*. Soit $\mathcal{O}$ dénotant $Obj(SSSMC_\Sigma)$ et $\mathbb{T}$ dénotant l'ensemble suivant d'axiomes définissant un *sssmc* :

$$\begin{aligned} O_1\ O_2 &=_{\mathbb{T}} O_2\ O_1 \\ (O_1\ O_2)\ O_3 &=_{\mathbb{T}} O_1\ (O_2\ O_3) \\ O_1\ \varepsilon &=_{\mathbb{T}} O_1 \end{aligned}$$

pour tout objets O_1, O_2, O_3 dans $\mathcal{O}$. Les flèches de $SSSMC_\Sigma$ sont exactement les flèches de Σ qui satisfont les axiomes de $\mathbb{T}$.

Puisque le bifoncteur $\_\ \_$ correspond à un opérateur associatif et commutatif sur des objets structurés, nous pouvons le voir comme un opérateur variadique, et nous le représentont habituellement dans une forme infixe non parenthésée.

Exemple 5. En instanciant la catégorie Σ, nous obtenons des objets structurés particuliers, comme suit :

Multi-ensembles de constantes Soit Σ une catégorie petite avec $\mathrm{Ob}(\Sigma)$ un ensemble de constantes. Nous prenons alors une flèche pour chaque paire d'objets distincts, et une flèche d'identité pour chaque objet. Par exemple, si $a, b, c, d \in \mathrm{Ob}(\Sigma)$, alors *aba*, *c*, *bdbbb*, *cabbacccb* sont des exemples de multi-ensembles. Nous remarquons que cette structure n'a pas de caractéristiques intrinsèques permettant d'exprimer des connexions entre objets ; ainsi elle est plus adéquate pour décrire un modèle chimique qu'un modèle biochimique.

Multi-ensemble de termes. Si, au lieu de constantes comme ci-dessus, nous considérons des termes algébriques sur une signature de premier ordre donnée $\mathcal{F}$, et un ensemble

de variables, nous obtenons des multi-ensembles de termes. Soient $a, b, c, f, g, h \in \mathcal{F}$, avec a, b, c des constantes, f et g unaires, et h binaire. Alors $f(c)\ f(c)\ g(h(a,c))\ h(a, g(f(b)))$ et $g(a)$ sont deux multi-ensembles de termes.

Graphes. Nous considérons la catégorie de graphes Graph dont les objets sont des graphes, et dont les flèches sont des morphismes de graphes. Alors, le bifoncteur _ _ correspond à une somme disjointe de graphes, et ε au graphe vide.

Nous considérons une sous-classe particulière d'objets dans $\mathcal{O}$ comme des variables, et nous la dénotons par $\mathcal{X}_{\mathcal{O}}$. Une variable est un objet avec des flèches vers tout autre objet. La définition exacte des variables dans un objet structuré dépend de chaque instance particulière de Σ. Nous dénotons par $\mathcal{V}ar(O)$ la classe des variables apparaissant dans la structure d'un objet O.

3.2 Abstractions

Nous définissons une abstraction comme étant une paire ordonnée de deux objets structurés dans $\mathcal{O}$, telle que le premier objet est appelé membre de gauche, et le second membre de droite, et étant équipée avec un morphisme entre ces deux membres. Ce morphisme spécifie une transformation de l'objet du membre de gauche en l'objet du membre de droite. Soit $\mathcal{A}_o$ l'ensemble d'abstractions suivant :

$$\mathcal{A}_0 \ ::= \{(O_1, f, O_2) \mid O_1, O_2 \in \mathcal{O}, f \in SSMC_{\Sigma}(O_1, O_2)\}$$

Pour une abstraction A, nous dénotons habituellement par L_A et R_A les membres de gauche et de droite respectivement. Nous représentons une abstraction (O_1, f, O_2), comme ayant un membre de gauche O_1 suivi par un opérateur $\Rightarrow$ qui n'apparaît pas dans les objets structurés de Σ, et un membre de droite O_2, ainsi $O_1 \overset{f}{\Rightarrow} O_2$. Hormi lorsque cela est nécessaire, nous ne spécifions pas le morphisme entre les deux membres d'une abstraction. Ainsi, nous pouvons aussi définir $\mathcal{A}_0$ comme la classe d'abstraction entre objets structurés $\mathcal{O} \Rightarrow \mathcal{O}$.

Exemple 6. Les abstractions définies sur le *sssmc* construit pour la catégorie de graphes avec des morphismes partiaux $\mathrm{Graph}^{\mathrm{P}}$ correspond au concept des transformations de graphes formalisé au moyen de l'approche de "simple pushout" (SPO).

Dans une seconde étape, nous définissons un morphisme $g : (L_A \overset{f}{\Rightarrow} R_A) \rightarrow (L_{A'} \overset{f'}{\Rightarrow} R_{A'})$ sur une abstraction de $\mathcal{A}_0$, comme une paire de morphismes sur des objets structurés $\langle g_L : L_A \rightarrow L_{A'}, g_R : R_A \rightarrow R_{A'} \rangle$ telle que $g_R \circ f = f' \circ g_L$. Nous pouvons alors définir une catégorie $\mathrm{Abs}_0(\mathcal{O})$ d'abstractions sur $\mathcal{O}$ avec $\mathcal{A}_0$ la classe d'objets, et des morphismes sur des abstraction de $\mathcal{A}_0$ sous forme de flèches.

Nous étendons la classe d'abstractions $\mathcal{A}_0$ avec une nouvelle classe d'abstractions définie comme $\{(A, g, A') \mid A, A' \in \mathcal{A}_0, g \in \mathrm{Abs}_0(\mathcal{O})(A, A')\}$ ou $(\mathcal{O} \Rightarrow \mathcal{O}) \Rightarrow (\mathcal{O} \Rightarrow \mathcal{O})$ pour résumer. En conséquence, la classe d'abstractions $\mathcal{A}_0$ est définie à présent par :

$$\mathcal{A}_0 ::= \mathcal{O} \Rightarrow \mathcal{O} \mid (\mathcal{O} \Rightarrow \mathcal{O}) \Rightarrow (\mathcal{O} \Rightarrow \mathcal{O})$$

3.3 Molécules Abstraites

L'idée principale de ce calcul est d'avoir toutes les entités au même niveau, que ce soient des objets représentant les états du système, ou des abstraction représentant différents comportements du système, ou encore leurs interactions. Nous appelons une telle entité une molécule abstraite. Nous définissons graduellement la classe des molécules abstraites de façon à inclure les abstractions dont le membre de gauche est un objet structuré, et le membre de droite est une molécule abstraite, et également les abstractions sur $\mathcal{A}_0$.

Soit M_0 une catégorie avec $Ob(\mathrm{M}_0) = \mathcal{O} \cup \mathcal{A}_0$, et $Arr(\mathrm{M}_0) = Arr(SSSMC_\Sigma) \cup Arr(\mathrm{Abs}_0(\mathcal{O}))$.

Alors, à partie de M_0 et du bifoncteur $\_\ \_$ vu précédemment, et de l'objet vide ε, nous pouvons construire un premier *sssmc* pour les molécules abstraites $SSSMC_{\mathrm{M}_0}$. Soit $\mathcal{M}_0$ la classe des objets de $SSSMC_{\mathrm{M}_0}$.

Nous pouvons alors définir des abstractions avec des objets structurés dans le membre de gauche, et des molécules abstraites dans le membre de droite, où le morphisme entre ces deux membres est un morphisme entre objets structurés :

$$\mathcal{A}_1 = \{(O, f, M) \mid O \in \mathcal{O}, M \in Ob(SSSMC_{\mathrm{M}_0}), f \in \mathrm{M}_0(O, M)\}$$

En résumé, nous pouvons représenter la classe de telles abstractions par $\mathcal{O} \Rightarrow \mathcal{M}_0$.

En nous fondant sur ces deux nouvelles classes d'abstractions, nous considérons une nouvelle catégorie M avec $Ob(\mathrm{M}) = Ob(\mathrm{M}_0) \cup \mathcal{A}_1$, et $Arr(\mathrm{M}_0) = Arr(SSSMC_\Sigma) \cup Arr(\mathrm{Abs}_0(\mathcal{O}))$. Alors, similairement à la construction de $SSSMC_{\mathrm{M}_0}$, nous considérons le *sssmc* associé à M dont les objets sont exactement les molécules abstraites. Soit $\mathcal{M}$ la classe des objets de $SSSMC_{\mathrm{M}}$.

Nous remarquons que, en suivant la procédure ci-dessus, nous pouvons construire itérativement des abstractions et molécules d'ordre supérieur.

Ayant des molécules abstraites construites en utilisant des concepts catégoriques, nous donnons ci-dessous leur définition schématique, que nous utiliserons dans le reste du chapitre, où nous ne ferons plus référence aux concepts catégoriques. Soit $\mathcal{X}$ l'union de $\mathcal{X}_\mathcal{O}$ avec une classe de variables pour un type quelconque de molécules abstraites. Alors, la syntaxe des objets du calcul est la suivante :

$$\begin{array}{lll} \mathcal{A}_0 & ::= & \mathcal{O} \Rightarrow \mathcal{O} \mid (\mathcal{O} \Rightarrow \mathcal{O}) \Rightarrow (\mathcal{O} \Rightarrow \mathcal{O}) \\ \mathcal{M}_0 & ::= & \mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{M}_0\ \mathcal{M}_0 \\ \mathcal{A} & ::= & \mathcal{A}_0 \mid \mathcal{O} \Rightarrow \mathcal{M}_0 \\ \mathcal{M} & ::= & \mathcal{X} \mid \mathcal{M}_0 \mid \mathcal{A} \mid \mathcal{M}\ \mathcal{M} \end{array}$$

Toute abstraction dans $\mathcal{A}$ a un opérateur flèche principal $\Rightarrow$ distingué, celui qui définit le morphisme entre les deux membres de l'abstraction.

Comme nous l'avons mentionné précédemment, $\mathbb{T}$ dénote l'ensemble des axiomes définissant les propriétés du *sssmc* des objets structurés et les morphismes entre eux. En conséquence, $\mathbb{T}$ génère une relation de congruence structurelle entre les objets dans $\mathcal{O}$. Puisque les abstractions et les molécules abstraites sont construites à partir d'objets structurés, il est naturel que nous étendions la relation de congruence structurelle sur aux molécules abstraites.

Définition 25 (Congruence structurelle sur des molécules abstraites). La relation de congruence structurelle sur des molécules abstraites est la relation de congruence la plus petite fermée par rapport à _ _ (juxtaposition) sur des ensembles de molécules abstraites, qui étende la congruence structurelle $=_{\mathbb{T}}$ sur des objets structurés de Σ, satisfaisant ainsi les axiomes suivants :

$$\begin{array}{rcl} M_1\ M_2 & =_{\mathbb{T}} & M_2\ M_1 \\ M_1\ \varepsilon & =_{\mathbb{T}} & M_1 \\ (M_1\ M_2)\ M_3 & =_{\mathbb{T}} & M_1\ (M_2\ M_3) \end{array}$$

pour tout $M_1, M_2, M_3 \in \mathcal{M}$.

L'opérateur de juxtaposition, tel qu'axiomatisé par la théorie $\mathbb{T}$ construit des multi-ensembles. De manière équivalente, nous pouvons considérer que la juxtaposition correspond à un opérateur variadique, ainsi que les parenthèses ne sont plus nécessaires. Si l'on veut un type de collection différent, de nouveaux axiomes doivent être ajoutés et/ou retirés de $\mathbb{T}$. Par exemple, si nous retirons l'aspect symétrique strict de la catégorie monoïdale (i.e., la commutativité), nous obtenons des listes, alors que si nous ajoutons un axiome pour l'idempotence, nous obtenons des ensembles.

3.4 Sous-objets, Sous-molécules, Substitutions, Filtrage

Tout objet structuré O qui n'est ni une variable, ni un opérateur constant de Σ peut être décomposé en deux objets structurés O_1 et O_2 collés ensembles, à partir d'une position relative de l'un par rapport à l'autre. Une position est décrite par une information de voisinage. Nous dénotons une telle décomposition par $O_1\lfloor O_2 \rfloor_{\mathcal{B}}$ and nous appelons O_1 un contexte, O_2 un sous-objet ou sous-molécule et $\mathcal{B}$ une information de voisinage.

Une abstraction a également une structure particulière donnée par la structure de ses membres et la flèche, ainsi elle peut être décomposée en un contexte, une sous-molécule, et une information de voisinage. Considérons par exemple l'abstraction $L \Rightarrow B$. Si nous décomposons chaque membre comme $L^-\lfloor L_1 \rfloor_{\mathcal{B}'}$ et $R^-\lfloor R_1 \rfloor_{\mathcal{B}''}$, alors l'abstraction $L \Rightarrow R$ a une décomposition $(L^- \Rightarrow R^-)\lfloor L_1 \Rightarrow R_1 \rfloor_{\mathcal{B}' \Rightarrow \mathcal{B}''}$.

Exemple 7. Considérons Σ une catégorie de graphes. Nous illustrons en Figure 3.1 une décomposition d'une abstraction sur des graphes $L \overset{f}{\Rightarrow} R$ en la sous-molécule $L_1 \overset{f'}{\Rightarrow} R_1$, le contexte $L^- \overset{f''}{\Rightarrow} R^-$, et l'information de voisinage $\mathcal{B}' \overset{f'''}{\Rightarrow} \mathcal{B}''$ où $\mathcal{B}'$ consiste en les arêtes $(1,4)$ et $(3,4)$, $\mathcal{B}''$ consiste en l'arête $(4,3)$, et f''' associe l'arête $(3,4)$ à l'arête $(4,3)$. Nous remarquons que f est décomposée en trois morphismes f' sur les sous-molécules, f'' sur les contextes, et f''' sur les informations de voisinage.

De par la structure non-hiérarchique des graphes, dans une décomposition de molécule de graphe, nous pouvons échanger le contexte avec la sous-molécule, et nous obtenons la même molécule de graphe initiale comme nous pouvons le voir dans l'exemple ci-dessus.

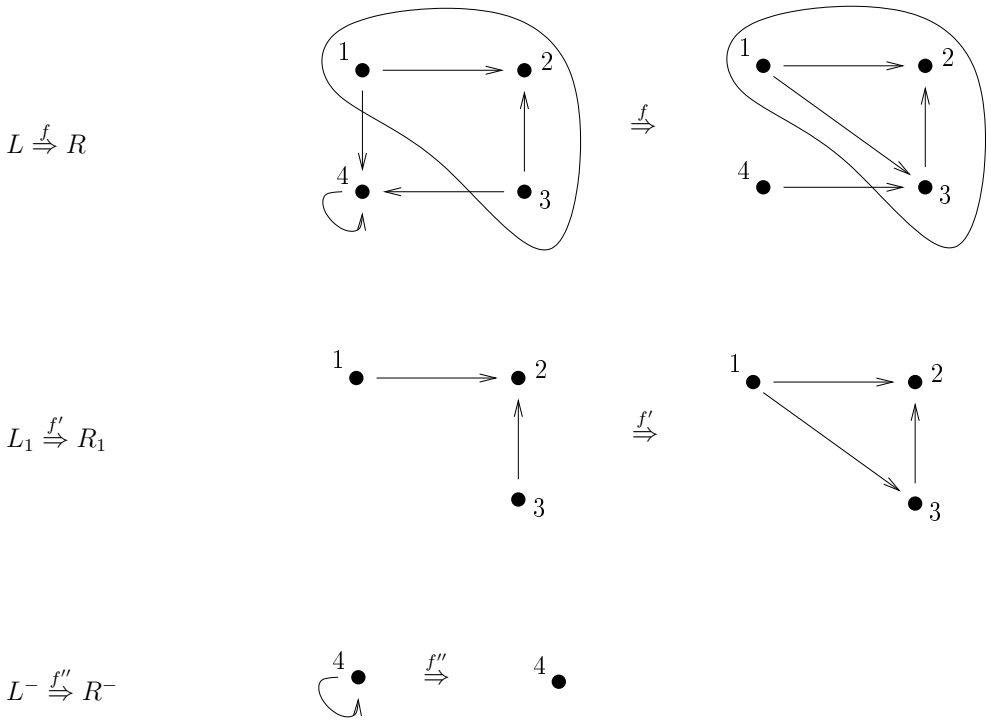


Fig. 3.1: A decomposition of an abstraction on graphs

Le seul type de lieur dans le $\rho_{\langle\Sigma\rangle}$-calcul est l'abstraction. Ainsi, les notions de variable libre et liée sont similaires à celles de ρ-calcul, i.e., dans $O \Rightarrow M$ les variables libres de O sont liées à M.

Définition 26 (Variables libres et liées). Étant donné une molécule abstraite M, son ensemble de variables libres $FV(M)$ et son ensemble de variables liées $BV(M)$ sont définis comme suit :

- si $M = X$ alors $FV(M) = \{X\}$ et $BV(M) = \emptyset$;
- si $M = O$ alors $FV(M) = \mathcal{V}ar(O)$ et $BV(M) = \emptyset$;
- si $M = M_1 \Rightarrow M_2$ alors $FV(M) = FV(M_2) \setminus FV(M_1)$ et $BV(M) = BV(M_2) \cup BV(M_1) \cup FV(M_1)$;
- si $M = M_1\ M_2$ alors $FV(M) = FV(M_1) \cup FV(M_2)$ et $BV(M) = BV(M_1) \cup BV(M_2)$.

Comme dans tout calcule impliquant des lieurs, nous travaillons modulo l'α-convention et modulo l'hygiène-convention de Barendregt [Bar84], i.e. les variables libres et liées ont des noms différents.

Définition 27 (Substitution). Une substitution σ est une association entre l'ensemble des variables $\mathcal{X}$ et l'ensemble des molécules abstraites $\mathcal{M}$, $\sigma : \mathcal{X} \mapsto \mathcal{M}$. Nous l'écrivons sous la forme $\{X_1 \mapsto M_1, \ldots, X_n \mapsto M_n\}$ s'il y a seulement un nombre fini de variables qui

ne sont pas associées à elles-mêmes. Il s'étend de manière unique à un homomorphisme de $\mathcal{M}$ vers $\mathcal{M}$.

L'application d'une substitution finie $\sigma = \{X_1 \mapsto M_1, \ldots, X_n \mapsto M_n\}$ à une molécule abstraite M, $\sigma(M)$, est obtenue en substituant M_i pour X_i, $1 \leq i \leq n$, en utilisant le renommage de variable approprié pour éviter la capture des variables libres [CK01].

Étant donné une molécule abstraite, nous nous intéressons à la transformation de certaines de ses sous-molécules, qui vérifient certaines caractéristiques. De telles caractéristiques définissent un motif. Le procédé d'identification de sa localisation dans la molécule donnée, et d'association avec un sous-objet est appelé sous-filtrage. Si la transformation concerne la molécule entière, i.e., le motif doit être identifié à une molécule par substitution, alors seule une association entre les éléments du motif et ceux de la sous-molécule doit être fournie, et le procédé est appelé filtrage.

Définition 28 (Sous-filtrage modulo $\mathbb{T}$). Une équation sous-filtrante modulo $\mathbb{T}$ est une paire ordonnée de molécules abstraites $M \lessdot_{\mathbb{T}} M'$. Une solution d'une équation sous-filtrante est un triplet $(\sigma, M'', \mathcal{B})$ où Σ est une substitution, M'' une molécule abstraite, et $\mathcal{B}$ une information de voisinage telle que $M''\lfloor\sigma(M)\rfloor_{\mathcal{B}} =_{\mathbb{T}} M'$. Nous appelons un problème sous-filtrant $M \lessdot_{\mathbb{T}} M'$ un problème filtrant si $\sigma(M) =_{\mathbb{T}} M'$, ou de manière équivalente, M'' est la molécule vide, et nous la dénotons par $M \ll M'$. Nous dénotons habituellement une solution sous-filtrante par le symbol ς, et l'ensemble de toutes les solutions de l'équation sous-filtrante $M \lessdot_{\mathbb{T}} M'$ par $\mathcal{S}ol(M \lessdot_{\mathbb{T}} M')$.

Pour chaque instantiation de la structure Σ des objets et $\mathbb{T}$, un algorithme de sous-filtrage modulo $\mathbb{T}$ devrait également être fourni. Pour des raisons d'optimisation, il peut être utile de définir une relation de congruence entre les solutions filtrantes, et de considérer alors uniquement les solutions correspondant aux représentants de classe ; ceci représente un moyen de réduire l'explosion combinatoire qui peut apparaître dans la situation où des multi-ensembles d'objets sont manipulés.

3.5 Mondes

Nous définissons la construction de monde en plaçant les molécules dans un environnement où elles peuvent interagir librement. En d'autres termes, un monde est une encapsulation d'une molécule abstraite existant dans un état du système modélisé. L'environnement est « conscient » de toutes les molécules abstraites qu'il contient. Nous représentons cette connaissance en considérant un opérateur variadique permutatif [ ] construisant un monde qui prend en argument toutes les molécules abstraites de l'environnement. Nous dénotons par $[M]$ un monde contenant la molécule abstraite $M \in \mathcal{M}$. Si M est une juxtaposition de m variables, n objets structurés de $\mathcal{O}$ et p abstractions, alors le monde $[M]$ est une construction basée sur $(m+n+p)$ composants. La classe des mondes $\mathcal{V}$ est définie comme suit :

$$\mathcal{V} \ ::= \ \mathcal{Y} \mid [\mathcal{M}]$$

où $\mathcal{Y}$ est un ensemble de variables.

Une représentation graphique intuitive de l'environnement en tant que monde, considère une boîte rectangulaire agissant comme un conteneur, fournissant une intuition d'un environnement. Nous illustrons en Figure 3.2 un monde consistant en deux objets structurés et trois abstractions. ‘

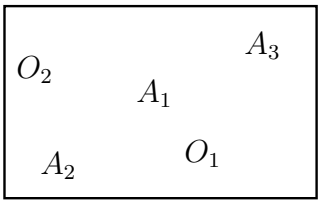


Fig. 3.2: Représentation sous forme de boîte d'un monde consistant en les abstractions A_1, A_2, et A_3, et les objets structurés O_1 et O_2.

3.6 Structures de Mondes ou Multivers

Il peut y avoir différentes façons pour un système d'évoluer d'un monde (un état) via une interaction réussie entre deux molécules. Afin de collecter tous les résultats possibles d'évolution, nous introduisons un opérateur variadique permutatif { } pour construire une structure de mondes en les juxtaposant. Une structure de mondes est aussi appelée un multivers (ou méta-univers, puisqu'un monde peut aussi être appelé univers). Les mondes possibles dans un multivers sont appelés mondes alternatifs. Soit $\mathcal{W}$ la classe des multivers. Alors un multivers consiste soit en une variable dans une ensemble $\mathcal{Z}$, soit une juxtaposition de mondes, ou encore une juxtaposition de multivers.

$$\mathcal{W} \ ::= \ \mathcal{Z} \mid \{\mathcal{V} \ldots \mathcal{V}\} \mid \{\mathcal{W} \ldots \mathcal{W}\}$$

La théorie sous-jacente de l'opérateur { } est donné par la congruence structurelle suivante :

Définition 29 (Congruence structurelle sur les mondes). La relation de congruence structurelle sur les multivers est la plus petite relation de congruence satisfaisant la propriété de { } étant variadique, et les axiomes d'applatissement suivants :

$$\begin{array}{rcl} \{\{V_1^1 \ldots V_{n_1}^1\} \ldots \{V_1^k \ldots V_{n_k}^k\}\} & \equiv & \{V_1^1 \ldots V_{n_1}^1 \ldots V_1^k \ldots V_{n_k}^k\} \\ \{\{V_1 \ldots V_n\}\ V_1' \ldots V_m'\} & \equiv & \{V_1 \ldots V_n\ V_1' \ldots V_m'\} \end{array}$$

3.7 Vue d'Ensemble de la Syntaxe de Base du Calcul

En Figure 3.3, nous résumons la syntaxe du calcul telle que définie jusqu'alors, qui inclut des objets structurés, des abstractions, des molécules abstraites, des mondes et des multivers.

(Objets structurés)	$\mathcal{O}$		
	$\mathcal{A}_0$	::=	$\mathcal{O} \Rightarrow \mathcal{O}$
		|	$(\mathcal{O} \Rightarrow \mathcal{O}) \Rightarrow (\mathcal{O} \Rightarrow \mathcal{O})$
	$\mathcal{M}_0$	::=	$\mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{M}_0\ \mathcal{M}_0$
(Abstraction)	$\mathcal{A}$	::=	$\mathcal{A}_0 \mid \mathcal{O} \Rightarrow \mathcal{M}_0$
(Molécules abstraites)	$\mathcal{M}$	::=	$\mathcal{X} \mid \mathcal{M}_0 \mid \mathcal{A} \mid \mathcal{M}\ \mathcal{M}$
(Mondes)	$\mathcal{V}$	::=	$\mathcal{Y} \mid [\mathcal{M}]$
(Multivers)	$\mathcal{W}$	::=	$\mathcal{Z} \mid \{\mathcal{V} \ldots \mathcal{V}\} \mid \{\mathcal{W} \ldots \mathcal{W}\}$

Fig. 3.3: Syntaxe de base de $\rho_{\langle\Sigma\rangle}$-calcul

4 Sémantique à Petit Pas

4.1 Sémantique de Base

La sémantique de $\rho_{\langle\Sigma\rangle}$-calcul correspond à un système dont l'état initial est un monde de la forme $[A_1 \ldots A_n\ O_1 \ldots O_m]$, avec $n, m \geq 0$. Chaque abstraction peut interagir de manière non-déterministe avec une molécule abstraite du même monde. Une telle interaction peut produire un résultat ayant du sens si l'équation filtrante entre le membre de gauche de l'abstraction et la molécule abstraite est valide ; dans le cas contraire, l'interaction non-réussie exprime simplement le mouvement Brownien modélisé par l'opérateur de juxtaposition permutatif.

Soit A une abstraction décrite par $L_A \Rightarrow R_A$ avec ξ le morphisme inclu dans l'opérateur flèche, et M une molécule abstraite telle qu'il existe une solution sous-filtrante $\varsigma \in \mathcal{S}ol(L_A \prec\!\!\prec M)$ avec $\varsigma = (\sigma, M', \mathcal{B})$; l'existence d'une solution sous-filtrante garantit le succès de l'interaction. L'interaction entre l'abstraction A et la molécule abstraite M produit une transformation de M selon A en remplaçant dans M la sous-molécule $\Sigma(L_A)$ par $\Sigma(R_A)$ en utilisant l'information de voisinage $\mathcal{B}$ mise à jour suivant ξ. Ainsi, si M a une décomposition $M'\lfloor\sigma(L_A)\rfloor_{\mathcal{B}}$, alors le résultat de la transformation de M suivant A est une molécule abstraite obtenue à partir de M en remplaçant la sous-molécule $\sigma(L_A)$ par $\sigma(R_A)$.

Une interaction entre une abstraction et une molécule abstraite est décrite par l'une des règles suivantes :

$$
\begin{array}{lcll}
A\ M & \longrightarrow & \{[\varsigma_1(R_A)] \ldots [\varsigma_n(R_A)]\} & \text{si } \mathcal{S}ol(L_A \prec\!\!\prec M) = \{\varsigma_1, \ldots, \varsigma_n\} \\
A\ M & \longrightarrow & A\ M & \text{sinon}
\end{array}
$$

Fig. 4.1: Sémantique de base de $\rho_{\langle\Sigma\rangle}$-calcul

Une application réussie de l'abstraction A à une molécule abstraite M, appartenant toutes deux au même monde, se réduit en une structure de mondes alternés (ou sous-états) $\{[M_1] \ldots [M_n]\}$ telle que $\mathcal{S}ol(L_A \prec\!\!\prec M) = \{\varsigma_1, \ldots, \varsigma_n\}$ et $\varsigma_i(R_A) = M_i$. Un échec de filtrage indique que l'interaction entre les deux entités est impossible, d'où le fait que l'abstraction initiale et l'objet structurés sont retournés inchangés. Cette approche est similaire à celle de Alchemy [FB96], où si deux molécules (représentées comme des λ-expressions typées) se collisionnent de manière aléatoire dans une réaction et leurs types sont incompatibles, alors la réaction est considérée élastique.

4.2 Rendre l'Application Explicite

Les règles d'évaluation données en Figure 4.1 donnent l'idée principale de l'application d'une abstraction sous forme d'une réécriture en une étape. Nous n'avons pas spécifié comment les opérations de filtrage et de remplacement, i.e., l'application effective de l'abstraction, sont réalisées, puisque ces opérations, habituellement définies au niveau de l'instance du calcul, dépendent de la structure Σ des objets, paramètre du calcul.

Nous raffinons la sémantique des interactions de manière à comprendre quels sont les acteurs et afin de mieux contrôler le procédé d'application. Revenons à la situation où l'application n'a pas réussie (le filtrage a échoué) : l'interaction est élastique et les acteurs initiaux sont retournés. Nous introduisons un second niveau d'organisation interne et de process, où des tests appropriés peuvent être réalisés. Nous définissons un opérateur d'application qui permet l'isolation d'une molécule abstraite M et une abstraction A choisie aléatoirement pour l'interaction. Ainsi, en plus de permettre différents tests d'être réalisés pendant l'application, l'opération d'application isole également ou bloque l'accès aux autres abstractions aux deux entités interagissant. Un monde d'application est construit en utilisant un opérateur d'application @ qui prend en arguments une abstraction et une molécule abstraite M.

En utilisant un mécanisme similaire à celui de CHAM, une interaction prend place dans un système après une procédure de préparation, et qui restructure le système, un procédé de réchauffement. Le réchauffement isole une abstraction A et une molécule abstraite M dans un monde d'application. Ceci est exprimé au moyen de la règle d'évaluation suivante :

$$\text{(Rchauffement)} \qquad [N\ A\ M] \longrightarrow_h [N\ A@M] \tag{4.1}$$

Les étapes calculant l'application d'une abstraction isolée A sur une molécule abstraite M, incluant les opérations de filtrage et d'application de substitutions (remplacement), sont exprimées par les règles d'évaluation suivantes :

$$\text{(Application)} \qquad A@M \longrightarrow_a \{[\varsigma_1(R_A)] \dots [\varsigma_n(R_A)]\} \quad \text{si} \quad \mathcal{S}ol(L_A \prec\!\!\!\prec M) = \{\varsigma_1, \dots, \varsigma_n\} \tag{4.2}$$

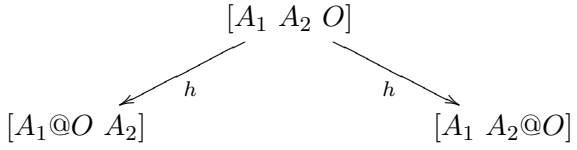
$$\text{(ApplicationEchec)} \qquad A@M \longrightarrow_{\text{af}} A\ M \quad \text{si} \quad \mathcal{S}ol(L_A \prec\!\!\!\prec M) = \emptyset \tag{4.3}$$

Quand l'application d'une abstraction sur un objet structuré réussit dans l'état actuel du système, une règle de refroidissement, le dual de la règle de réchauffement, prend place et se charge de reconstruire l'état du système en branchant le résultat de la réécriture dans l'environnement. Le contexte consistant en des molécules ne participant pas à l'interaction, est dupliqué pour chaque résultat au moyen de la règle d'évaluation suivante :

$$\text{(Refroidissement)} \qquad [N\ \{[M_1] \dots [M_n]\}] \longrightarrow_c \{[N\ M_1] \dots [N\ M_n]\} \tag{4.4}$$

4.3 Au sujet de la Confluence Locale

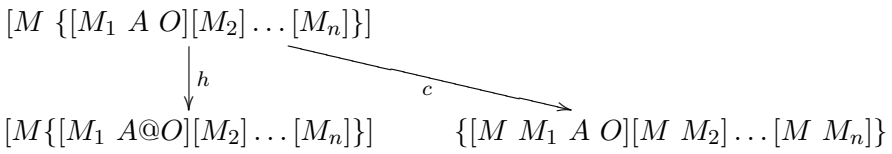
Le modèle chimique est hautement non-déterministe : d'une part, via la stratégie d'interaction intrinsèque non-déterministe premettant de choisir les molécules abstraites à réchauffer, et d'autre part, via le choix de la solution sous-filtrante. En considérant une structure de tous les résultats possibles d'une application, nous éliminons le non-déterminisme apporté par des solutions multiples au sous-filtrage. Cependant, le degré de non-déterminisme restant rend le calcul localement non-confluent. Prenons un exemple :



Le monde consistant en deux abstractions A_1 et A_2 et un objet structuré O est chauffé, donnant naissance à deux possibilités correspondant à chacune des deux abstractions à appliquer à O. Dans ce cas particulier, où la paire critique créée par les deux abstractions est joignable, la réduction converge, mais en général, elle ne converge pas. En conséquence, la relation de réécriture induite par les règles de réduction donnant la sémantique de base du calcul abstrait biochimique n'est pas locallement confluent. Dans de telles conditions, la confluence peut habituellement être forcée en utilisant des stratégies d'évaluation appropriées. En Chapitre 5, nous donnons quelques moyens pour ajouter des stratégies au calcul, afin de contrôler les interactions entre abstractions et objets structurés.

4.4 D'abord refroidir, puis réchauffer

Analysons les conditions pour le réchauffement et le refroidissement d'un monde, et l'importance de leur succession dans le temps. Les règles de réduction de réchauffement et de refroidissement donnent naissance aux paires critiques suivantes :



Si nous considérons que $A@O \longrightarrow_a \{[N_1]\ldots[N_k]\}$, la paire critique ci-dessus est joignable, puisque d'une part :

$$
\begin{array}{l}
[M\ \{[M_1\ A@O][M_2]\ldots[M_n]\}] \longrightarrow_a \\
[M\ \{[M_1\ \{[N_1]\ldots[N_k]\}][M_2]\ldots[M_n]\}] \longrightarrow_c \\
[M\ \{\{[M_1\ N_1]\ldots[M_1\ N_k]\}[M_2]\ldots[M_n]\}] \equiv \\
[M\ \{[M_1\ N_1]\ldots[M_1\ N_k][M_2]\ldots[M_n]\}] \longrightarrow_c \\
\{[M\ M_1\ N_1]\ldots[M\ M_1\ N_k][M\ M_2]\ldots[M\ M_n]\}
\end{array}
$$

et d'autre part :

$$\begin{array}{l}\{[M\ M_1\ A\ O][M\ M_2]\ldots[M\ M_n]\} \longrightarrow_h \\ \{[M\ M_1\ A@O][M\ M_2]\ldots[M\ M_n]\} \longrightarrow_a \\ \{[M\ M_1\ \{[N_1]\ldots[N_k]\}][M\ M_2]\ldots[M\ M_n]\} \longrightarrow_c \\ \{\{[M\ M_1\ N_1]\ldots[M\ M_1\ N_k]\}[M\ M_2]\ldots[M\ M_n]\} \equiv \\ \{[M\ M_1\ N_1]\ldots[M\ M_1\ N_k][M\ M_2]\ldots[M\ M_n]\}\end{array}$$

Si $A@O \longrightarrow_{af} A\ O$ nous obtenons toujours la joignabilité. Ce résultat montre que l'ordre de réduction en utilisant le réchauffement et le refroidissement n'est pas important. Afin de suivre l'intuition biochimique, et pour des raisons de simplicité, nous décidons de donner une priorité d'application plus haute à la règle d'évaluation de refroidissement. En conséquence, un monde est réchauffé seulement après un refroidissement du multivers auquel il appartient.

4.5 Vue d'ensemble de la Sémantique du Calcul avec Application Explicite

En Figure 4.2, nous regroupons toutes les règles d'évaluation du calcul où l'application d'une abstraction à une molécule abstraite est rendue explicite :

(Réchauffement)	$[N\ A\ M] \longrightarrow_h [N\ A@M]$
(Application)	$A@M \longrightarrow_a \{[\varsigma_1(R_A)]\ldots[\varsigma_n(R_A)]\}$ si $\mathcal{S}ol(L_A \prec\!\!\prec M) = \{\varsigma_1, \ldots, \varsigma_n\}$
(ApplicationEchec)	$A@M \longrightarrow_{\text{af}} A\ M$ if $\mathcal{S}ol(L_A \prec\!\!\prec M) = \emptyset$
(Refroidissement)	$[N\ \{[M_1]\ldots[M_n]\}] \longrightarrow_c \{[N\ M_1]\ldots[N\ M_n]\}$

Fig. 4.2: Sémantique de $\rho_{\langle\Sigma\rangle}$-calcul avec application explicite

5 Ajouter des Stratégies au Calcul

Le parallélisme intrinsèque de la réécriture sur des sous-termes disjoints en utilisant les abstractions disponibles dans un monde peut être changé en considérant un mécanisme de contrôle ou un ordre d'application des abstraction. Les stratégies de réécriture fournissent un tel contrôle dans un cadre basé sur des règles. Dans cette section, nous définissons les stratégies comme des objets du calcul, en utilisant les constructions de base, comme on peut le faire dans le λ-calcul ou le γ-calcul. Pour une telle définition, nous utilisons une approche similaire à celle de [CKLW03] où les stratégies de réécriture sont encodées dans des règles de réécriture. Les définitions des objets de stratégie sont données avec les preuves de leur correction par rapport à la sémantique des stratégies abstraites qu'elles encodent. Alors, grâce aux stratégies, de nouvelles extensions du calcul sont possibles, comme par exemple l'interception d'un échec dans l'application ou pour définir des stratégies persistantes.

5.1 Des Stratégies sous Formes d'Abstractions

Considérons pour les combinateurs de stratégie donnés en Section 1.7, les objets suivants (alias) : id pour Id, fail pour $Fail$, first pour $First$, seq pour $\_;\_$, not pour Not, ifThenElse pour IfThenElse, try pour Try, repeat pour $Repeat$. L'opérateur d'application pour les stratégies $[\_](\_)$ de la Section 1.7 correspond à la construction @. Les abstractions de $\mathcal{A}$ sont des exemples élémentaires de stratégies et correspondent au système de réduction abstrait sur lequel nous définissons les stratégies.

Dans ce qui suit, lorsque nous souhaitons mettre en relief l'utilisation de stratégies comme abstractions, nous remplaçons le symbole pour l'abstraction par le symbole pour la stratégie S dans chaque règle d'évaluation.

En premier, nous étendons la syntaxe des molécules abstraites avec l'échec ou objet *stuck*, stk. Nous imposons la restriction sur un monde qu'il ne doit contenir aucune abstraction dont l'application peut produire stk. Cet objet d'échec stk est le résultat de l'échec de l'application d'une abstraction à une molécule abstraite.

$$\text{(ApplicationEchec)} \qquad A@M \longrightarrow_{\text{af}} \{[\text{stk}]\} \;\text{ si }\; \mathcal{S}ol(L_A \prec\!\!\prec M) = \emptyset \tag{5.1}$$

Nous appelons une abstraction étendue ou une stratégie une abstraction dont le membre de droite peut contenir le constructeur de l'application.

Soient S, S_1, S_2 des stratégies. Nous encodons les stratégies Id, $Fail$, $\_;\_$, et $First$

sous la forme des alias pour les abstractions étendues suivants respectivement :

$$\begin{aligned}
\mathrm{id} &\triangleq X \Rightarrow X \\
\mathrm{fail} &\triangleq X \Rightarrow \mathrm{stk} \\
\mathrm{seq}(S_1, S_2) &\triangleq X \Rightarrow S_2@(S_1@X) \\
\mathrm{first}(S_1, S_2) &\triangleq X \Rightarrow (S_1@X)\ \ (\mathrm{stk} \Rightarrow (S_2@X))@(S_1@X) \\
\mathrm{not}(S) &\triangleq X \Rightarrow \mathrm{first}(\mathrm{stk} \Rightarrow X, X' \Rightarrow \mathrm{stk})@(S@X) \\
\mathrm{ifThenElse}(S_1, S_2, S_3) &\triangleq X \Rightarrow \mathrm{first}(\mathrm{stk} \Rightarrow S_3@X, X' \Rightarrow S_2@X)@(S_1@X)
\end{aligned}$$

L'échec de l'application d'une stratégie abstraite comme définie en Section 1.7 est représenté par l'ensemble vide. Cependant, nous définissons un échec de l'application d'une stratégie dans le $\rho_{\langle\Sigma\rangle}$-calcul non pas par un ensemble vide, mais par le singleton contenant l'objet échec stk, i.e., une structure avec un monde $\{[\mathrm{stk}]\}$. L'échec explicite fournit une plus grande expressivité aux objets de stratégie dans le $\rho_{\langle\Sigma\rangle}$-calcul. En considérant une stratégie avec l'objet échec comme membre de gauche, nous pouvons attraper et traiter de manière appropriée un échec de l'application par une autre stratégie.

Les objets correspondant à la stratégie abstraite composée *Try* est alors définie aisément à partir des stratégies first et id, comme vu dans le cas général des stratégies abstraites, cf Section 1.7.

$$\mathrm{try}(S) \triangleq \mathrm{first}(S, \mathrm{id})$$

La stratégie composée *Repeat* est définie en utilisant l'opérateur de récursion μ et les objets de stratégie try et seq comme suit :

$$\mathrm{repeat}(S) \triangleq \mu X.\mathrm{try}(\mathrm{seq}(S, X))$$

Nous pouvons encoder l'abstraction μ en utilisant le combinateur de point-fixe du λ-calcul comme cela a été fait pour encoder les itérateurs dans le ρ-calcul [CK01] :

$$\mathrm{repeat}(S) \triangleq \Theta@J(S)$$

où

$$\begin{aligned}
\Theta &\triangleq A@A \\
A &\triangleq X \Rightarrow (Y \Rightarrow Y@((X@X)@Y))) \\
J(S) &\triangleq Y \Rightarrow (X \Rightarrow \mathrm{first}(\mathrm{seq}(S, Y), \mathrm{id})@X)
\end{aligned}$$

Puisque nous considérons tous les résultats possibles d'une application, et que nous pouvons appliquer une stratégie à un monde d'application comme c'est le cas pour la définition de seq, nous avons besoin d'une règle d'évaluation qui distribue l'application d'une stratégie à chaque monde dans la structure :

$$\text{(AppRefroidissement)} \qquad S@\{[M_1] \ldots [M_n]\} \longrightarrow_{\mathrm{ac}} \{[S@M_1] \ldots [S@M_n]\} \tag{5.2}$$

Le nom de la règle provient du fait qu'elle a les caractéristiques d'une règle de refroidissement, en branchant dans deux contextes (ici, un monde d'application) les éléments d'une structure de mondes.

Une seconde règle est nécessaire pour applatir un monde consistant uniquement en un multivers :

$$\text{(Applatissement)} \qquad \{[\{[M_1] \dots [M_n]\}]\} \longrightarrow_f \{[M_1] \dots [M_n]\} \tag{5.3}$$

Si un échec apparaît pendant le procédé d'application, nous le manipulons explicitement. Nous retirons un monde contenant l'objet échec si c'est l'unique monde dans un multivers juxtaposé à d'autres multivers, ou si c'est juxtaposé à un autre monde, se comportant ainsi comme un élément neutre pour la juxtaposition de multivers et mondes. En conséquence, nous étendons la relation de congruence structurelle sur les mondes de la Définition 29 avec les deux axiomes suivants :

$$\begin{aligned} \{[\text{stk}]\}\ W' &\equiv W' \\ [\text{stk}]\ V' &\equiv V' \end{aligned}$$

Par exemple, le premier axiome ci-dessus peut être utilisé pour réduire l'application d'une stratégie $\text{first}(S_1, S_2)$ à un objet O lorqu'au moins une des molécules $S_1@O$ and $(\text{stk} \Rightarrow (S_2@X))@(S_1@X)$ se réduit en l'objet échec stk.

5.2 Appel-par-Nom dans le Calcul avec Stratégies

Nous considérons une stratégie d'évaluation de type appel-par-nom pour le calcul, qui empêche toute réduction à l'intérieur d'une abstraction. Dans ce qui suit, nous motivons le choix d'une telle stratégie d'évaluation.

Nous avons vu que la définition d'une stratégie peut contenir des mondes d'application. Voyons ce qui se passe si une règle d'abstraction (soit (Application), (ApplicationEchec), ou (AppRefroidissement)) est appliquée à l'intérieur d'une abstraction. Considérons par exemple le monde $[X \Rightarrow S_2@S_1@X\ O]$. Alors la règle de refroidissement et la règle d'application se chevauchent comment sur la Figure 5.1.

Si en particulier les membres de gauche de S_1 et S_2 ne sont ni des variables, ni des constantes d'échec stk, et que l'application séquentielle de S_1 suivie de S_2 et O réussie, alors la branche de droite se réduit en quelque chose différent de $\{[\text{stk}]\}$.

Le branchement gauche mène à un résultat non-souhaité obtenu en réduisant trop tôt dans le membre de droite de l'abstraction. Un autre branchement non-souhaité est éliminé dès le début grâce au typage des molécules abstraites : puisque $S_1@X$ n'est pas un objet structuré, la règle d'application ne peut pas être utilisée pour réduire $S_2@S_1@X$. En conclusion, pour une expression $S_2@S_1@M$ la priorité de l'opérateur d'application @ va de droite à gauche, c'est-à-dire nous paramétrons l'expression $S_2@S_1@M$ comme $S_2@(S_1@M)$.

5.3 Correction de l' Encodage des Stratégies sous Forme d'Abstractions

Nous énonçons que notre encodage des stratégies de réécriture sous forme d'abstractions dans le $\rho_{\langle\Sigma\rangle}$-calcul est correct par rapport à la sémantique des stratégies abstraites

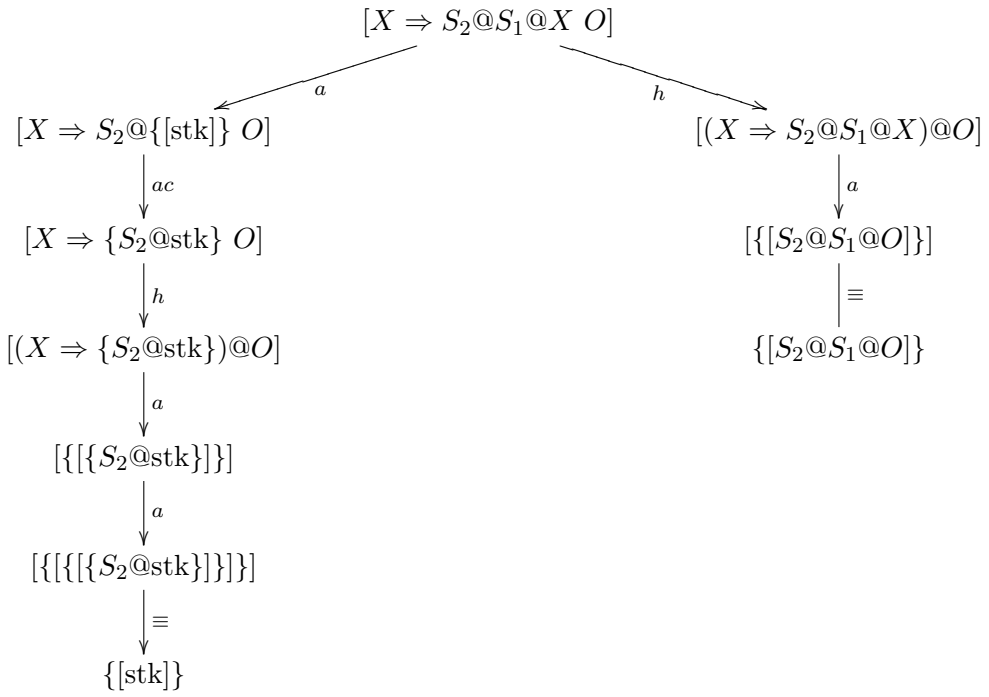


Fig. 5.1: Illustration de la nécessité d'une évaluation de type appel-par-nom dans le calcul

telle que donnée en Section 1.7. Pour une stratégie de réécriture ζ et son encodage S dans le calcul, nous énonçons que l'encodage est correct s'il préserve les solutions. La correction de l'encodage dépend de la correction de la définition de réduction d'une abstraction et d'une molécule abstraite par rapport à la relation de réécriture définie sur les molécules abstraites.

Proposition 1 (Correction de id et fail). Les objets id et fail sont de encodages corrects des stratégies abstraites Id et $Fail$ respectivement.

Proposition 2 (Correction de seq). Si les abstractions étendues S_1 et S_2 sont des encodages corrects des stratégies abstraites ζ_1 et ζ_2 respectivement, alors $\text{seq}(S_1, S_2)$ est un encodage correct de la stratégie abstraite $\text{Sequence}(\zeta_1, \zeta_2)$.

Proposition 3 (Correction de first). Si les abstractions étendues S_1 et S_2 sont des encodages corrects des stratégies abstraites ζ_1 et ζ_2 respectivement, alors $\text{first}(S_1, S_2)$ est un encodage correct de la stratégie abstraite $\text{First}(\zeta_1, \zeta_2)$.

Proposition 4 (Correction de not). Si l'abstraction étendue S est un encodage correct de la stratégie abstraite ζ, alors $\text{not}(S)$ est un encodage correct de la stratégie abstraite $\text{Not}(\zeta)$.

Proposition 5 (Correction de ifThenElse). Si les abstractions étendues S_1, S_2, S_3 sont des encodages corrects des stratégies abstraites $\zeta_1, \zeta_2, \zeta_3$ respectivement, alors ifThenElse(S_1, S_2, S_3) est un encodage correct de la stratégie abstraite IfThenElse$(\zeta_1, \zeta_2, \zeta_3)$.

Remarque 1. La correction de l'encodage de l'opérateur de récursion μ dans le $\rho_{\langle\Sigma\rangle}$-calcul est équivalent à la notion de correction correspondante dans le ρ-calcul. A présent, si S est un encodage correct d'une stratégie abstraite ζ, alors try(S) et repeat(S) sont des encodages corrects de Try(ζ) et Repeat(ζ) respectivement.

Le théorème suivant résume les résultats précédents sur la correction de l'encodage de stratégies.

Théorème 3. Pour chaque objet de stratégie S obtenu en combinant des objets de stratégie primitifs id, fail, seq, first, not, ifThenElse, et pour chaque molécule abstraite M, un calcul réussi de $S@M$ génère une structure de mondes $\{[M_1] \dots [M_n]\}$ telle que chaque molécule abstraite M_i est obtenue à partir de M par réécriture stratégique suivant la stratégie S.

Les preuves des résultats énoncés dans cette section sont disponibles dans la version anglaise de ce document.

5.4 Étendre la Sémantique avec des Stratégies et une Récupération d'Échec

En nous basant sur les définitions de stratégies, nous pouvons reformuler la règle de réduction principale modélisant l'interaction entre une stratégie S et une molécule abstraite M dans un monde en utilisant un mécanisme d'interception d'échec :

$$\text{(RchauffementR)} \quad [N\ S\ M] \longrightarrow_{\text{hr}} [N\ \text{seq}(S, \text{try}(\text{stk} \Rightarrow S\ M))@M] \tag{5.4}$$

Une réduction utilisant la règle (5.4) procède en l'une des étapes suivantes :
- si $S@M$ se réduit en l'objet d'échec stk, alors la stratégie try(stk $\Rightarrow S\ M$) restaure les molécules initiales sujettes à réduction ;
- si $S@M$ réussit, alors l'application de la stratégie try(stk $\Rightarrow S\ M$) ne change pas le résultat.

Nous appelons cette réduction améliorée réchauffement avec récupération et stk $\Rightarrow$ $S\ M$ une règle de récupération. Les deux molécules S et M du membre de droite de la règle de récupération ne peuvent pas être réchauffées avant que la règle ne soit appliquée puisqu'elles sont des arguments de l'objet flèche et non juxtaposées dans un monde. Ainsi, il n'y a aucun risque d'une réduction récursive utilisant cette règle de réchauffement.

Le comportement présenté ci-dessus pour la règle de réchauffement avec récupération est possible sous l'hypothèse que les emplacement de stratégies directement dans un monde ne doivent pas retourner l'objet d'échec stk.

Remarque 2. La règle de réchauffement avec récupération (5.4) est juste un exemple de manipulation d'un échec de filtrage apparaissant dans le procédé d'application. Au lieu

de simplement récupérer l'abstraction interagissante S et la molécule abstraite M, on peut imaginer d'autres façons de manipuler un échec dans une interaction, comme par exemple en modifiant M ou même S.

5.5 Stratégies Persistentes

À ce niveau de définition du calcul, une stratégie est consommée par une interaction réussie avec une molécule abstraite M. Un avantage est que, puisque nous travaillons avec des multi-ensembles, une stratégie peut être multiple, et chaque interaction entre la stratégie et une molécule abstraite M consomme une occurrence de la stratégie. Cela permet de contrôler le nombre maximum de fois qu'une interaction peut avoir lieu.

Quelques fois, il peut être intéressant d'avoir une persistence des stratégies. Dans ce cas, une stratégie ne devrait pas être consommée par la réduction. Dans ce but, nous définissons un combinateur de stratégie persistent qui applique une stratégie donnée en argument, et, en cas de succès, la duplique :

$$S! \triangleq \mu X.\mathrm{seq}(S, \mathrm{first}(\mathrm{stk} \Rightarrow \mathrm{stk}, Y \Rightarrow (Y\ X)))$$

Durant l'application de la stratégie $S!$ à une molécule abstraite M, deux choses peuvent se passer :

- Si $S!@M$ se réduit en $\{[\mathrm{stk}]\}$ alors l'abstraction $\mathrm{stk} \Rightarrow \mathrm{stk}$ est appliquée ; alors la règle de réchauffement avec récupération remplace stk avec la juxtaposition $S!\ M$;
- Si $S!@M$ se réduit à un multivers W alors le second argument de la stratégie first est appliqué, Y est instancié apr W et le résultat final est $S!\ W$; alors la règle de refroidissement distribue la stratégie $S!$ à chaque monde de W.

Comme pour l'encodage de la stratégie *Repeat*, nous utilisons un encodage des itérateurs comme dans le ρ-calcul [CK01] :

$$\begin{array}{rcl} S! & \triangleq & \Theta@K(S) \\ K(S) & \triangleq & Z \Rightarrow (X \Rightarrow \mathrm{seq}(S, \mathrm{first}(\mathrm{stk} \Rightarrow \mathrm{stk}, Y \Rightarrow (Y\ Z))@X) \end{array}$$

À partir de la correction de l'encodage de l'itérateur μ comme dans le ρ-calcul [CK01], le résultat suivant prouve que le combinateur de stratégie persistent est correctement défini :

Proposition 6. Soit M une molécule abstraite et S une stratégie telle que toute abstraction apparaissant dans S n'a pas stk comme membre de droite. Si $S@M \longrightarrow^* \{[\mathrm{stk}]\}$, alors $S!\ M \longrightarrow_{hr} \longrightarrow^* \{[S!\ M]\}$, sinon, si $S@M \longrightarrow^* \{[M_1] \ldots [M_n]\}$, $n \geq 1$, i.e., l'application n'échoue pas, alors $S!\ M \longrightarrow_{hr} \longrightarrow^* \{[S!\ M_1] \ldots [S!\ M_n]\}$.

Remarque 3. Si nous considérons que application réussie d'une stratégie à une molécule abstraite ne consomme pas la stratégie, alors les stratégies sont persistentes par définition, et nous n'avons plus besoin de définir le combinateur persistent pour les stratégies. Cependant, l'approche actuelle nous donne la liberté de fournir des stratégies avec un comportement persistent, alors que d'autres existent dans un nombre limité d'occurrences, ainsi elles sont consommées par des applications.

5.6 Vue d'ensemble de la Syntaxe et de la Sémantique du Calcul avec Stratégies

En Figure 5.2, nous donnons une syntaxe du calcul avec stratégies et en Figure 5.3, nous revisitons les règles d'évaluation du calcul où une abstraction est généralisées à une stratégie.

(Objets structurés)	$\mathcal{O}$		
	$\mathcal{A}_0$	$::=$	$\mathcal{O} \Rightarrow \mathcal{O} \mid (\mathcal{O} \Rightarrow \mathcal{O}) \Rightarrow (\mathcal{O} \Rightarrow \mathcal{O})$
	$\mathcal{M}_0$	$::=$	$\mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{M}_0\ \mathcal{M}_0$
(Abstractions)	$\mathcal{A}$	$::=$	$\mathcal{A}_0 \mid \mathcal{O} \Rightarrow \mathcal{M}_0$
(Molécules abstraites)	$\mathcal{M}$	$::=$	$\mathcal{X} \mid \mathcal{M}_0 \mid \mathcal{S} \mid \mathcal{M}\ \mathcal{M} \mid \mathrm{stk}$
(Stratégies)	$\mathcal{S}$	$::=$	$\mathcal{A}$
		$\mid$	$\mathrm{stk} \Rightarrow \mathrm{stk} \mid \mathrm{stk} \Rightarrow \mathcal{M}$
		$\mid$	$\mathcal{X} \Rightarrow \mathrm{stk} \mid \mathcal{X} \Rightarrow \mathcal{M}$
		$\mid$	$\mathcal{X} \Rightarrow \mathcal{V}_{app} \mid \mathcal{X} \Rightarrow \mathcal{V}_{app}\ \mathcal{V}_{app}$
(Mondes d'application)	$\mathcal{V}_{app}$	$::=$	$\mathcal{S}@\mathcal{M} \mid \mathcal{S}@\mathcal{V}_{app} \mid \mathcal{S}@\mathcal{W}$
(Mondes)	$\mathcal{V}$	$::=$	$\mathcal{Y} \mid [\mathcal{M}] \mid [\mathcal{M}\ \mathcal{V}_{app}] \mid [\mathcal{M}\ \mathcal{W}]$
(Multivers)	$\mathcal{W}$	$::=$	$\mathcal{Z} \mid \{\mathcal{V} \dots \mathcal{V}\} \mid \{\mathcal{W} \dots \mathcal{W}\}$

Fig. 5.2: Syntaxe du $\rho_{\langle\Sigma\rangle}$-calcul avec stratégies

(RéchauffementR)	$[N\ S\ M] \longrightarrow_{\mathrm{hr}} [N\ \mathrm{seq}(S, \mathrm{try}(\mathrm{stk} \Rightarrow S\ M))@M]$
(Application)	$S@M \longrightarrow_a \{[\varsigma_1(R_S)] \dots [\varsigma_n(R_S)]\}$ si $\mathcal{S}ol(L_S \prec\!\!\prec M) = \{\varsigma_1, \dots, \varsigma_n\}$
(ApplicationEchec)	$S@M \longrightarrow_{\mathrm{af}} \{[\mathrm{stk}]\}$ si $\mathcal{S}ol(L_S \prec\!\!\prec M) = \emptyset$
(Refroidissement)	$[M\ \{[M_1] \dots [M_n]\}] \longrightarrow_c \{[M\ M_1] \dots [M\ M_n]\}$
(AppRefroidissement)	$S@\{[M_1] \dots [M_n]\} \longrightarrow_{\mathrm{ac}} \{[S@M_1] \dots [S@M_n]\}$
(Applatissement)	$\{[\{[M_1] \dots [M_n]\}]\} \longrightarrow_f \{[M_1] \dots [M_n]\}$

Fig. 5.3: Sémantique du $\rho_{\langle\Sigma\rangle}$-calcul avec stratégies

6 Rélation de Reduction à Gros Grain

Dans cette section, nous définissons une nouvelle relation de réduction basée sur ma réduction à grain plus fin définie en Figure 5.3. Cette relation consiste en une étape de réchauffement, suivi par des applications et d'autres opérations pour réarranger la structure, et finalement une étape de refroidissement. À ce niveau de gros grain, nous ne nous concentrons plus sur le processus interne de l'interaction entre une stratégie et une molécule, mais sur son résultat final. Nous utiliserons une telle relation de réduction à des fins de vérification au Chapitre 14.

Définition 30 (Étape d'évolution). Une étape d'évolution d'un monde $[N\ S\ M]$ est une réduction correspondant à la composition séquentielle de trois étapes comme suit :

1. la règle de réchauffement (RéchauffementR)

$$[N\ S\ M] \longrightarrow_{\mathrm{hr}} [N\ \mathrm{seq}(S, \mathrm{try}(\mathrm{stk} \Rightarrow S\ M))@M]$$

2. l'union des règles d'application (Application) et (ApplicationEchec), la règle de distribution (AppRefroidissement), la règle d'applatissement (Applatissement), et modulo la relation de congruence structurelle sur les mondes données en Définition 29 et étendue pour l'objet d'échec stk :

$$[N\ \mathrm{seq}(S, \mathrm{try}(\mathrm{stk} \Rightarrow S\ M))@M] \longrightarrow^* [N\ \{[M_1] \dots [M_n]\}]$$

3. la règle de refroidissement (Refroidissement) :

$$[N\ \{[M_1] \dots [M_n]\}] \longrightarrow \{[N\ M_1] \dots [N\ M_n]\}$$

Une étape d'évolution est silencieuse (ou non-observable), dénotée par $\xrightarrow{\tau}$, si le règle de récupération $\mathrm{stk} \Rightarrow S\ M$ est appliquée durant la seconde étape. En d'autres termes, la stratégie S choisie pour être appliquées à M échoue. Une étape d'évolution est visible si elle n'est pas silencieuse.

Définition 31 (Mondes inertes et stables). Nous disons qu'un monde V est inerte si soit :
- $\exists\ O_1, \dots, O_m \in \mathcal{O}$ tel que $V \equiv [O_1 \dots O_m]$ ou
- $\exists\ S_1, \dots, S_n \in \mathcal{S}$ du même ordre tel que $V \equiv [S_1 \dots S_n]$.

Nous disons qu'un monde $V = [S_1 \dots S_n\ O_1 \dots O_m]$ est stable s'il peut évoluer uniquement via des étapes silencieuses.

Un multivers est inerte (resp. stable) si chaque monde composite est inerte (resp. stable).

En d'autres termes, un monde est inerte s'il consiste uniquement en un multi-ensemble d'objets structurés de $\mathcal{O}$ ou en un multi-ensemble de stratégies non-interagissante, et il est stable si toutes les molécules abstraites sont irréductibles par rapport au reste des stratégies de ce monde.

Définition 32 (Réduction en une étape à gros grain d'un monde). Une réduction en une étape à gros grain d'un monde est induite par un nombre fini d'étapes d'évolution silencieuses et une étape d'évolution visible. Nous la notons $\Rightarrow$ ou $\overset{S}{\Rightarrow}$ pour $(\overset{\tau}{\longrightarrow})^* \longrightarrow$ et $(\overset{\tau}{\longrightarrow})^* \longrightarrow_S$ respectivement si la stratégie appliquée réussie est pertinente.

En supprimant les étapes silencieuses, nous obtenons un modèle équivalent mais plus petit en taille. Le choix de la stratégie et de la molécule abstraite pour le réchauffement devrait être juste [CGP00] dans le sens où si une stratégie est choisie infiniment souvent pour l'application, alors les autres stratégies doivent également être choisies infiniment souvent. Si un monde contient plus d'une stratégie, nous ne voulons pas qu'une seule stratégie soit appliquée encore et encore, puisqu'en particulier, si elle n'est pas applicable, le système ne va jamais évoluer.

Informellement, la relation de réduction en une étape à gros grain entre une stratégie S et une molécule abstraite M énonce que :

$$[S\ M] \overset{S}{\Rightarrow} \{[M_1] \dots [M_n]\} \quad \text{si} \quad [S](M) = \{M_1, \dots, M_n\}, n \geq 1$$

Nous avons besoin d'un niveau de réduction à gros grain dans le $\rho_{\langle\Sigma\rangle}$-calcul instancié pour les graphes à ports afin de vérifier en Chapitre 14 la satisfiabilité d'une formule temporelle seulement après le refroidissement et avant le réchauffement d'un monde.

La réduction en une étape à gros grain d'un multivers correspond à une réduction en une étape à gros grain d'un nombre maximum de mondes tel qu'au moins un monde est réductible dans une étape à gros grain :

Définition 33 (Réduction en une étape à gros grain d'un multivers). Nous disons qu'un multivers $\{V_1 \dots V_n\}$ se réduit de manière synchrone en une étape à gros grain dans un multivers W, ce qui s'écrit $\{V_1 \dots V_n\} \Rightarrow W$, s'il existe une partition $\{i_1, \dots, i_p\} \cup \{i_{p+1}, \dots, i_n\}$ de $\{1, \dots, n\}$, avec $1 \leq p \leq n$, où :
- $V_{i_k} \Rightarrow W_{i_k}$, pour tout k, $1 \leq k \leq p$, et
- V_{i_l} est stable, pour tout l, $p+1 \leq l \leq n$,

telle que $W \equiv \{W_{i_1} \dots W_{i_p} V_{i_{p+1}} \dots V_{i_n}\}$.

7 Stratégies Possibles pour le Calcul

Une extension possible du calcul est de bénéficier de la nature concurrente intrinsèque à la réécriture [Mes92]. Dans cette optique, toutes les interactions possibles peuvent prendre place en parallèle dans un monde modélisant le déplacement Brownien – un aspect essentiel du modèle chimique. Une stratégie parallèle devrait sélectionner plusieurs abstractions et les appliquer sur une molécule abstraite si leurs membres de gauche filtrent des parties disjointes de la molécule. Alors, pour chaque structure particulière Σ, on doit définir le caractère disjoint de deux sous-molécules dans une molécule abstraite.

En ajoutant plus de contrôle au moyen de stratégies pour grouper l'application d'abstractions, nous pouvons formaliser d'autres types de parallélisme. Une stratégie intéressante pour la régulation du parallélisme lorsqu'on modélise des systèmes réels et biologiques et particulier, concerne la dimension stochastique du choix des abstractions à appliquer. Une description formelle de l'application stochastique de règles de réécriture (abstractions) peut être trouvée dans [Spi06].

Nous obtenons une application séquentielle d'abstractions en utilisant par exemple la stratégie don't care du langage ELAN [BKKR01a] : celle-ci choisit à partir d'un ensemble de stratégies données, une stratégie n'échouant pas pour l'état actuel du système ; si elle regroupe toutes les stratégies à partir de l'état, alors l'évolution du système est séquentiel puisqu'une seule stratégie est appliquée à chaque étape d'évolution. Dans [SMC+08] une stratégie séquentielle stochastique est définie pour P systèmes basée sur l'algorithme de simulation stochastique de Gillespie [Gil77] où les règles sont assignées des probabilités qui peuvent changer durant la simulation. En outre, les concepts de réécriture probabiliste et les stratégies probabilistes ont été étudiées dans [BK02, BG06].

D'autres stratégies parallèles utilisées dans les modèles de calcul basés sur les règles et inspirés par la biologie sont par exemple ceux inspirés par les mécanismes de contrôle disponibles pour les P systèmes, les modèles du calcul de membrane [Pau02]. Dans [IYD05], les auteurs revoient certaines notions du parallélisme pour les P systèmes. En termes du calcul que nous avons présenté dans cette partie, pour k le nombre de stratégies applicables aux objets structurés dans un monde et $n \leq k$, les réductions parallèles suivantes peuvent être considérées :

n-Max-Parallèle spécifie qu'un ensemble maximal d'au plus n stratégies est appliqué en une étape et qu'aucun autre sous-ensemble plus grand n'est applicable ;

$\leq n$-Parallèle spécifie que tout sous-ensemble d'au plus n stratégies est appliqué ;

n-Parallèle spécifie que tout sous-ensemble d'exactement n stratégies est appliqué (aussi appelé parallélisme limité).

8 Comparaison avec le γ-Calcul et HOCL

Dans cette section, nous comparons l'expressivité du calcul biochimique abstrait par rapport au γ-calcul, et à HOCL qui le généralise.

L'aspect principal de cette généralisation concerne l'utilisation de règles de réécriture sur des objets structurés dans le style ρ-calcul à la place d'abstractions comme dans le λ-calcul. HOCL considère une large classe de motifs qui sont des renommages adaptés de λ-expressions souvent complexes. Cependant, nous gagnons en puissance expressive dans le $\rho_{\langle\Sigma\rangle}$-calcul en considérant une classe plus abstraite de motifs.

Dans le γ-calcul, les conditions sur les réductions dues à l'encapsulation de solutions impose une stratégie d'application en profondeur. Le $\rho_{\langle\Sigma\rangle}$-calcul n'a pas de constructeur d'encapsulation pour les molécules abstraites, mais ce manque syntaxique est privilégié par rapport à l'ajout de stratégies dans le calcul. Cependant, la réduction dans le $\rho_{\langle\Sigma\rangle}$-calcul utilise une stratégie d'évaluation, l'évaluation d'appel-par-nom, qui interdit la réduction sous un opérateur flèche dans une abstraction. L'utilisation de motifs structurés enrichit le $\rho_{\langle\Sigma\rangle}$-calcul avec la capacité d'encoder des stratégies sous forme d'abstractions. Ceci ouvre la voie à la définition de diverses stratégies pour contrôler l'évolution d'un état.

Dans une première étape, HOCL étend le γ-calcul avec deux structures d'ordres supérieur : les conditions sur les réactions et la capture atomique (plusieurs arguments pour une γ-abstraction). L'utilisation de stratégies permet la définition d'abstractions avec des conditions Booléennes puisque nous pouvons définir la relation de congruence sur les Booléens. Nous n'avons pas formalisé les abstractions conditionnelles ici, mais l'approche est similaire à celle du ρ-calcul pour l'encodage de réécritures conditionnelles [CK01] où une règle de réécriture conditionnelle $l \twoheadrightarrow r$ si c est représentée par le ρ-term $l \twoheadrightarrow (\text{True} \twoheadrightarrow r)c_\rho$ où True est une constante et c_ρ le ρ-term décrivant la réduction du terme c en une constante Booléenne, soit True, soit False.

Nous remarquons également que les stratégies persistentes correspondent aux règles de réaction à usage multiple du γ-calcul, qui sont des règles de réaction non consommées par une réduction.

Nous n'avons pas considéré l'extension de HOCL avec multiciplicité négative ou infinie pour les molécules [BFR06b, Rad07], mais cela est aussi possible pour le $\rho_{\langle\Sigma\rangle}$-calcul en utilisant une approche similaire.

9 Conclusion

Dans cette partie, nous avons introduit un calcul d'ordre supérieur basé sur la métaphore chimique classique. En permettant des motifs complexes pour les réactions et une structure riche pour décrire les molécules et les connexions possibles entre elles, nous fournissons un calcul avec une coloration biologique additionnelle.

Cette partie présente les fondations de la thèse. Dans les chapitres suivants, nous allons instancier la structure Σ avec la structure de graphes avec ports et la transformation de molécules structures avec la relation de réécriture de graphes avec ports telle que définie en Chapitre 10. Les caractéristiques principales du calcul résultant de ce travail sont donnés dans le Chapitre 11 avec la preuve que les graphes avec ports représentent une structure unifiante pour représenter des règles de réécriture de graphes avec ports et pour décrire opérationnellement l'application d'une règle de réécriture de graphes avec ports à un graphe avec ports. Ensuite, en Chapitre 14, le calcul biochimique sur les graphes avec ports est enrichi avec des fonctionnalités de vérification, une approche qui pourrait être utilisée également pour d'autres structures.

Travaux futurs

Pour conclure cette partie, nous aimerions pointer quelques directions en lien avec le travail présenté ici.

- Une direction de recherche intéressante est de voir comment les structures topologiques considérées dans MGS [GM02a, Spi06] peuvent s'intégrer au calcul présenté ici. Le caractère utile des structures topologiques et de leurs transformations pour modéliser des systèmes avec une structure dynamique a été démontré [Gia03] (pour les systèmes de membrane par exemple, voir [GM02b], et pour un exemple biologique réel, voir [SM05]).
- Dans la sémantique du $\rho_{\langle\Sigma\rangle}$-calcul une abstraction est appliquée localement à un objet représentant la structure physique de l'état du système modélisé puisque nous résolvons une équation de filtrage entre le membre de gauche d'une abstraction et une partie de molécule abstraite. Mais que se passe-t-il si nous voulons qu'une abstraction soit appliquée globalement sur une molécule abstraite ? Par exemple, si nous voulons qu'une règle trouve un motif M, le conserve, et supprime tous les autres objets du monde. Si nous considérons l'abstraction $MN \Rightarrow M$, elle va fournir tous les résultats possibles du filtrage de N avec les molécules du monde qui ne filtrent pas M. Bien-sûr, cela fournira le résultat escompté, mais aussi certaines solutions correspondant à des contextes partiels. Nous pourrions traiter ce problème en considérant deux types d'abstraction, les globales et les locales, alors que, au niveau sémantique, les règles d'évaluation décrivant leurs applications vont considérer des solutions d'une

équation de filtrage sur le système entier ou sur un sous-système respectivement. Alors l'exemple de l'abstraction ci-dessus devrait être globale. Il serait intéressant d'analyser les implications de la différentiation entre les abstractions.

- L'approche actuelle pour modéliser les interactions entre une abstraction A et une molécule abstraite M consiste à les réchauffer, et seulement ensuite à tester si A est applicable à M (i.e., si le membre de gauche de A filtre vers un sous-motif de M), et dans le cas positif, de l'appliquer. Une autre approche potentiellement intéressante est de déplacer le test d'applicabilité d'une abstraction dans la règle de réchauffement ; alors la règle d'application n'échouera jamais. Cependant, de tels tests d'applicabilité d'une abstraction à une molécule abstraite ne marche plus comme souhaité pour les stratégies puisque le membre de gauche d'une stratégie peut être une variable et le problème de sous-filtrage a toujours une solution. Alors, le problème qui devrait être résolu est de déterminer les conditions d'application pour différents constructeurs de stratégie.
- Dans Chapitre 7, nous avons mentionné quelques pistes d'évolution parallèle pour les systèmes. Nous devrions étudier plus avant comment celles-ci peuvent être exprimées comme stratégies de réécriture, et si cela est possible, comment les encoder comme stratégies dans le $\rho_{\langle\Sigma\rangle}$-calcul.

Deuxième partie

Une Instance du Calcul Biochimique Abstrait Basée sur des Graphes

10 Réécriture de Graphes avec Ports

Les graphes sont des structures de données (ou types de données abstraits) utilisés intensivement pour décrire des structures complexes telles que diagrammes UML, architecture de microprocesseurs, documents XML, réseaux de communication, flux de données, réseaux de neurones, systèmes biologiques, etc., de façon directe et intuitive. Pour les systèmes complexes, en plus de fournir une description statique, des transformations de graphes permettent de modéliser leur évolution dynamique de manière uniforme. Le calcul par transformation de graphes n'est pas limité à la programmation, spécification ou implantation, c'est un concept fondamental également pour la concurrence et la distribution, la modélisation visuelle et la transformation de modèles, et les modèles calculatoires en général [EP05]. Des approches différentes ont été proposées pour formaliser les transformations de graphes et leurs applications [Roz97, EEKR97, EKMR97].

Dans ce chapitre, nous étudions une classe particulière de graphes, où les nœuds ont des sites de connexion explicites appelés ports, et où les arcs sont attachés à des ports spécifiques des nœuds. Cette structure de graphes est inspirée par les complexes moléculaires formés dans les interactions protéine-protéine dans un réseau biochimique : une protéine est caractérisée par une collection de petites rustines à sa surface, appelées domaines ou sites fonctionnels, utilisées pour se connecter à une autre protéine. Ainsi, la protéine avec sa collection de sites est modélisée par un graphe avec ports. L'objectif de ce chapitre est de définir formellement les graphes avec ports, le sous-filtrage de graphes avec ports, les règles de réécriture de graphes avec ports et la relation de réécriture de graphes avec ports tels que nous pouvons instancier le Calcul Biochimique Abstrait introduit dans Partie I. Le résultat est un calcul sur les graphes avec ports présenté dans Chapitre 11 avec des applications en modélisation de systèmes autonomes et de réseaux biochimiques.

Nous commençons par définir les graphes avec ports en Section 10.1, puis nous continuons avec la définition d'un morphisme de graphes avec ports et les définitions inhérentes de composition et de sous-graphes (Section 10.2). Nous introduisons également un morphisme de graphes avec ports particulier défini uniquement sur les nœuds qui vont être utilisés pour encoder la correspondance entre éléments du membre de gauche et du membre de droite d'une règle de réécriture de graphes avec ports. En Section 10.3, nous définissons les équations de filtrage et sous-filtrage pour graphes avec ports et nous fournissons un algorithme de sous-filtrage correct et complet. Dans les deux sections suivantes, 10.4 et 10.5, nous donnons la définition de règles de réécriture de graphes avec ports et définissons une relation de réécriture sur les graphes avec ports. Alors, en Section 10.6, nous montrons brièvement comment obtenir une relation de réécriture de graphes avec ports stratégique en instanciant les définitions des systèmes de réduction abstraits et les stratégies abstraites du Chapitre 1. Pour les situations où nous pouvons

ou avons besoin de spécifier une information partielle sur les ports des nœuds dans une règle de réécriture de graphe avec ports, nous définissons les graphes avec ports faibles en Section 10.7, et adaptons la relation de réécriture de graphes avec ports définie plus tôt. En Section 10.8, nous étudions la confluence de la relation de réécriture de graphes avec ports en nous basant sur des résultats connus pour les hypergraphes. Nous terminons avec une comparaison avec les systèmes réactifs bigraphicaux (Section 10.9).

10.1 Graphes avec Ports

Nous commençons par définir comment les ports sont associés à des noms de nœuds au moyen d'une signature.

Définition 34 (P-Signature). Une p-signature est une paire d'ensembles de noms $\nabla = \langle \nabla_{\mathscr{N}}, \nabla_{\mathscr{P}} \rangle$ où :
- $\nabla_{\mathscr{N}}$ est un ensemble de noms de nœuds,
- $\nabla_{\mathscr{P}}$ est un ensemble de noms de ports,

tels que chaque noms de nœud N est associés à un ensemble fini de ports $\text{Interface}(N) \subseteq \nabla_{\mathscr{P}}$.

Notons que, par définition, les noms de ports associés à un nom de nœud sont distincts deux à deux.

Dans ce qui suit, les symboles $a, b, c, \ldots$ décrivent l'ensemble $\nabla_{\mathscr{P}}$ des constantes de nom de port, $A, B, C, \ldots$ l'ensemble $\nabla_{\mathscr{N}}$ des constantes de nom de nœud, $x, y, z, \ldots$ l'ensemble $\mathcal{X}_{\mathscr{P}}$ des variables de nom de port, et $X, Y, Z, \ldots$ l'ensemble $\mathcal{X}_{\mathscr{N}}$ des variables de nom de nœud. Tous les symboles peuvent être indicés. Nous notons par $\nabla^{\mathcal{X}}$ la p-signature associant à un nom de nœud de $\nabla_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{N}}$, une interface (ensemble fini de noms de port) de $\nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}}$. Nous étendons la définition d'Interface d'une variable de nom de nœud de telle sorte que $\text{Interface}(X) \subseteq \nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}}$ pour tout $X \in \mathcal{X}_{\mathscr{N}}$.

Définition 35 (Graphe avec port). Étant donnée une p-signature $\nabla^{\mathcal{X}}$, un graphe avec port étiqueté sur $\nabla^{\mathcal{X}}$ est un n-uplet $G = \langle V_G, E_G, lv_G, le_G \rangle$ où :
- V_G est un ensemble fini de nœuds,
- E_G est un multi-ensemble fini d'arêtes,
  $E_G = \{\langle (v_1, p_1), (v_2, p_2) \rangle \mid v_i \in V_G, p_i \in \text{Interface}(lv_G(v_i)) \cup \mathcal{X}_{\mathscr{P}}\}$;
- $lv_G : V_G \to \nabla_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{N}}$ est la fonction d'étiquetage pour les nœuds,
- $le_G : E_G \to (\nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}}) \times (\nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}})$ est la fonction d'étiquetage pour les arêtes telle que $le_G(\langle (v_1, p_1), (v_2, p_2) \rangle) = (p_1, p_2)$.

Nous représentons les nœuds par des identifiants uniques qui sont des mots non-vides utilisant des entiers et symboles littéraux $\{i, j, k, \ldots\}$. Par exemple, $i.j.1$, 2, 1.3 sont trois identifiants de nœud. Les identifiants doivent être uniques car nous autorisons différents nœuds à avoir le même nom. Ainsi, l'ensemble des nœuds est donné sous la forme d'un ensemble d'identifiants uniques, et chaque nœud a un type associé décrit par un nom n de $\nabla_{\mathscr{N}}$ et un ensemble de ports de $\text{Interface}(n) \cup \mathcal{X}_{\mathscr{P}}$.

Nous notons (i) ID(G) l'ensemble des symboles littéraux apparaissant dans les identifiants de nœud de G, et (ii) $\mathcal{V}ar(G)$ l'ensemble ID(G) auquel nous ajoutons l'ensemble des variables de nom apparaissant dans G.

En Figure 10.1, nous illustrons deux vues d'un graphe avec ports : sur la gauche, nous utilisons la représentation classique d'un multi-graphe étiqueté, alors que sur la droite, nous mettons les ports en relief. Nous utiliserons cette dernière représentation, car plus suggestive, pour les graphes avec ports.

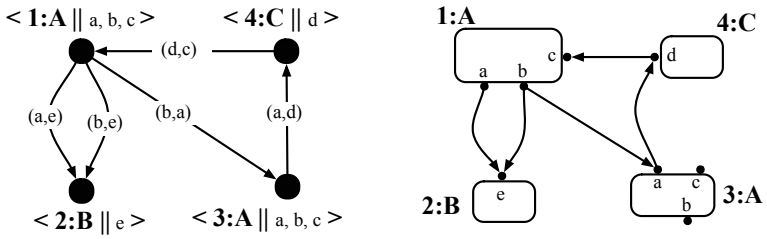


Fig. 10.1: Deux vues d'un graphe avec ports.

10.2 Morphismes de Graphes avec Ports et Morphismes de Nœuds

Dans cette section, nous introduisons les concepts liés au graphes avec ports nécessaires à la définition de transformations sur les graphes avec ports. Nous commençons par définir les morphismes de graphes avec ports utilisés pour mettre en relation deux graphes avec ports dans le but de définir la relation de sous-filtrage. Nous étudions également quelques instances particulières de tels morphismes.

Un morphisme de graphes avec ports lie les éléments de deux graphes avec ports en préservant les sources et cibles des arêtes, les noms de nœud constants et les interfaces associées à renommage de variable près.

Définition 36 (Morphisme de graphes avec ports). Étant donnés deux graphes avec ports G et H sur $\nabla^{\mathcal{X}}$ avec $\mathcal{V}ar(H) \subseteq \mathcal{V}ar(G)$, un morphisme de graphes avec ports $f : G \to H$ est une paire de fonctions $f = \langle f_V : V_G \to V_H, f_E : E_G \to E_H \rangle$ qui, pour un renommage de variables ψ, satisfait les conditions suivantes :

- pour chaque $v \in V_G$, $lv_G(v) = (f_V \circ lv_H)(v)$ si $lv_G(v) \notin \mathcal{X}_{\mathcal{N}}$ et $(f_V \circ lv_h)(v) \notin \mathcal{X}_{\mathcal{N}}$;
- pour chaque $v \in V_G$, si Interface$(lv_G(v)) = \{p_1, \ldots, p_i, x_1, \ldots, x_j, y_1, \ldots, y_l\}$ avec $i, j, l \geq 0$, alors Interface$((f_V \circ lv_H)(v)) = \{p_1, \ldots, p_{i+j}, z_1, \ldots, z_l\}$ avec $0 \leq k \leq j$ tel que $\{y_1, \ldots, y_l\} =_\psi \{z_1, \ldots, z_l\}$
- pour chaque $e \in E_G$, si $le_G(e) = \langle (v_1, p_1), (v_2, p_2) \rangle$, alors $(f_E \circ le_H)(e) = \langle (f_V(v_1), p'_1), (f_V(v_2), p'_2) \rangle$ avec $p'_i \in$ Interface$(f_V(v_i))$, tel que $p_i =_\psi p'_i$.

Définition 37 (Sous-graphe avec ports). Un morphisme de graphes avec ports $f : G \to H$ est une inclusion (également notée $f : G \hookrightarrow H$) si $f_V(v) = v$ et $f_E(e) = e$ pour tout

$v \in V_G$ et $e \in E_G$ à un renommage de variables près. Alors G est un sous-graphe avec ports de H dénoté par $G \subseteq H$.

Nous disons qu'un sous-graphe avec ports G de H est un sous-graphe avec ports complet si pour toute paire de ports de G connectés par un multi-ensemble d'arêtes E' dans H, le multi-ensemble E' apparaît également dans G.

Définition 38 (Sous-graphe avec ports induit). Étant donné un morphisme de graphes avec ports $f : G \to H$, $f(G)$ est appelé le sous-graphes avec ports induit de H.

Définition 39 (Composition de morphismes de graphes avec ports). Soient $f : G \to H$ et $g : H \to I$ deux morphismes de graphes avec ports. Alors leur composition $h : G \to I$ habituellement dénotée par $g \circ f$ (ou $f; g$) est la paire de fonctions $h = \langle h_V : V_G \to V_I, h_E : E_G \to E_I \rangle$ définies par $h_V = g_V \circ f_V$ et $h_E = g_E \circ f_E$.

Comme dans le cas des morphismes de graphes classiques, l'opération de composition sur les morphismes de graphes avec ports est associative, avec comme élément neutre le morphisme identité.

Définition 40 (Morphismes de graphes avec ports particuliers). Soit f un morphisme de graphes avec ports, $f = \langle f_V, f_E \rangle : G \to H$. Nous disons que f est un monomorphisme (ou mono ou encore morphisme injectif) si f_V et f_E sont tous les deux injectifs. Nous disons que f est un épimorphisme (ou épi ou encore morphisme surjectif) si f_V et f_E sont tous deux surjectifs.

Si f est mono et épi, i.e., f_V et f_E sont tous deux bijectifs, alors f est un isomorphisme (ou iso) et nous écrivons $G \cong H$; en outre, nous dénotons par $[G] = \{H \mid H \cong G\}$ la classe d'isomorphismes de G.

Dans ce qui suit, nous introduisons un morphisme de graphes particulier qui associe à un nœud d'un graphe un ensemble de nœuds d'un autre graphe.

Définition 41 (Morphisme de nœuds). Étant donnés deux graphes avec ports G et H sur $\nabla^{\mathcal{X}}$ avec $\mathcal{V}ar(H) \subseteq \mathcal{V}ar(G)$, un morphisme de nœuds est une fonction $\xi : V_G \to \mathcal{P}(V_H)$ et nous la dénotons par $\xi : G \to H$.

Si $\xi_1 : G \to H$ et $\xi_2 : H \to I$ sont deux morphismes de nœuds, alors leur composition est un morphisme de nœuds $\xi_2 \circ \xi_1 : G \to I$ défini comme suit :

$$(\xi_2 \circ \xi_1)(a) = \bigcup_{b \in \xi_1(a)} \xi_2(b), \ \forall a \in V_G$$

Par convention, si $\xi_1(a) = \emptyset$, alors $(\xi_2 \circ \xi_1)(a) = \emptyset$.

Le morphisme de nœuds identité $\xi : G \to G$ associe à un nœud le singleton contenant le même nœud : $\xi(v) = \{v\}$, $\forall v \in V_G$.

Nous étendons un morphisme de nœud $\xi : V_G \to \mathcal{P}(V_H)$ sur les arêtes de G, en définissant $\xi : E_G \to \mathcal{P}(E_H)$ comme suit :
pour $e \in E_G$ avec $e = \langle (v, p), (u, r) \rangle$,

- si $\xi(v) = \{v_1, \ldots, v_k\}$ et $\xi(u) = \{u_1, \ldots, u_n\}$, $k, n \geq 1$, alors
  $\xi(e) = \{\langle (v_i, p), (u_j, r) \rangle \mid p \in \text{Interface}(v_i), r \in \text{Interface}(u_j)\}$;

– si $\xi(v) = \emptyset$ ou $\xi(u) = \emptyset$ alors $\xi(e) = \emptyset$.

Nous utilisons les notations $G \xrightarrow{f} H$ et $G \overset{\xi}{\Rightarrow} H$ pour un morphisme de graphes avec ports $f : G \to H$ et un morphisme de nœuds $\xi : G \to H$ respectivement.

Remarque 4. Un morphisme de graphes avec ports $g : G \to H$ peut être vu comme un morphisme de nœuds qui associe à chaque élément x de G le singleton $\{g(x)\}$. La composition d'un morphisme de nœud avec un morphisme de graphes avec ports $G \xrightarrow{g} H \overset{\xi}{\Rightarrow} I$ est un morphisme de nœuds $G \overset{\xi \circ g}{\Longrightarrow} I$ défini comme suit :

$$(\xi \circ g)(v) = \xi(g(v)),\ \forall v \in V_G$$
$$(\xi \circ g)(e) = \xi(g(e)),\ \forall e \in E_G$$

Inversement, la composition d'un morphisme de graphes avec ports avec un morphisme de nœuds $G \overset{\xi}{\Rightarrow} H \xrightarrow{g} I$ est également un morphisme de nœuds $G \overset{g \circ \xi}{\Longrightarrow} I$ défini comme suit :

$$\forall v \in V_G, (g \circ \xi)(v) = \begin{cases} \{g(v_1), \ldots, g(v_n)\} & , \text{ si } \xi(v) = \{v_1, \ldots, v_n\},\ n \geq 1 \\ \emptyset & , \text{ si } \xi(v) = \emptyset \end{cases}$$
$$\forall e \in E_G, (g \circ \xi)(e) = \begin{cases} \{g(e_1), \ldots, g(e_m)\} & , \text{ si } \xi(e) = \{e_1, \ldots, e_m\},\ m \geq 1, \\ \emptyset & , \text{ si } \xi(e) = \emptyset \end{cases}$$

10.3 Filtrage

10.3.1 Définition Générale

Soient L et G deux graphes avec ports sur la même p-signature $\nabla^{\mathcal{X}}$ tels qu'ils ne partagent aucun identifiant de nœud, variable de nom de nœud ou variable de nom de port.

Définition 42 (Sous-filtrage). Nous disons que le problème de sous-filtrage strict entre L et G a des solutions s'il existe une décomposition de G de la forme $g(L) \cup G^- \cup \mathcal{B}$ où :
- $g : L \to G$ est un morphisme de graphes avec ports appelé morphisme sous-filtrant tel que le motif $g(L)$ est un sous-graphe complet de G ;
- G^- est un graphe avec ports appelé contexte, avec $G^- = G \setminus g(L)$, donné par l'ensemble des nœuds $V_{G^-} = V_G \setminus V_{g(L)}$, et l'ensemble des arêtes $E_{G^-} = \{\langle (u,p),(v,r)\rangle \in E_G \mid u, v \in V_{G^-}\}$;
- $\mathcal{B}$ est un ensemble d'arêtes (appelés ponts) qui ont une extrémité dans le graphe de contexte et l'autre dans le sous-graphe filtré.

Nous écrivons alors $G = G^- \lfloor g(L) \rfloor_{\mathcal{B}}$.

Nous notons le problème de sous-filtrage entre un motif de graphe avec ports L et un graphe avec ports sujet G par $L \prec\!\!\prec G$. Les solutions au problème de sous-filtrage ont la forme $(g, G^-, \mathcal{B})$. Alors, par $\mathcal{S}ol(L \prec\!\!\prec G)$ nous dénotons l'ensemble de toutes les solutions du problème de sous-filtrage.

En général, nous supposons que pour tout problème de sous-filtrage $L \prec\!\!\!\ll G$, l'ensemble des variables et identifiants de nœud apparaissant dans chaque graphe avec ports sont disjoints, i.e., $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$ et $\mathrm{ID}(L) \cap \mathrm{ID}(G) = \emptyset$. Si ce n'est pas le cas, nous renommons de manière appropriée les variables et identifiants de nœud afin de satisfaire cette condition. Nous présumons que tous les ports avec noms variables appartenant à un même graphe avec ports sont disjoints deux à deux.

Nous qualifions un sous-filtrage de strict, lorsque le motif est un sous-graphe avec ports induit complet de G. Un sous-filtrage non-strict considère un quatrième composant de la décomposition de G selon un graphe avec ports L et un morphisme de graphes avec ports g représenté par un ensemble d'arêtes non-filtrées de G avec les deux extrémités dans $g(L)$. Alors $g(L) \cup \mathcal{U}_e$ est un sous-graphe induit complet de G et nous écrivons $G = G^- \lfloor g(L) \rfloor_{\mathcal{B}}^{\mathcal{U}_e}$.

Formellement, les ensembles de ponts et d'arêtes non-filtrées sont définis comme suit :

$$\begin{aligned} \mathcal{B} &= \{e \in E_G \mid (s_G(e) \in G^- \wedge t_G(e) \in g(L)) \vee (s_G(e) \in g(L) \wedge t_G(e) \in G^-)\} \\ \mathcal{U}_e &= \{e \in E_G \mid e \notin g(L) \wedge s_G(e) \in g(L) \wedge t_G(e) \in g(L)\} \end{aligned}$$

Nous obtenons la définition habituelle du filtrage entre le motif de graphe avec ports L et le graphe avec ports sujet G à partir de la définition du sous-filtrage où le contexte est vide, ou de manière équivalente, où le morphisme de sous-filtrage est en fait un isomorphisme.

Définition 43 (Filtrage). Nous disons que L filtre strictement G s'il existe un isomorphisme de graphes avec ports $g : L \to G$. Nous dénotons le problème de filtrage entre un motif de graphe avec port L et un graphe avec ports sujet G par $L \ll G$.

Le filtrage est non-strict si le morphisme de graphes avec ports g est un monomorphisme et seul g_V doit être surjectif. Alors G est décomposé en le graphe avec ports $g(L)$ et l'ensemble des arêtes non-filtrées (i.e., arêtes sans une pré-image dans L).

10.3.2 Un algorithme de sous-filtrage

Afin de fournir un algorithme de sous-filtrage pour les graphes avec ports, nous représentons un graphe avec ports par l'ensemble des nœuds étiquetés et l'ensemble (potentiellement vide) des équations d'adjacence (ou listes). Un nœud étiqueté est défini par le n-uplet consistant en l'identifiant du nœud, le nom du nœud et l'interface. Une équation d'adjacence est définie par l'identifiant du nœud source et l'ensemble des voisins, où un voisin est donné par l'identifiant de nœud cible et l'ensemble des arêtes (paires de ports) :

$$\begin{aligned} \mathcal{N} &::= id : nname : \{pname, \ldots, pname\} \\ \mathcal{E}q &::= (id^\frown(pname, pname)^*)^* \end{aligned}$$

Nous utilisons le symbole S pour désigner des ensembles de toute sorte, le symbole N pour désigner des ensembles de nœuds étiquetés, le symbole E pour désigner des ensembles d'équations d'adjacence, et les symboles s, n, et e pour désigner des éléments d'un ensemble générique, des nœuds, et des équations d'adjacence respectivement. Tous ces symboles peuvent être indexés.

Exemple 8. Le graphe avec ports représenté en Figure 10.2, dénoté par G, est encodé de la manière suivante :

– l'ensemble des nœuds étiquetés :

$$\{1 : A : \{a, b\}, 2 : B : \{e\}, 3 : A : \{a, b, c\}, 4 : C : \{d\}\}$$

– l'ensemble des équations d'adjacence :

$$\begin{array}{l}\{1 ::= (1^\frown\emptyset), (2^\frown(a, e), (a, e), (b, e)), (3^\frown(b, a)), (4^\frown\emptyset);\\ 2 ::= (1^\frown\emptyset), (2^\frown\emptyset), (3^\frown\emptyset), (4^\frown\emptyset);\\ 3 ::= (1^\frown\emptyset), (2^\frown\emptyset), (3^\frown\emptyset), (4^\frown(a, d));\\ 4 ::= (1^\frown(d, b)), (2^\frown\emptyset), (3^\frown\emptyset), (4^\frown\emptyset)\}\end{array}$$

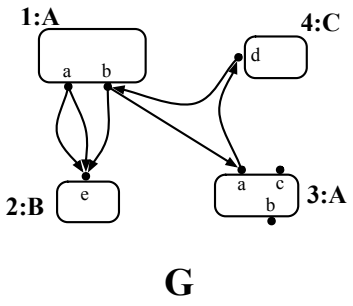


Fig. 10.2: Un exemple de graphe avec ports

Définition 44 (Pré-solution du problème de sous-filtrage de graphe avec ports). Nous disons que (σ, N, E) est une pré-solution du problème de sous-filtrage $(N_1, E_1) \preccurlyeq (N_2, E_2)$ si :

1. $\sigma : \mathrm{ID}(N_1) \cup \mathcal{X}_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{P}} \to \mathrm{ID}(N_2) \cup \nabla$ est une substitution définie sur l'ensemble des identifiants de œud, les variables de nom de nœud, et variables de nom de port apparaissant dans (N_1, E_1) ;
2. $\sigma(N_1) \subseteq N_2$, $\sigma(E_1) \subseteq E_2$, i.e., $\sigma((N_1, E_1))$ est un sous-graphe de (N_2, E_2) ;
3. $E = E_2 \setminus \sigma(E_1)$, $N = N_2 \setminus \sigma(N_1)$.

Si (N_1, E_1) et (N_2, E_2) encodent les graphes avec ports G_1 et G_2 respectivement, et (σ, N, E) est une pré-solution du problème de sous-filtrage $(N_1, E_1) \preccurlyeq (N_2, E_2)$, alors nous pouvons trouver facilement la solution correspondante pour le problème de sous-filtrage $G_1 \preccurlyeq G_2$. Ainsi, nous utiliserons également les termes de solution sous-filtrante pour une pré-solution à moins que cela ne soit pas clair à partir du contexte.

Dans ce qui suit, nous présentons un algorithme de sous-filtrage. Nous encodons le problème de sous-filtrage $(N_1, E_1) \preccurlyeq (N_2, E_2)$ pour une donnée d'entrée du problème définie sous forme de 5-uplet $N_1 \preccurlyeq N_2 \wr E_1 \star E_2 \int \emptyset \rhd \emptyset$ sur lequel les règles de l'algorithme peuvent être appliquées. Les termes impliqués dans l'algorithme de sous-filtrage, appelés termes filtrants, ont une structure "$\_ \wr \_ \star \_ \int \_ \rhd \_$" avec la sémantique suivante pour chaque argument :

(1^{er}) une liste (conjonction) d'équations de filtrage obtenues récursivement en décomposant et résolvant les équations de filtrage et en traversant le graphe avec port à partir des informations de voisinage ;

(2^e & 3^e) deux ensembles d'équations d'adjacence fournissant les informations de voisinage pour chaque membre des équations de filtrage en première position ;

(4^e) un ensemble de nœuds qui va former le graphe avec ports de contexte, extrait à partir des ensembles de nœuds de filtrage de différentes tailles ;

(5^e) la substitution (partielle) construite en résolvant les équations de filtrage dans le premier argument.

Nous considérons qu'un terme filtrant est en forme normale si, soit le premier argument a la valeur false, soit le premier argument a la valeur true (i.e., il n'y a plus d'équations de filtrage) et la première liste d'équations d'adjacence est vide. Ainsi, les deux formes normales possibles pour un terme filtrant sont false $\wr E'_1 \star E'_2 \int N' \rhd \sigma'$ et true $\wr \emptyset \star E'_2 \int N' \rhd \sigma$.

Nous regroupons les règles en trois ensembles selon leur type de modification opérée sur les termes : décomposer (D_{1-9}), résoudre (S_{1-3}), et nettoyer (C_{1-2}). Une caractéristique commune à toutes les règles de décomposition et résolution est que chacune d'elles tente de résoudre la première équation / équation la plus à gauche de la liste à partir de la première position du terme.

Les règles de décomposition agissent seulement sur la première équation de filtrage dans la liste en la décomposant ou en l'effaçant, ou en retournant un résultat d'échec pour le sous-filtrage. Nous dénotons par M une liste d'équations de filtrage.

$$
\begin{array}{ll}
(\mathrm{D}_1) & (S_1 \lessdot S_2) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow \\
 & \bigvee_{\substack{s_1 \in S_1 \\ s_2 \in S_2}} (s_1 \lessdot s_2) \wedge (S_1 \setminus \{s_1\}) \lessdot (S_2 \setminus \{s_2\}) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \\
 & \text{si } |S_1| \geq 1, |S_2| \geq 1, |S_1| + |S_2| \geq 3 \\
(\mathrm{D}_2) & (S \lessdot \emptyset) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow \text{false} \wr E_1 \star E_2 \int N \rhd \sigma \\
 & \text{si } |S| \geq 1 \\
(\mathrm{D}_3) & (\emptyset \lessdot N_2) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow M \wr E_1 \star E_2 \int N \cup N_2 \rhd \sigma \\
(\mathrm{D}_4) & (\emptyset \lessdot S) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow M \wr E_1 \star E_2 \int N \rhd \sigma \\
(\mathrm{D}_5) & (\emptyset \lessdot P) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow \text{false} \wr E_1 \star E_2 \int N \rhd \sigma
\end{array}
$$

$$(\mathrm{D}_6)\quad (id_1 : nname_1 : P_1 \prec\!\!\!\prec id_2 : nname_2 : P_2) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow$$
$$(id_1 \prec\!\!\!\prec id_2) \wedge (nname_1 \prec\!\!\!\prec nname_2) \wedge (P_1 \prec\!\!\!\prec P_2) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma$$

$$(\mathrm{D}_{6\mathrm{bis}})\quad (id_1 : nname_1 : P_1 \prec\!\!\!\prec id_2 : nname_2 : P_2) \wedge M \wr E_1 \star E_2 \int N \# N_0 \rhd \sigma \rightarrow$$
$$\bigvee_{\substack{P_2 = P_2', P_2'' \\ |P_1| = |P_2'|}} (id_1 \prec\!\!\!\prec id_2) \wedge (nname_1 \prec\!\!\!\prec nname_2) \wedge (P_1 \prec\!\!\!\prec P_2') \wedge M$$
$$\wr E_1 \star E_2 \int N \# N_0 \cup \{id_2 : nname_2 : P_2''\} \rhd \sigma$$

$$(\mathrm{D}_7)\quad (id_1 \frown S_1 \prec\!\!\!\prec id_2 \frown S_2) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow$$
$$(id_1 \prec\!\!\!\prec id_2) \wedge (S_1 \prec\!\!\!\prec S_2) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma$$

$$(\mathrm{D}_8)\quad (p_1, p_2) \prec\!\!\!\prec (p_3, p_4) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow$$
$$p_1 \prec\!\!\!\prec p_3 \wedge p_2 \prec\!\!\!\prec p_4 \wedge M \wr E_1 \star E_2 \int N \rhd \sigma$$

$$(\mathrm{D}_9)\quad a \prec\!\!\!\prec b \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow$$
$$\text{si } (a \neq b) \text{ alors } \text{ false } \wr E_1 \star E_2 \int N \rhd \sigma$$
$$\text{sinon } M \wr E_1 \star E_2 \int N \rhd \sigma$$

Chacune des équations de résolution est appliquée si le membre de gauche d'une équation de filtrage est une variable. La substitution actuelle est étendue avec une association entre la variable et la constante dans le membre de droite de l'équation de filtrage, et chaque occurrence de la variable est remplacée par la constante dans l'ensemble du terme de filtrage.

$$(\mathrm{S}_1)\quad (x \ll id) \wedge M \wr E_1; x = H_1; E_2 \star E_3; id = H_2; E_4 \int N \rhd \sigma \rightarrow$$
$$(H_1\{x \mapsto id\} \ll H_2) \wedge M\{x \mapsto id\} \wr (E_1; x = H_1; E_2)\{x \mapsto id\} \star E_3; id = H_2; E_4 \int$$
$$N \rhd \sigma \cup \{x \mapsto id\}$$

$$(\mathrm{S}_2)\quad (x \ll nname) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow$$
$$(M\{x \mapsto nname\} \wr E_1\{x \mapsto nname\} \star E_2 \int N \rhd \sigma \cup \{x \mapsto nname\}$$

$$(\mathrm{S}_3)\quad (x \ll p) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow$$
$$(M\{x \mapsto p\} \wr E_1\{x \mapsto p\} \star E_2 \int N \rhd \sigma \cup \{x \mapsto p\}$$

Les règles de nettoyage sont appliquées lorsque toutes les équations sont résolues. La première règle efface les ensembles communs de voisins à partir des équations d'adjacence du même identifiant afin d'obtenir le reste des arêtes du contexte et les ponts. Alors, nous définissons les termes de filtrage ayant le premier composant égal à false comme éléments d'identité en les retirant en utilisant la règle (C_2).

$$(\mathrm{C}_1)\quad \text{true } \wr E_1; id = H; E_2 \star E_3; id = H', H, H''; E_4 \int N \rhd \sigma \rightarrow$$
$$\text{true } \wr E_1; id = \emptyset; E_2 \star E_3; id = H', H''; E_4 \int N \rhd \sigma$$

$$(\mathrm{C}_2)\quad (\text{false } \wr E_1 \star E_2 \int N \rhd \sigma) \vee T \rightarrow T$$

Clairement, l'algorithme de sous-filtrage présenté ici n'est pas conçu pour être très efficace. Nous présentons un algorithme simple et très intuitif pour résoudre un problème de sous-filtrage. Des algorithmes plus efficaces sont disponibles dans la littérature sur

les graphes, en particulier pour les outils implantant des transformations de graphes tels que PROGRES (PROgramming with Graph REwriting Systems) [SWZ97] et AGG (Attributed Graph Grammars) [ERT97].

Lemme 1. L'algorithme de sous-filtrage termine.

Théorème 4. Tout terme de filtrage $M \wr E_1 \star E_2 \int N \rhd \sigma$ a une unique forme normale par rapport aux règles (D_{1-9}), (S_{1-3}), (C_{1-2}) consistant en la disjonction des termes de filtrage en forme normale.

Si la forme normale du terme de filtrage $N_1 \prec\!\!\prec N_2 \wr E_1 \star E_2 \int N \rhd \sigma$ correspondant au problème de sous-filtrage $(N_1, E_1) \prec\!\!\prec (N_2, E_2)$ a la forme :

1. false $\wr E_1 \star E_2 \int N \rhd \sigma$, alors le problème de sous-filtrage n'a pas de solution ;
2. $\bigvee_{i=1,n}$ true $\wr \emptyset \star E_i \int N_i \rhd \sigma_i$, $n \geq 1$, alors chaque n-uplet (σ_i, N_i, E_i), pour $i = 1, n$, est une pré-solution du problème de sous-filtrage.

10.4 Règles de Réécriture

Définition 45 (Règle de réécriture de graphe avec ports). Une règle de réécriture de graphe avec ports est donnée par deux graphes avec ports L, R sur une p-signature $\nabla^{\mathcal{X}}$ et un morphisme de nœuds $\xi : V_L \to \mathcal{P}(V_R)$, dénoté par $L \Rightarrow R$ ou $L \overset{\xi}{\Rightarrow} R$, tel que $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(L)$ et $\mathrm{ID}(R) \subseteq \mathrm{ID}(L)$. Les graphes avec ports L and R sont appelés le membre de gauche et de droite de la règle respectivement, et ξ le morphisme de correspondance.

Remarque 5. Dans le contexte de la définition ci-dessus, nous présentons informellement dans ce qui suit, quelques opérations particulières que les règles peuvent appliquer sur les nœuds :

1. s'il y a un nœud $v \in V_L$ avec $\xi(v) = \{v\}$, alors v n'est pas modifié par la règle ;
2. s'il y a un nœud $v \in V_L$ avec $\xi(v) = \emptyset$ alors le nœud v est effacé par la règle ;
3. s'il y a un nœud $v \in V_L$ avec $\xi(v) = \{v_1, v_2\}$, alors la règle duplique ou découpe le nœud ;
4. s'il y a un nœud $u \in V_R$ tel que $\xi^{-1}(\{u\})$ est défini et $\xi^{-1}(\{u\}) = \{v_1, \ldots, v_k\}$ et $\xi(v_i) = \{u\}$, pour tout $i = 1, .., k$, cela signifie que la règle fusionne les nœuds $v_1, \ldots, v_k$ dans u ;
5. s'il y a un nœud $u \in V_R$ et $v_1, \ldots, v_k$ sont tous les nœuds de V_L, avec $k \geq 2$, tels que $u \in \xi(v_1) \cap \ldots \cap \xi(v_k)$ et il existe i, $1 \leq i \leq k$ tel que v_i se découpe, alors la règle fusionne partiellement les nœuds $v_1, \ldots, v_k$ dans u.

Nous pouvons représenter une règle de réécriture de graphe avec ports sous forme de graphe avec ports si nous définissons un encodage de nœud dans le morphisme de nœud ξ donnant la correspondance entre les nœuds de L et les nœuds de R :

Définition 46 (Encoder une règle de réécriture de graphe avec ports sous forme de graphe avec ports). Une règle de réécriture de graphe avec ports $L \Rightarrow R$ est un graphe avec ports consistant en :

- deux graphes avec ports L et R sur une p-signature $\nabla^{\mathcal{X}}$ appelés, comme d'habitude, les membres de gauche et de droite respectivement, tels que $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(L)$ et $\mathrm{ID}(R) \subseteq \mathrm{ID}(L)$,
- un nœud spécial "flèche", $\Rightarrow$, qui a un port pour chaque port dans L qui est préservé dans R, et le port trou noir, nommé bh,
- des arêtes de L vers $\Rightarrow$ associant chaque port de L de manière unique à un port de $\Rightarrow$ s'il n'est pas effacé, et à bh sinon,
- des arêtes de $\Rightarrow$ vers R, associant chaque port différent de bh à un port de R.

tels que la correspondance d'un port de L à des ports de R est exprimée par les arêtes incidentes au nœud flèche.

Exprimer une règle de réécriture de graphe avec ports comme graphe avec ports est très adaptée aux représentations graphiques. Cependant, pour étudier les propriétés de la relation de réécriture de graphe avec ports, dans cette section, nous utilisons la première définition de règle de réécriture de graphe avec port donnée en Définition 45.

Définition 47 (Système de réécriture de graphe avec ports). Un système de réécriture de graphe avec ports (sur une p-signature $\nabla^{\mathcal{X}}$) est un ensemble fini de règles de réécriture de graphes avec ports (sur une p-signature $\nabla^{\mathcal{X}}$).

Exemple 9. Nous illustrons en Figure 10.3 quatre règles de réécriture de graphe avec ports : (a) décomposer un nœud avec deux ports tous deux connectés au même port d'un autre nœud en deux nœuds, ayant chacun un port, (b) effacer un nœud et ses ports, (c) effacer deux arêtes, et (d) fusionner deux nœuds.

10.5 Relation de Réécriture

Soit $L \Rightarrow R$ une règle de réécriture de graphe avec ports, et G un graphe avec ports tels qu'il existe un morphisme de sous-filtrage g pour L dans G.

L'étape de réécriture correspondant à appliquer $L \Rightarrow R$ à G peut être décomposée dans les étapes plus petites ci-dessous, comme illustré en Figure 10.4 :

1. trouver le morphisme de sous-filtrage g, le contexte G^{-}, et les ponts $\mathcal{B}$ (leur orientation n'est pas importante dans la représentation graphique),
2. identifier le membre de gauche de la règle avec un sous-graphe isomorphe dans G,
3. traduire les ponts selon le nœud flèche $\Rightarrow$,
4. retirer le nœud flèche et le sous-graphe isomorphe $g(L)$ qui est à présent déconnecté du graphe de contexte.

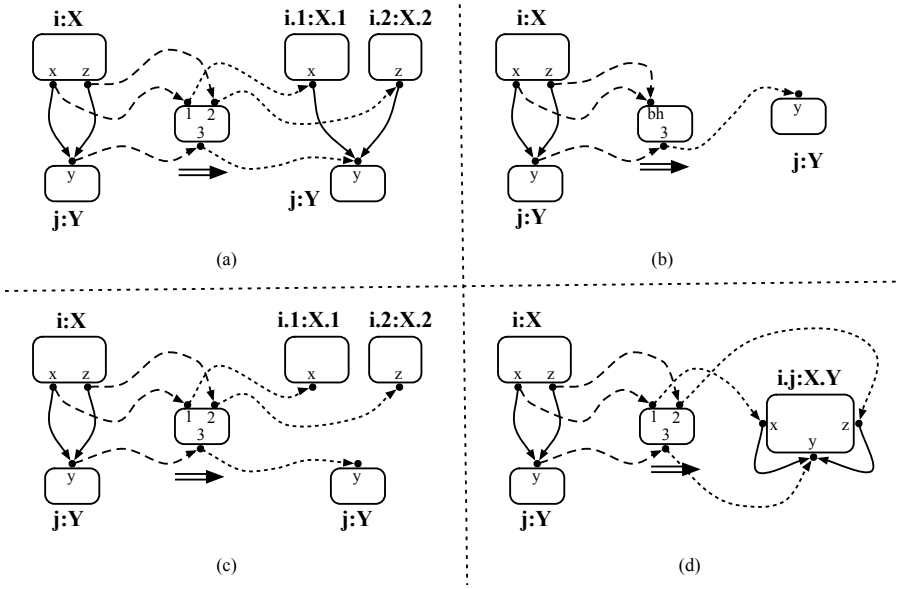


Fig. 10.3: Règles de réécriture de graphe avec ports

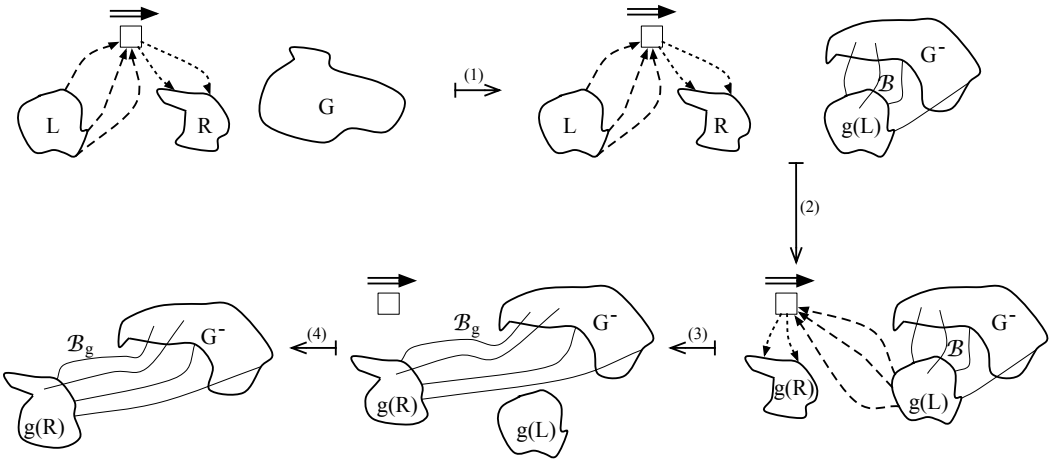


Fig. 10.4: Étapes de réécriture

Le point délicat lors de l'application de $L \Rightarrow R$ à G consiste à définir de manière adéquate le remplacement de $g(L)$ par $g(R)$ dans G et comment $g(R)$ est reconnecté avec G.

Habituellement, dans la transformation de graphe, le filtrage doit vérifier une condition de pendaison qui garantit que les extrémités des arêtes dans G^- sont préservées (non effacées) par l'application de la règle. Un pont ou une arête non-filtrée n'est pas pendante si l'extrémité v de $g_V(L_V)$ n'est pas effacée par l'application, i.e., $\xi(v) \neq \emptyset$.

Si un pont ou une arête non-filtrée a une extrémité dans $g(L)$ qui est modifiée par

la règle de réécriture, alors il ne peut pas être reconnecté à $g(R)$ lors du remplacement. Cependant, en étendant le morphisme de nœud ξ sur les arêtes, on fournit un moyen de le reconnecter ; nous appelons cette opération traduction de flèche et elle est nécessaire pour les ponts ou arêtes non-filtrées.

Si nous considérons la seconde définition de règle de réécriture de graphe avec ports, le nœud flèche $\Rightarrow$ ainsi que les arêtes incidentes enregistrent les modifications des extrémités des ponts ou nœuds partiels non-filtrés. L'opération traduction de flèche prend un nœud flèche $\Rightarrow$ ainsi que toutes les arêtes incidentes et une arête $\langle (v,p),(u,p')\rangle$ telle qu'une extrémité de l'arête, par exemple (u,p'), est la source d'une arête (p',r) incidente à $\Rightarrow$, et elle retourne l'ensemble des arêtes $\langle (v,p),(w_i,p'_i)\rangle$ pour (w_i,p'_i) une cible de l'arête ayant la source r dans le nœud flèche.

Définition 48. Étant donné un système de réécriture de graphe avec ports $\mathcal{R}$, un graphe avec ports G se réécrit en un graphe avec ports G', dénoté par $G \Rightarrow_{\mathcal{R}} G'$, s'il existe :
- une règle de réécriture de graphe avec ports $L \Rightarrow R$ dans $\mathcal{R}$ et
- une solution $(g, G^-, \mathcal{B})$ au problème de sous-filtrage $L \prec\!\!\prec G$

telles que $G = G^-\lfloor g(L)\rfloor_{\mathcal{B}}$ et $G' = G^-\lfloor g(R)\rfloor_{\Downarrow_g \mathcal{B}}$, où par $\Downarrow_g \mathcal{B}$, nous dénotons la traduction de flèche des ponts, en utilisant le nœud flèche et ses arêtes incidentes instanciées par le morphisme de sous-filtrage g.

Notons que cette définition donne la condition pour réécrire un graphe avec ports en utilisant une règle de réécriture, et également la construction du graphe avec ports en résultant.

Nous pouvons à présent définir l'application d'une solution $\sigma = (g, G^-, \mathcal{B})$ d'un problème de sous-filtrage $L \prec\!\!\prec G$ sur R en tant que $\sigma(R) = G^-\lfloor g(R)\rfloor_{\Downarrow_g \mathcal{B}}$.

Si nous considérons, dans la Définition 48, un morphisme de graphe avec ports non-strict g, alors un nouvel ensemble d'arêtes est identifié, l'ensemble des arêtes non-filtrées $\mathcal{U}_e$. En conséquence, nous avons $G = G^-\lfloor g(L)\rfloor_{\mathcal{B}}^{\mathcal{U}_e}$ et $G = G^-\lfloor g(R)\rfloor_{\Downarrow_g \mathcal{B}}^{\Downarrow_g \mathcal{U}_e}$.

Exemple 10. Nous illustrons en Figure 10.5 un résultat de l'application de la règle donnée en Figure 10.3 sur le graphe avec ports G basée sur la décomposition suivante :
- le morphisme de sous-filtrage : $g(i) = 1$, $g(j) = 2$, $g(X) = A$, $g(Y) = B$, $g(x) = a$, $g(z) = b$, $g(y) = e$,
- le graphe de contexte G^- consiste en les nœuds identifiés par 3 et 4 et les arêtes $\langle (3,a),(4,d)\rangle$, et
- les ponts $\langle (1,b),(3,a)\rangle$ et $\langle (4,d),(1,b)\rangle$.

Alors les deux ponts sont modifiés par la traduction de flèche car ils ont une extrémité appartenant à un nœud qui est découpé par la règle d'application.

10.6 Réécriture Stratégique de Graphe avec Ports

À partir de la définition de réécriture stratégique abstraite rappelée dans Chapitre 1, nous pouvons formaliser la réécriture stratégique de graphe avec ports.

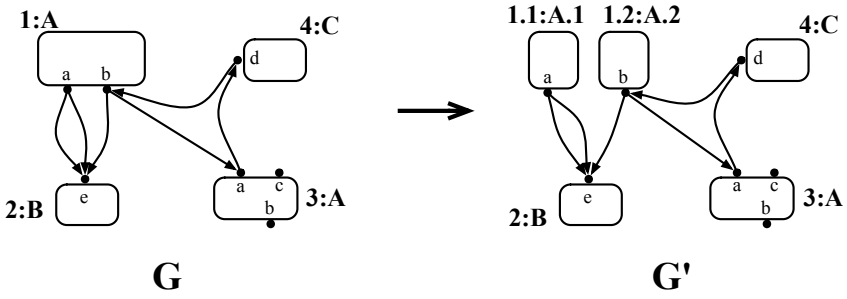


Fig. 10.5: Une application de la règle de réécriture de graphe avec ports de la Figure 10.3 (a) sur un graphe avec ports G résultant en le graphe avec ports G'

Définition 49 (Réécriture stratégique de graphe avec ports). Étant donné un système de réduction abstrait $\mathcal{A} = (\mathcal{O}_{\mathcal{R}}, \mathcal{S}_{\mathcal{R}})$ généré par un système de réécriture de graphe avec ports $\mathcal{R}$, et une stratégie ζ de $\mathcal{A}$, une dérivation stratégique de réécriture de graphe avec ports (ou dérivation de réécriture de graphe avec ports sous la stratégie ζ) est un élément de ζ.

Une étape de réécriture stratégique de graphe avec ports sous ζ est une étape de réécriture $G \rightarrow_{\mathcal{R}} G'$ qui apparaît dans une dérivation de ζ.

L'idée d'imposer des conditions de séquençage sur les applications de transformation de graphe remonte aux grammaires de graphes programmés introduites dans [Bun79] qui considèrent des structures de contrôle impératives. Plus tard, les auteurs de PROGRES considérèrent non seulement ces structures de contrôle impératives déterministes pour appliquer des transformations de graphe, mais ils les définirent dans un style à la Prolog en utilisant la recherche en profondeur et le retour-arrière [Sch97, SWZ97] afin de contrôler le choix non-déterministe de l'application de règle. Plus récemment, dans [HP01], les auteurs ont montré que l'application non-déterministe, la composition séquencielle, et l'itération forment un ensemble minimal de constructions de programmation suffisantes pour assurer la complétude calculatoire d'un langage de programmation fondé sur la transformation de graphe.

10.7 Graphes avec Ports Faibles

Parfois, nous n'avons qu'une information partielle sur les ports ou les nœuds ; ou, pour des raisons de simplicité, nous considérons dans une règle de réécriture de graphe avec ports, uniquement les ports pertinents pour la transformation. Nous introduisons une restriction des graphes avec ports où seul un sous-ensemble des ports de l'interface d'un nom de nœud est considéré. Nous appelons un tel sous-ensemble InterfacePertinente et les graphes avec ports sont appelés graphes avec ports faibles. Dans ce qui suit, nous ajustons en conséquence les définitions de morphisme et relation de réécriture pour les graphes avec ports faibles.

Définition 50 (Graphe avec ports faible). Un graphe avec ports faible G sur une p-signature $\nabla^{\mathcal{X}}$ est un graphe avec ports avec la restriction suivante : l'ensemble des ports (interface) de chaque nœud $v \in V_G$ est un sous-ensemble de Interface($lv_G(v)$). Nous dénotons par InterfacePertinente(v) l'ensemble des ports associé à chaque nœud $v \in V_G$.

Définition 51 (Morphisme de graphes avec ports faible). Un morphisme de graphes avec ports faible est un morphisme de graphes avec ports $f : G \to H$ avec la condition faible sur l'égalité des interfaces de deux nœuds associés :

$$\text{pour chaque } v \in V_G,\ \text{InterfacePertinente}(lv_G(v)) \subseteq \text{InterfacePertinente}((f_V \circ lv_H)(v))$$

En suivant les définition de la Section 10.2, les définitions correspondantes pour les graphes avec ports faibles peuvent être déduites simplement, en particulier, les notions de règle de réécriture de graphe avec ports faible, sous-graphe faible, et sous-filtrage faible.

Un morphisme de graphes avec ports faible implique l'existence d'un ensemble d'interfaces de nœuds partiels non-filtrés, correspondant à un ensemble de nœuds partiels non-filtrés $\mathcal{U}_n$:

$$\mathcal{U}_n = \{v \in g_V(V_L) \mid \text{Interface}(lv_H(v)) \setminus \text{InterfacePertinente}(v) \neq \emptyset\}$$

Soit $L \Rightarrow R$ une règle de réécriture de graphe avec ports faible, sous-filtrant faiblement un graphe avec ports G en utilisant la décomposition suivante :

$$G = G^- \lfloor g(L) \cup \mathcal{U}_n \rfloor_{\mathcal{B}}$$

Alors le résultat de l'application de la règle de réécriture basé sur ce sous-filtrage a la forme :

$$G' = G^- \lfloor g(R) \cup V \rfloor_{\Downarrow_g \mathcal{B}} \quad \text{où} \quad V \in \Downarrow_g \mathcal{U}_n$$

puisque l'application de la traduction de flèche peut avoir différents choix de destination pour les ports non-filtrés.

De manière évidente, pour un morphisme de graphes avec port non-strict faible, nous avons :

$$G = G^- \lfloor g(L) \cup \mathcal{U}_n \rfloor_{\mathcal{B}}^{\mathcal{U}_e} \quad \text{et} \quad G' = G^- \lfloor g(R) \cup V \rfloor_{\Downarrow_g \mathcal{B}}^{\Downarrow_g \mathcal{U}_e} \quad \text{où} \quad V \in \Downarrow_g \mathcal{U}_n$$

Exemple 11. En Figure 10.6, nous illustrons le graphe avec ports résultant de la réécriture de G (aussi donné en Figure 12.1) en utilisant la règle donnée en Figure 10.3 (a). Les graphes avec ports résultats G' et G'' sont obtenus en découpant le nœud 1, en choisissant de placer le port non-filtré c en 1.1 et en 1.2 respectivement, puis en redirigeant les deux ponts $\langle (4, d), (1, c) \rangle$ et $\langle (1, b), (3, a) \rangle$ vers 1.1 et 1.2 respectivement. Dans l'étape intermédiaire, nous mettons en relief l'incidence du nœud 4 au nœud partiel non-filtré 1 avec l'ensemble de ports $\{c\}$. La substitution de nœud peut associer ce nœud partiel à chacun des deux nœuds résultants 1.1 et 1.2. Ainsi, deux solutions sont possibles.

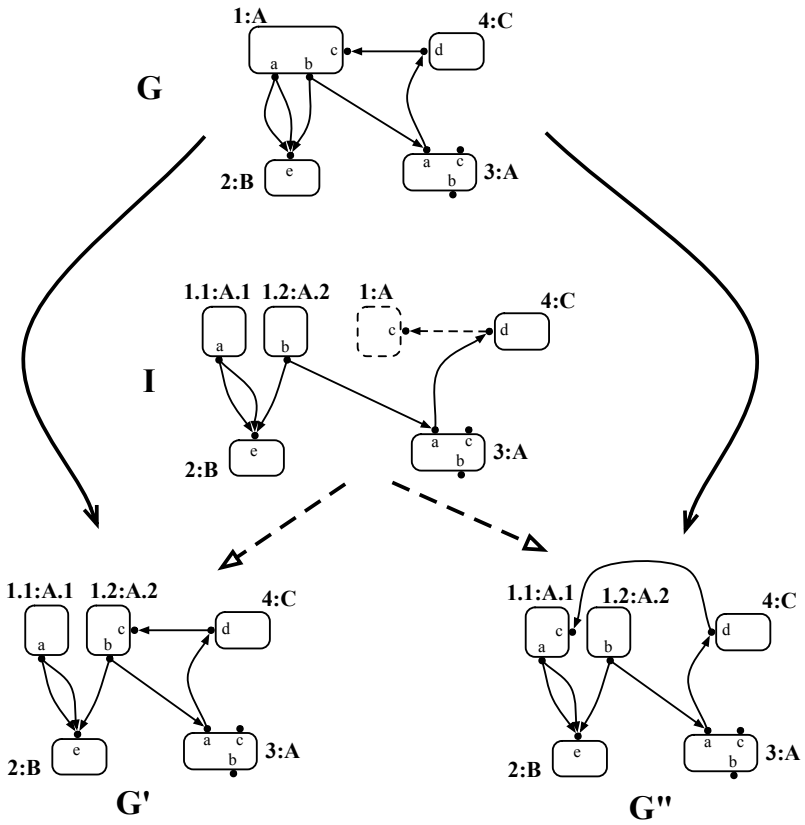


Fig. 10.6: Une application de la règle de réécriture de graphe avec ports donnée en Figure 10.3 (a) sur G résultant en deux graphes avec ports G' et G'' avec un graphe avec ports intermédiaire I

10.8 Sur la Confluence de la Réécriture de Graphe avec Ports

Dans cette section, nous analysons la propriété de confluence pour les systèmes de réécriture de graphe avec ports, en nous basant sur l'approche utilisée dans [Plu05] pour les systèmes de réécriture d'hypergraphe. En premier, nous introduisons la catégorie de graphe avec port où nous définissons le concept de dérivation directe correspondant à la réécriture de graphe avec ports en une étape. Puis, nous donnons informellement les conditions que les systèmes de réécriture de graphe avec ports doivent satisfaire afin d'être localement confluent.

Soit PGraph une catégorie ayant comme objets des graphes avec ports étiquetés sur une p-signature fixe ∇ et des morphismes de graphes avec ports et des morphismes de nœud comme flèches.

Nous définissons un pushout particulier dans la catégorie PGraph fondée sur la définition classique de pushout de [Mac98].

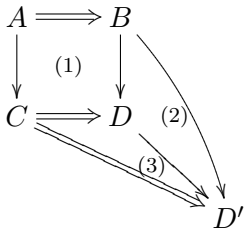
Définition 52 (Pushout alternant). Étant donnés un morphisme de nœuds $A \Rightarrow B$ et un

morphisme de graphes avec ports $A \to C$, un graphe avec ports D avec un morphisme de graphes avec ports $B \to D$ et un morphisme de nœuds $C \Rightarrow D$ est un pushout alternant de $A \Rightarrow B$ et $A \to C$ si les conditions suivantes sont satisfaites :

Commutativité $A \Rightarrow B \to D = A \to C \Rightarrow D$ ou, via un diagramme, les diagrammes commutent :

$$\begin{array}{ccc} A & \Longrightarrow & B \\ \downarrow & (1) & \downarrow \\ C & \Longrightarrow & D \end{array}$$

Universalité Pour tout graphe avec ports D', morphisme de graphes avec ports $B \to D'$ et morphisme de nœud $C \Rightarrow D'$ tels que $A \Rightarrow B \to D' = A \to C \Rightarrow D'$, il existe un unique morphisme de graphes avec ports $D \to D'$ tel que $B \to D \to D' = B \to D'$ et $C \Rightarrow D \to D' = C \Rightarrow D'$ (i.e., les diagrammes (2) et (3) ci-dessous commutent respectivement).



Remarque 6. Un pushout alternant dans PGraph est un pushout si nous considérons les morphismes de graphes avec ports comme un morphisme de nœuds associant un nœud au singleton contenant son image.

La construction d'un pushout alternant est basée sur la construction des pushouts en théorie des ensembles comme donnée en [Ehr79]. Soient $\xi : A \Rightarrow B$ un morphisme de nœuds et $g : A \to C$ un morphisme de graphes avec ports injectif. Alors un pushout alternant

$$\begin{array}{ccc} A & \overset{\xi}{\Longrightarrow} & B \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle g^*} \\ C & \underset{\xi^*}{\Longrightarrow} & D \end{array}$$

peut être construit comme suit :

- $V_D = (V_B + V_C)/_{\approx}$ où $\approx$ est la relation d'équivalence générée par la relation $\sim$ définie par $g(v) \sim \xi(v)$ pour tout $v \in V_A$. Alors, pour $u \in V_B + V_C$, la classe d'équivalence de u par rapport à $\approx$ est définie comme :

$$[u] = \begin{cases} \{g(u)\} & \text{si } u \in V_B \\ \{v\} & \text{si } u \in V_C \setminus V_{g(A)} \end{cases}$$

  Nous remarquons que la première case couvre aussi la situation d'un nœud $u \in V_{g(A)}$ puisqu'il existe un nœud $v \in V_A$ tel que $g(v) = u$, et $\xi(v) = \{v_1, \ldots, v_n\} \subseteq V_B$. Ainsi $[u] = \{[v_1], \ldots, [v_n]\}$.
  En d'autres termes, pour $u \in V_A$, l'équivalence $g(v) \sim \xi(v)$ implique que le représentant de la classe d'équivalence de $g(v)$ et $\xi(v)$ soit $(g \circ \xi)(v)$ défini en Section 10.2.

- $E_D = (E_B + E_C)/_{\approx}$ où $\approx$ est la relation d'équivalence générée par la relation $\sim$ définie par $g(e) \sim \xi(e)$ pour tout $e \in E_A$. Alors pour $e \in E_B + E_C$:
  - si $e \in E_B$, alors $[e] = e$,
  - si $e \in E_C$, $s_C(e) \in V_{g(A)}$, $t_C(e) \notin V_{g(A)}$, et $\xi(g^{-1}(s_C(e))) = \emptyset$, alors $[e]$ n'est pas défini ;
  - si $e \in E_C$, $t_C(e) \in V_{g(A)}$, $s_C(e) \notin V_{g(A)}$, et $\xi(g^{-1}(t_C(e))) = \emptyset$, alors $[e]$ n'est pas défini ;
  - si $e \in E_C$, $s_C(e) \in V_{g(A)}$, $t_C(e) \notin V_{g(A)}$, et $\xi(g^{-1}(s_C(e))) = \{v_1, \ldots, v_k\}$, alors $[e] = \{e_1, \ldots, e_k\}$ où $s_D(e_i) = g(v_i)$ et $t_D(e_i) = t_C(e)$, pour tout $i = 1, k$;
  - si $e \in E_C$, $t_C(e) \in V_{g(A)}$, $s_C(e) \notin V_{g(A)}$, et $\xi(g^{-1}(t_C(e))) = \{v_1, \ldots, v_k\}$, alors $[e] = \{e_1, \ldots, e_k\}$ où $t_D(e_i) = g(v_i)$ et $s_D(e_i) = s_C(e)$, pour tout $i = 1, k$.
- $\xi^* : C \Rightarrow D$ et $g^* : B \to D$ sont les morphismes de nœuds et de graphes avec ports respectifs envoyant chaque élément dans sa classe d'équivalence, c'est-à-dire, $\xi^*(x) = [x]$ et $g^*(x) = [x]$ pour les nœuds et les arêtes séparément.

Définition 53 (Dérivation). Soient G et H deux graphes avec ports, $L \overset{\xi}{\Rightarrow} R$ une règle de réécriture de graphe avec ports, et $g : L \to G$ un morphisme de graphes avec ports injectif. Alors G dérive directement H par ξ et g, dénoté par $G \Rightarrow_{\xi,g} H$ si le pushout alternatif suivant existe :

$$\begin{array}{ccc} L & \overset{\xi}{\Longrightarrow} & R \\ {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle g^*} \\ G & \underset{\xi^*}{\Longrightarrow} & H \end{array}$$

Une dérivation d'un graphe avec ports G vers un graphe avec ports H, dénoté par $G \Rightarrow^* H$, est une séquence de dérivations directes $G = G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n = H$ pour un $n \geq 0$.

Nous remarquons que la construction donnée ci-dessus pour le pushout alternant correspond à une réécriture de graphe avec ports de G en H utilisant la règle de réécriture de graphe avec ports $L \overset{\xi}{\Rightarrow} R$ et le morphisme sous-filtrant g.

Lemme 2 (Correspondance entre une relation de réécriture et une dérivation directe). Soient G et H deux graphes avec ports, $L \overset{\xi}{\Rightarrow} R$ une règle de réécriture de graphe avec ports, et $g : L \to G$ un morphisme de graphes avec ports injectif. Alors G se réécrit en H en utilisant ξ et g si et seulement si G dérive directement H par ξ et g.

Soient $L_i \overset{\xi_i}{\Rightarrow} R_i$, $i = 1, 2$, deux règles de réécriture de graphe avec ports. Les deux réécritures à une étape (ou dérivations directes) $G \Rightarrow_{\xi_i,g_i} H_i$, $i = 1, 2$, avec $g_1 \neq g_2$ si $\xi_1 = \xi_2$ sont indépendantes si l'intersection des membres de gauche de r_1 et r_2 dans G consiste en des éléments non modifiés par l'application de la règle. Si les deux étapes sont indépendantes et $G = g_1(L_1) \cup g_2(L_2)$, alors la paire des deux réécritures est une paire critique. Une paire critique $H_1 \Leftarrow G \Rightarrow H_2$ est joignable s'il existe deux graphes avec ports isomorphes I_1 et I_2 tels que $H_i \Rightarrow^* I_i$.

La joignabilité de toutes les paires critiques d'un système de réécriture de graphes avec ports ne garantit pas la confluence locale (comme c'est le cas pour les systèmes de

réécriture de termes). La condition de joignabilité d'une paire critique $H_1 \Leftarrow_{\xi,g_1} G \Rightarrow_{\xi,g_2} H_2$ doit être renforcée de sorte que pour chaque nœud v dans G préservé par les deux dérivations directes sous la forme du même ensemble de nœuds $\{v_1, \ldots, v_n\}$, $n \geq 1$, il y a des dérivations $H_1 \Rightarrow^* I_1$ et $H_2 \Rightarrow^* I_2$ et un isomorphisme $f : I_1 \to I_2$ tels que l'ensemble $\{v_1, \ldots, v_n\}$ a une décomposition en V' et V'' avec

- tous les nœuds dans I_1 et I_2 dérivés des nœuds dans V' sont relié par l'isomorphisme f, et
- tous les nœuds de V'' sont effacés pendant les dérivations.

Cette condition est appelée joignabilité forte.

Alors, les résultats sur la confluence locale et la confluence prouvés pour les systèmes de réécriture d'hypergraphe dans [Plu05] sont également utilisés pour les systèmes de réécriture de graphes avec ports.

Lemme 3 (Confluence). Un système de réécriture de graphes avec ports est confluent localement si toutes ses paires critiques sont fortement joignables. Si, de plus, le système de réécriture de graphes avec ports termine, alors il est confluent.

10.9 Comparaison avec les Systèmes Réactifs de Bigraphes

Les systèmes réactifs de bigraphes (Bigraphical Reactive Systems, BRSs) sont un modèle graphique de calcul où la connectivité et la localité sont importants [Mil01, JM04, Mil06]. Un bigraphe consiste en une forêt d'arbres décrivant les positions des objets calculatoires dans le système, appelée graphe de places, et un hypergraphe décrivant les liens ou connexions entre les objets, appelé graphe de liens. Alors, une règle de réaction dans un BRS est une paire de bigraphes, un redex représenté par un motif d'entrelacement et liage, et un réactum indiquant comment les réactions changent le motif. Via de telles règles de réaction, BRS représente un formalisme adapté pour la modélisation de la connectivité mobile (en changeant le graphe de liens) et de la localité mobile (en changeant le graphe de places).

Un objet dans un bigraphe est représenté par un nœud avec un nombre fixe de ports pour connecter d'autres nœuds donnés par un contrôle associé. L'ensemble de tous les contrôles possibles pour les nœuds d'un bigraphe représente sa signature. De plus, puisque les aspects de localité et connectivité sont indépendants, les nœuds peuvent être entrelacés et les ports sont liés indépendemment de l'entrelacement.

Les graphes avec ports sont une variation des bigraphes où : les nœuds sont étiquetés avec des contrôles d'arité fixe (correspondant aux ensembles de ports) mais les contrôles sont atomiques (pas d'entrelacement), le graphe de liens est simplement un graphe, et non un hypergraphe (une arête connecte exactement deux ports de nœuds). Cependant, les ports dans les graphes avec ports n'ont pas un degré d'incidence fixé, alors que pour les bigraphes, le degré d'incidence d'un port est 1 ou 0 (quand le port est libre).

L'application d'une règle de réaction sur un bigraphe est basée sur la décomposition du bigraphe en redex de la règle de réaction, un contexte, et certains paramètres discrets, et en remplaçant le redex par le réactum. Le problème de trouver une telle décomposition, appelé filtrage, est traité dans [BDGM07] en utilisant une classe de bigraphes basiques et

des opérateurs pour décomposer le bigraphe et après, à partir de telles décompositions, pour à la fois le redex et le bigraphe à transformer, en identifiant inductivement le redex et le contexte. Les façons dont est traité le problème de filtrage pour les bigraphes et pour les graphes avec ports semble équivalent modulo l'expressivité additionnelle du formalisme bigraphique : le problème de filtrage est réduit dans les deux cas à la recherche d'un redex, d'un contexte, d'un paramètre ; la fonctionnalité d'un tel paramètre est similaire à la notion d'ensemble de ponts utilisée pour les graphes avec ports. En conséquence, en étudiant de manière plus approfondie l'algorithme de filtrage à base de décomposition fourni pour les bigraphes, il semblerait qu'on puisse trouver un algorithme de filtrage pour les graphes avec ports qui soit plus efficace, et du moins plus élégant. Cependant, nous avons défini des règles de réaction plus expressives pour les graphes avec ports qui spécifient la division d'un nœud, alors que dans les systèmes bigraphiques, on ne peut que dupliquer un nœud. Il serait intéressant d'étudier quelle modification, une telle opération de division impliquerait pour le modèle catégorique des bigraphes.

À partir de travaux existants sur les bigraphes et les systèmes de réaction de bigraphes, nous pourrions améliorer la théorie mathématique des graphes avec ports, en utilisant la théorie des catégories, ainsi qu'en empruntant des techniques pour la définition d'extensions des graphes avec ports, telles que, par exemple, les bigraphes stochastiques [KMT08].

Nous avons identifié la structure des graphes avec ports indépendemment des bigraphes existants, dans le contexte de la modélisation d'interaction entre protéines, ainsi comme un formalisme basé sur l'information nécessaire à la modélisation de tels procédés biochimiques. Ajouter les capacités d'entrelacement pour les nœuds comme dans les bigraphes pourrait permettre de modéliser d'autres types d'interactions biologiques, comme par exemple, le processus de "budding" de la membrane, où les molécules vivant à l'intérieur de la cellule sont transportées vers d'autres cellules, qui est déjà formalisée dans le contexte des bigraphes stochastiques [KMT08]. Néanmoins, d'autres exemples intéressants motivant la nécessité d'ajouter l'entrelacement aux graphes avec ports et ne nécessitant pas toutes les caractéristiques spécifiques aux bigraphes devraient être envisagés.

10.10 Conclusions

Dans ce chapitre, nous avons formalisé la structure des graphes avec ports, le problème de (sous)-filtrage pour les graphes avec ports, et une relation de réécriture de graphe avec ports. Nous avons aussi analysé la confluence de la relation de réécriture de graphe avec ports en nous basant sur des résultats classiques pour les hypergraphes. Le sous-filtrage et la relation de réécriture définis pour les graphes avec ports dans ce chapitre sont alors utilisés pour instancier le calcul biochimique abstrait dans le prochain chapitre, afin d'obtenir un calcul de réécriture pour les graphes avec ports.

Il y a encore du travail à faire dans la formalisation des fondements mathématiques de la réécriture de graphes avec ports en utilisant des éléments de la théorie des catégories, en particulier pour analyser l'adéquation de la structure de graphes avec ports

comme structure de haut-niveau dans le contexte des systèmes de remplacement de haut-niveau (High-Level Replacement Systems, HLRS) [EHKPP90]. Alors, nous devrions tester quelles conditions-HLR sont satisfaites dans la catégorie des graphes avec ports, afin que nous puissions bénéficier des résultats existant pour les systèmes HLR, par exemple le parallélisme. Aussi, dans cette direction, une autre perspective théorique d'étude de la structure des graphes avec ports pourrait être de considérer les catégories HLR adhésives [EPPH06].

Un autre direction future, plus pragmatique, concerne le problème de (sous)-filtrage sur les graphes avec ports et la recherche d'un algorithme de filtrage efficace. Une solution possible est de mettre en relation un filtrage de graphes avec ports avec un filtrage de bigraphes ; cependant les graphes avec ports ont une structure bien plus simple que les bigraphes, ainsi, un algorithme de filtrage pour les bigraphes pourrait parfois ne pas être efficace pour les graphes avec ports. Aussi, quelques algorithmes généraux pour le filtrage par motifs pourraient être considérés [Ull76, ST99, LV02, Val02, Zün96]. De par la structure des graphes avec ports, qui est plus riche que celle de graphe, et en même temps, qui est un genre particulier de bigraphes, il semblerait que l'algorithme de filtrage pour les graphes avec ports soit situé quelque part entre les algorithmes de filtrage classiques pour les graphes, et ceux pour les bigraphes.

11 Le ρ_{pg}-Calcul : Un Calcul Biochimique Basé sur la Réécriture Stratégique de Graphe avec Ports

Introduction

Dans la Partie I, nous avons introduit un Calcul Biochimique Abstrait, le $\rho_{\langle\Sigma\rangle}$-calcul, modélisant des interactions entre molécules abstraites sur une structure décrite par les objets d'une catégorie Σ. Dans ce chapitre, nous instancions le $\rho_{\langle\Sigma\rangle}$-calcul avec la catégorie PGraph : nous considérons la structure de graphe avec ports et la relation de réécriture de graphe avec ports définies dans le Chapitre 10 pour modéliser les molécules abstraites et leur interactions respectivement. Le résultat de cela est un calcul de réécriture de graphe avec ports, appelé le ρ_{pg}-calcul. La structure de graphe avec ports apporte une plus grande expressivité dans la représentation des objets du calcul. En conséquence, nous sommes capable de définir des règles d'évaluation pour le filtrage et le remplacement en utilisant des nœuds et des transformations de graphes avec ports appropriés.

En Section 11.1, nous présentons les caractéristiques principales de la syntaxe, et en Section 11.2 la sémantique du calcul fournie par la structure de graphes avec ports.

11.1 Syntaxe

Nous introduisons pas à pas, tous les éléments définissant la syntaxe du calcul. Nous commençons par définir les objets graphes avec ports sous forme de graphes avec ports et les abstractions (règles de réécriture de graphe avec ports) également sous forme de graphes avec port. Toutes les entités du calcul sont présentée graphiquement ici.

Nœuds

Un nœud est décrit par un identifiant unique, un nom, et un ensemble de ports. Les identifiants de nœud sont soit des entiers, soit des lettres minuscules entre i et n, ou leur concaténation. Les noms de nœud sont (i) des chaînes de caractères majuscules qui peuvent être indexées par ou concaténées avec des entiers, et (ii) des concaténations de noms. Les noms de ports sont des chaînes de caractères minuscules éventuellement indexées par des entiers. Soient id, $pname$, et $nname$ les ensembles des identifiants de nœuds, des noms de ports, et des noms de nœuds respectivement.

La syntaxe des ports et nœuds est la suivante :

$$\begin{array}{rrcl} \text{Ports} & \mathcal{P} & ::= & \bullet\, pname \\ \text{Nœuds} & \mathcal{N} & ::= & \mathcal{X}_{\mathcal{N}} \mid id : nname : \mathcal{P}^{+} \end{array}$$

Un port est un point avec un nom de port. Un nœud est soit une variable, soit une entité avec un identifiant unique, un nom de nœud et un ensemble de ports. Graphiquement, un nœud est représenté sous la forme d'une boîte avec des points à la surface, correspondant aux ports.

Objets Graphes avec Ports

Nous modélisons les états d'un système au moyen de graphes avec ports appelés objets graphes avec ports, qui peuvent être soit une variable, soit un graphe avec ports vide, soit un nœud, soit deux objets graphes avec ports juxtaposés, soit un nœud avec une boucle sur le même port, soit un nœud avec une arête connectant deux ports différents, soit deux objets graphes avec ports connectés :

$$\begin{array}{rrcll} \text{Objets Graphes avec Ports} & \mathcal{O} & ::= & \mathcal{X}_{\mathcal{O}} & \text{variables} \\ & & \mid & \varepsilon & \text{graphe vide} \\ & & \mid & \mathcal{N} & \text{nœud} \\ & & \mid & \mathcal{O}\ \mathcal{O} & \text{juxtaposition} \\ & & \mid & \mathcal{N} \text{ (boucle)} & \text{boucle} \\ & & \mid & \mathcal{N} \text{ (connexion)} & \text{connexion} \\ & & \mid & \mathcal{O} \text{ (connexion) } \mathcal{O} & \text{connexion} \end{array}$$

Abstractions

Une abstraction de premier ordre dans le ρ_{pg}-calcul consiste en deux objets graphes avec ports pour les membres de gauche et de droite, et un nœud flèche intégrant la correspondance entre les deux membres. Le nœud flèche a deux ports particuliers, un port poignée p_0 et un port trou noir bh. Quelques restrictions doivent être imposées sur la connectivité de ces deux types de ports. Un port poignée peut être connecté uniquement à un autre port poignée, et un port trou noir peut uniquement être la cible d'une arête dont la source est soit un port poignée ou un port trou noir.

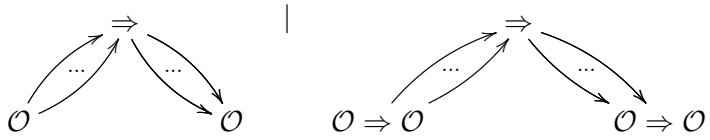
À partir de la Définition 46, une abstraction $O_1 \Rightarrow O_2$ est un graphe avec ports construit comme suit :

- le nœud flèche a un port pour chaque port de O_1 qui est préservé dans O_2, le port trou noir et le port poignée ;
- une arête connecte chaque port préservé p de O_1 à son port correspondant du nœud flèche, qui à son tour est connecté aux ports correspondants de p dans O_2 ;

– les ports effacés de O_1 sont connectés au trou noir.

Pour les abstractions de la forme $(O_1 \Rightarrow O_2) \Rightarrow (O'_1 \Rightarrow O'_2)$, le nœud flèche principale connecte des ports distincts du trou noir pour la flèche principale de $O_1 \Rightarrow O_2$ à des ports de la flèche principale de $O'_1 \Rightarrow O'_2$ suivant le morphisme entre les deux abstractions.

Les abstractions de $\mathcal{A}_0$ sont définies graphiquement comme suit :

$\mathcal{A}_0 ::=$ 

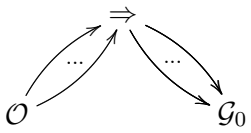
Nous identifions le nœud flèche principale d'une abstraction comme le nœud flèche dont la poignée n'est pas mis en correspondance avec un autre nœud flèche. Ainsi, dans une abstraction, la poignée de la flèche principale ne peut pas être une cible ou une source d'une arête dont la source ou cible respectivement est un nœud flèche.

Par construction, une abstraction est un graphe avec ports.

Une classe intermédiaire de molécules de graphes avec ports abstraites est définie comme suit :

$$\mathcal{G}_0 ::= \mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{G}_0 \; \mathcal{G}_0$$

Alors, l'ensemble des abstractions est défini par :

$\mathcal{A} ::= \quad \mathcal{A}_0 \quad |$ 

où pour chaque abstraction de la forme $O \Rightarrow G_0$ avec $G_0 \in \mathcal{G}_0$, les poignées des nœuds flèches des abstractions dans G_0 sont connectés à la poignée du nœud flèche dans l'abstraction.

Exemple 12. En Figure 11.1 nous illustrons une abstraction de la forme $O \Rightarrow G_0$ avec $G_0 \in \mathcal{G}_0$ qui spécifie qu'un nœud avec n'importe quel nom et avec deux ports tous les deux connectés à un port d'un autre nœud se découpe en deux nœuds, ayant chacun un port, les connexions restant préservées. De plus, l'abstraction introduit une nouvelle abstraction qui découpe de manière similaire tout nœud ayant le même nom que celui déjà découpé par l'abstraction. Ici, nous représentons le câblage en utilisant des tirets afin de distinguer clairement les arêtes des membres de gauche et de droite de l'abstraction.

Molécules Graphes avec Ports

Soit $\mathcal{X}$ un ensemble de variables. L'ensemble $\mathcal{G}$ de tous les objets ρ_{pg} élémentaires du calcul, appelés molécules graphes avec ports est défini schématiquement par :

$$\mathcal{G} ::= \mathcal{X} \mid \mathcal{O} \mid \mathcal{A} \mid \mathcal{G} \; \mathcal{G} \mid \varepsilon$$

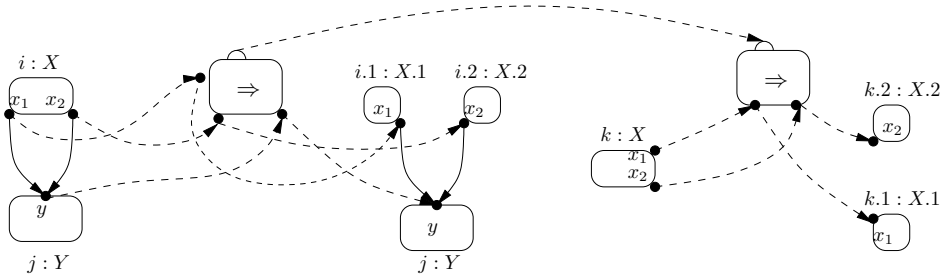


Fig. 11.1: Une abstraction avec câblages explicites qui spécifient le découpage d'un nœud identifié par le couple de variables $i : X$ connectée à un nœud $j : Y$ et introduit une règle découpant tout nœud ayant le même nom que i

En utilisant une structure de graphes avec ports pour instancier la structure paramétrique Σ dans le $\rho_{\langle\Sigma\rangle}$-calcul, nous obtenons une définition de molécules graphes avec ports.

En nous basant sur la structure particulière des graphes avec ports, nous définissons une congruence structurelle sur les molécules graphes avec ports.

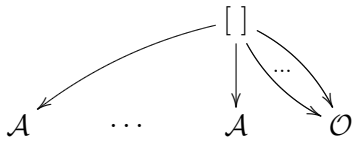
Définition 54 (Congruence structurelle sur les molécules graphes avec ports). La congruence structurelle sur les molécules graphes avec ports dans monde simple est la plus petite congruence close par rapport à une permutation des ports d'un nœud, à la juxtaposition permutative _ _ et à la connexion d'arêtes, satisfaisant ainsi les axiomes suivants :

$$
\begin{aligned}
i : name : (p_1, \ldots, p_k) &\equiv i : name : (p_\pi(1), \ldots, p_\pi(k)) \text{ pour toute permutation de taille } k \\
O_1 \curvearrowright O_2 &\equiv O_2 \curvearrowleft O_1 \\
G\ \varepsilon &\equiv G \\
G_1 \ldots G_n &\equiv G_\pi(1) \ldots G_\pi(n) \text{ pour toute permutation de taille } n
\end{aligned}
$$

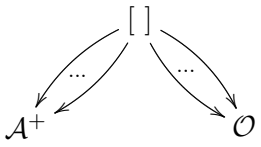
Mondes

Nous rappelons qu'un monde est un conteneur représentant les molécules graphes avec ports dans un environnement. L'environnement est "conscient" de tous les nœuds qu'il contient et des connexions entre eux. Afin d'exprimer cette conscience, nous attachons à chaque nœud dans un objet graphe avec ports, un port poignée auxiliaire, dénoté habituellement par p_0. Nous définissons alors l'opérateur [ ] comme un nœud avec un port poignée, et un monde est un graphe avec ports où la poignée du nœud [ ] est connecté aux poignées des flèches principales de toutes les abstractions et aux poignées de tous les nœuds de l'objet graphe avec ports qui ne sont pas des abstractions. Nous rappelons que ce nœud [ ] correspond à un opérateur variadique permutatif.

Un monde a la représentation graphique suivante :



ou, de manière équivalente :



ou simplement un objet graphe avec ports s'il n'y a pas d'abstraction dans l'état :

À l'écrit, nous utilisons la notation $[G]$ pour un monde contenant une molécule graphe avec ports $G \in \mathcal{G}$. Par exemple, les notations correspondantes pour les trois graphes avec ports représentés précédemment sont $[\mathcal{A} \ldots \mathcal{A}\ \mathcal{O}]$, $[\mathcal{A}^+\ \mathcal{O}]$, et $[\mathcal{O}]$ respectivement.

Soit $\mathcal{Y}$ un ensemble de variables. Alors l'ensemble $\mathcal{V}$ des mondes est défini comme suit :

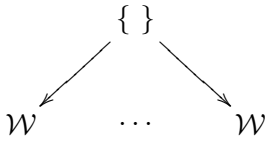
$$\mathcal{V} ::= \mathcal{Y} \mid [\mathcal{G}]$$

Multivers

Nous introduisons un nœud avec un port poignée pour l'opérateur { } qui construit une structure de mondes (ou multivers). La poignée du nœud { } est connecté aux poignées de tous les mondes composants. Une structure de mondes consiste soit en une variable dans un ensemble $\mathcal{Z}$, soit un monde, soit un multi-ensemble de structures de mondes :

$$\mathcal{W} ::= \mathcal{Z} \mid \{\mathcal{V} \ldots \mathcal{V}\} \mid \{\mathcal{W} \ldots \mathcal{W}\}$$

De manière similaire aux mondes, la notation $\{\mathcal{W}\ \mathcal{W}\}$ représente le graphe avec ports suivant :



La théorie sous-jacente à la structure de multi-ensemble de l'opérateur { } suit les lignes de la congruence structurelle définie dans le cas abstrait en Section 3.6 basée sur la congruence structurelle pour les molécules graphes avec ports de la Définition 54.

11.2 Sémantique

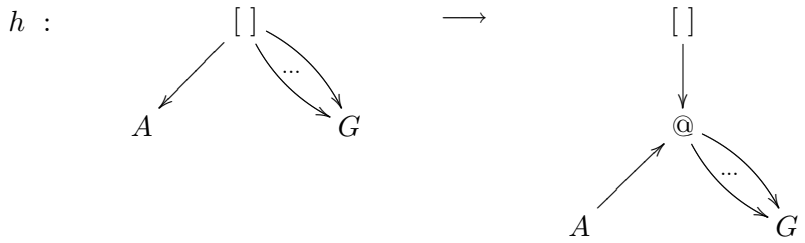
11.2.1 Règles d'Évaluation sous forme de Règles de Réécriture de Graphes avec Ports

Le ρ_{pg}-calcul exprime l'évolution d'un système dont l'état initial est un monde de la forme $\{[A_1 \dots A_n\ O]\}$, avec A_i une abstraction, pour tout i, $1 \leq i \leq n$, $n \geq 0$, et O un objet graphe avec ports. L'interaction réussie entre une abstraction et une molécule graphe avec ports donne naissance à une transformation de graphe avec ports. Nous instancions les règles d'évaluation du $\rho_{\langle\Sigma\rangle}$-calcul (Figure 4.1) avec les molécules graphes avec ports. La résolution du problème de sous-filtrage, ainsi que l'application de la solution du problème de sous-filtrage sont définies à l'extérieur du calcul, en nous basant sur la relation de réécriture de graphe avec ports définie en Section 10.5.

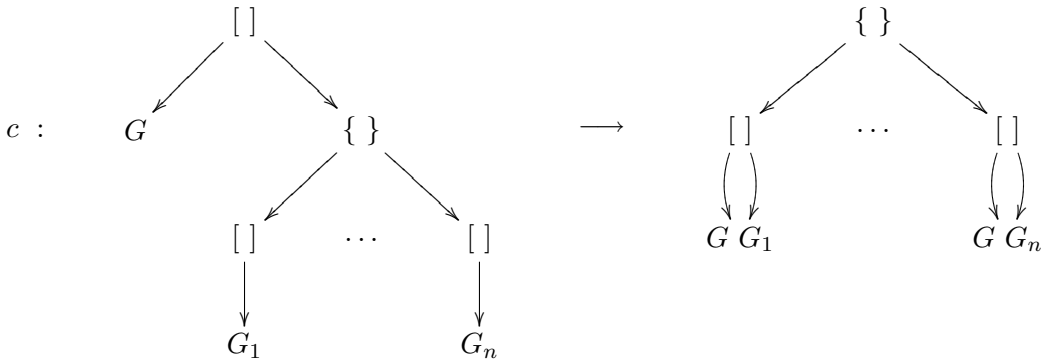
$$(L \Rightarrow R)\ G \longrightarrow \{[\varsigma_1(R)] \dots [\varsigma_n(R)]\} \quad \text{si} \quad \mathcal{S}ol(L \prec\!\!\prec G) = \{\varsigma_1, \dots, \varsigma_n\} \tag{11.1}$$

$$A\ G \longrightarrow A\ G \quad \text{sinon} \tag{11.2}$$

Le règles de réchauffement et de refroidissement ont à présent une représentation basée sur les graphes avec ports. La règle de réchauffement est la règle de réécriture de graphe avec ports suivante :



La règle de refroidissement est représentée par la règle de réécriture de graphe avec ports suivante :



La sémantique du $\rho_{\langle\Sigma\rangle}$-calcul incluant des stratégies qui peuvent être instanciées aisément en utilisant les graphes avec ports et tous les opérateurs auxiliaires pour modéliser l'interaction, correspond aux nœuds avec des ports. En particulier, la construction échec stk est un nœud avec un port poignée.

11.2.2 Mécanisme d'Application sous forme de Règle de Réécriture de Graphe avec Ports

Toutes les étapes calculant l'application d'une abstraction à un objet graphe avec ports, incluant les opérations de (sous)-filtrage et de remplacement, sont exprimables utilisant les graphes avec ports en considérant quelques autres nœuds auxiliaires (nœuds de filtrage, par exemple) et en étendant la relation de réduction avec quelques règles d'évaluation graphiques. Ce mécanisme peut être internalisé au calcul, mais c'est très technique. Nous donnons uniquement l'idée principal de cette encodage, les détails peuvent être trouvés dans la version anglaise de ce document.

L'idée d'utiliser des graphes avec ports pour encoder le calcul lui-même, l'application et les résultats, et à présent pour encoder également l'algorithme de filtrage, a été inspirée par l'encodage du ρ-calcul en utilisant les Réseaux d'Interaction (Interaction Nets) et les Réseaux Bigraphiques (Bigraphical Nets) [FMS06].

Nous considérons deux types de nœuds de filtrage :

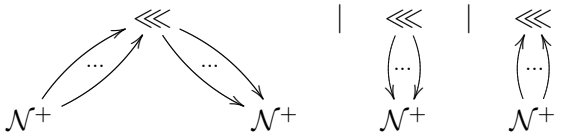
1. $\ll$ pour filtrer deux ports, son port poignée est la cible de l'arête connectant le port motif et, en même temps, est la source de l'arête connecté au port sujet au filtrage ;
2. $\lll$ pour filtrer deux ensembles de ports ; tous les ports du motif sont connectés au poignée de $\lll$, qui est à son tour connecté à tous les ports du sujet.

Quand un nœud de filtrage connecte deux ports poignées dans à la fois le motif et le sujet, alors on dit que nous filtrons des nœuds.

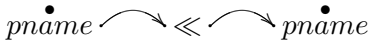
Nous considérons deux nœuds de filtrage différents, puisque nous voulons faire une distinction quand le nœud de filtrage $\lll$ connecte deux singletons. Une condition similaire a également été utilisée dans l'algorithme de filtrage donné pour les graphes avec ports au Chapitre 10 dans la règle de décomposition D_1. Nous ajoutons les constructions de sous-graphes avec ports suivantes à l'ensemble des objets élémentaires du calcul :

Filtrage entre nœuds

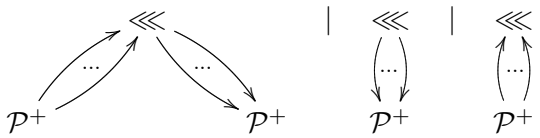
Filtrage entre ensembles de nœuds



Filtrage entre nœuds



Filtrage entre ensembles de nœuds



11.3 Conclusions

Dans ce chapitre, nous avons vu comment le Calcul Biochimique Abstrait est instancié par la structure de graphes avec ports. Cette structure apporte une plus grande flexibilité au calcul en permettant l'encodage d'opérations de méta-niveau telles que le filtrage et le remplacement utilisées dans l'application comme règles d'évaluation. Ceci prouve encore une fois l'universalité de la structure de graphes avec ports, en plus de la possibilité d'encoder une règle de réécriture de graphe avec ports sous forme de graphe avec ports que nous avons montré au Chapitre 10.

Dans [FMS06], les auteurs proposent des réseaux bigraphiques pour encoder le ρ-calcul dans le contexte de la définition de stratégies de réduction efficaces ; en particulier ils utilisent les réseaux de bigraphiques pour modéliser le filtrage dans le ρ-calcul. Un réseau bigraphique est une combinaison entre des réseaux d'interaction et des bigraphes, afin que les contrôles associés aux nœuds aient un port principal distingué et que les interactions agissent entre ports principaux. Alors, un travail futur intéressant concerne l'étude d'un calcul biochimique sur les bigraphes et l'adéquation des réseaux bigraphiques à l'expression du filtrage et du remplacement dans le calcul comme nous l'avons fait pour le ρ_{pg}-calcul.

Conclusion de la Thèse et Travaux Futurs

Conclusion

Dans cette thèse, nous avons défini un calcul biochimique adapté aux besoins de la modélisation des interactions biologiques et des systèmes autonomes, nous avons exploré son expressivité et étudié certaines de ses propriétés. Notre approche diffère du modèle chimique classique par la formalisation à un niveau supérieur et l'utilisation de la réécriture stratégique de graphes pour modéliser les réseaux biochimiques.

Les entités de base du calcul biochimique abstrait (ou $\rho_{\langle\Sigma\rangle}$-calcul) sont (i) des molécules définies sur une structure Σ qui les dote des possibilités de connectivité, (ii) des abstractions sur de telles molécules structurées et (iii) une application d'abstraction. La différence principale du modèle chimique réside dans la structure des molécules et la possibilité de définir une abstraction sur des modèles complexes. D'un point de vue informatique, nous avons conçu ce calcul pour modéliser des interactions concurrentes entre les abstractions et les molécules. Avec l'aide de la réécriture stratégique, quelques interactions peuvent être conçues avec plus de contrôle. Grâce à l'expressivité de l'aspect biochimique du $\rho_{\langle\Sigma\rangle}$-calcul, nous avons codé des stratégies comme objets du calcul.

Nous avons identifié une structure de graphes et l'avons formalisée en tant que graphes avec ports, nous avons défini un algorithme de filtrage correspondant et une relation de réécriture. En parallèle, nous avons fourni un codage pour les graphes avec ports et pour la réécriture de graphes avec ports en utilisant des termes algébriques définis sur une signature de premier ordre et la réécriture de termes. Ensuite nous avons utilisé la structure des graphes avec ports pour instancier le calcul biochimique abstrait. Le résultat est un calcul biochimique basé sur la réécriture sous stratégies de graphes avec ports. Dans le cadre de ce calcul nous avons montré la grande expressivité de la structure de graphes avec ports, par exemple pour exprimer des abstractions et des opérations de méta-niveau comme le filtrage et le remplacement utilisé dans l'application d'une abstraction à une molécule.

Nous avons validé notre modèle biochimique basé sur la réécriture sous stratégies de graphes avec ports, en l'appliquant à l'exemple d'un système autonome et également à un fragment d'un système biochimique réel. L'application au calcul autonome est héritée du modèle chimique d'ordre supérieur et l'application biologique représente une des motivations de cette thèse. Nous avons vu que la structure de graphes avec ports est bien adaptée à la modélisation des deux types d'application et, en outre, la réécriture sous stratégies nous permet de modéliser les propriétés d'auto-organisation des systèmes autonomes et de générer des réseaux biochimiques.

Finalement, nous avons prouvé une fois de plus l'extensibilité et l'expressivité du calcul en incluant une technique de vérification à l'exécution. La motivation principale provient de la modélisation des propriétés auto-réparatrices dans le calcul autonome. Nous avons pu mettre en pratique et intégrer la vérification à l'exécution dans le calcul, grâce à la

capacité du calcul à encoder des stratégies et à manipuler l'échec.

Travaux futurs

Une première amélioration du calcul pourrait être la définition de nouvelles stratégies pour contrôler les interactions entre les molécules. Nous avons brièvement présenté quelques stratégies parallèles possibles dans le Chapitre 7, mais d'autres études devraient être menées pour déterminer comment les encoder et également pour trouver d'autres stratégies. L'équilibre approprié entre les interactions contrôlées et non contrôlées est une question intéressante à adresser pour une application donnée et les systèmes biologiques peuvent nous fournir des intuitions valables.

Sur un plus long terme, le travail sur le calcul présenté dans cette thèse s'adresse au problème de l'adéquation des stratégies pour décrire les systèmes biochimiques et les propriétés d'auto-organisation de systèmes autonomes en général. Nous pensons que l'utilisation des stratégies peut aider des biologistes et des informaticiens à mieux comprendre le comportement de tels systèmes, leurs propriétés d'auto-organisation par exemple, pour en déduire de nouveaux principes d'organisation et de comportement, aussi bien que pour définir de nouveaux modèles informatiques bio-inspirés. À cette fin, nous devrions essayer de modéliser autant d'exemples appropriés de systèmes biologiques que possible et voir si leur comportement peut être décrit en utilisant des stratégies existantes ou si de nouvelles stratégies doivent être définies.

Le calcul que nous avons proposé dans cette thèse est un formalisme qualitatif. Une extension du calcul pourrait être de le doter d'une sémantique quantitative. En particulier pour le ρ_{pg}-calcul il devrait être facile de définir une relation stochastique de réécriture sur les graphes avec ports, puisqu'elle a déjà été étudiée pour les bigraphes [KMT08]. Alors nous devrions également intégrer des stratégies de réécriture stochastiques. Dans le cadre de la modélisation des réseaux biochimiques, la dimension stochastique du modèle nous permettra de dériver des modèles quantitatifs plus réalistes tels que les modèles basés sur des équations différentielles ordinaires ou des modèles stoechiométriques.

Une autre direction intéressante à poursuivre inclut la similitude entre les graphes avec ports gauches et les bigraphes. D'une part nous pourrions enrichir le formalisme de réécriture de graphes avec ports en adaptant les résultats, les propriétés et les applications des bigraphes. D'une part, nous pourrions définir un calcul biochimique basé sur les systèmes réactifs de bigraphes, étudier ses propriétés et expressivité et voir si nous pouvons exprimer l'application d'une règle de réaction de bigraphes par l'intermédiaire des transformations de bigraphes comme nous pouvons le faire avec les graphes avec ports. Également au sujet des structures alternatives pour instancier le calcul biochimique abstrait, nous devrions étudier lesquels des exemples existants inspirés de la littérature sur les systèmes de remplacement de haut-niveau, pourraient fournir au calcul des applications intéressantes.

Il serait très utile d'avoir une implantation du calcul biochimique. Dans une première

phase nous devrions considérer l'exécution du modèle chimique [Rad07]. Dans une seconde étape nous devrions fournir une implantation pour la réécriture de graphes avec ports. Dans cette thèse nous avons défini une sémantique opérationnelle pour la réécriture de graphes avec ports basée sur la réécriture de termes, et nous l'avons mise en application dans TOM pour les graphes moléculaires ; cependant, l'exécution courante n'est pas très efficace pour de grands graphes avec ports. Nous avons déjà quelques idées pour améliorer son efficacité, soit en utilisant une bibliothèque de TOM concernant des termes-graphes, soit en considérant des algorithmes de filtrage plus efficaces pour des graphes généraux ou pour des bigraphes. Notre formalisme basé sur des graphes est visuel, donc assez facile à comprendre. Nous devrions tirer profit de cet aspect en fournissant une interface graphique pour aider à la visualisation des graphes avec ports et de leurs transformations.

Troisième partie

Applications. Implantation. Extensions

12 Case Studies for the ρ_{pg}-calcul

In this chapter we analyze two applications for the ρ_{pg}-calcul. In the first part we show that the extension from higher-order chemical model to the biochemical calculus based on strategic port graph rewriting preserves the good properties of modeling autonomous systems. We illustrate its expressivity for modeling properties of such systems using an example of a mail delivery system. In the second part we instantiate the ρ_{pg}-calcul for modeling interactions between proteins and generating a biochemical network. The motivation of this second case study comes naturally from the biological inspiration model for port graphs.

12.1 Autonomic Computing

Autonomic computing [KC03] refers to self-manageable systems initially provided with some high-level instructions from administrators. This is a concept introduced in 2001 with an intended biological connotation : the simplest biological example is that of the human nervous system where the brain does not need to handle all low-level still vital functions of the body. The four most-important aspects of self-management as presented in [KC03] are self-configuration, self-optimization, self-healing, and self-protection.

This idea of biologically inspired formalism gained much interest with the recent development of large scale distributed systems such as service infrastructures and Grids. For such systems, there is a crucial need for theories and formal frameworks to model computations, to define languages for programming and to establish foundations for verifying important properties of these systems. Several approaches contributed to this ambitious goal. Without exhaustivity, let us mention in particular the brane calculus [Car05b, DP04] and the bigraphical reactive systems [Mil06], but also several formalisms inspired from biology such as [CG00, RPS+04, LT07, QLF+06].

The chemical programming as formalized by HOCL uses the chemical reaction metaphor to express the coordination of computations and it proved to be well-suited for modeling self-organizing and autonomic systems or grids in particular [BFR06a, BFR07].

In this section, we propose port graphs as a formal model for distributed resources. Each resource is modeled by a node with explicit connection points called ports. We model the lack of global information, the autonomous and distributed behavior of components by a multiset of port graphs and rewrite rules which are applied locally, concurrently, and nondeterministically. Moreover strategic port graph rewriting takes into account control on computations by allowing us to chain rewrite rules. By moving from port graph rewriting to the ρ_{pg}-calcul, we are able to express rules and strategies as port graphs and so to rewrite them as well. The ρ_{pg}-calcul also permits the design of rules that create new rules. We apply this formalism for a mail delivery system example borrowed from [BRF04] and

illustrate through this example the additional expressivity brought by strategic rewriting. This work was presented in [AK08d, AK08b].

12.1.1 Strategy-Based Modeling of Self-Management

In autonomic computing, systems and their components reconfigure themselves automatically according to directives (rules and strategies) given initially by administrators. Based on these primary directives and their acquired knowledge along the execution, the systems and their components seek new ways of optimizing their performance and efficiency via new rewrite rules and strategies that they deduce and include in their own behavior. Since there is no ideal system, functioning problems and malicious attacks or failure cascades may occur, and the systems must be prepared to face them and to solve them. In the following, we show how the properties of self-configuration, self-healing, self-protection and self-optimization can be handled by the autonomic mail delivery system inspired from [BRF04] and modeled using the ρ_{pg}-calcul. For this purpose some additional rewrite rules are defined.

In addition to the previously presented concepts, we assume a few more information available in the nodes and ports, which corresponds to existing notions in graph theory. In particular, each port has a degree information that counts the number of incoming and outgoing edges connecting it to other nodes of the object port graph. In our running example, this information allows us to express, through conditions in the rules, that a port is saturated, or on the contrary that it has no incident edge. Graphically, we represent the condition that a port has the incidence degree 0 by a slashed dangling edge connected to the port. If we do not consider the incidence degree information for the ports, the previous condition correspond to a negative application condition in the graph transformation framework [HHT96].

Exemple 13 (A mail delivery system). In order to illustrate our approach and the proposed concepts, we develop the example of a mail delivery system borrowed from [BRF04]. It consists of a network of several mail servers, each with its own address domain ; the clients send messages for other clients first to their server domain, which in turn forwards them to the network and recovers the messages sent to its clients. Servers are distributed resources with connections between them, when sending and receiving the messages.

In Figure 12.1 we illustrate an initial configuration of the mail delivery system. The network is a node with several ports, each port being connected to at most one server. A server node has a handler port for connecting to the network, and several ports for the clients. A client node has a handler port for connecting to a server. All client, server and network nodes have two ports for the incoming and outgoing messages respectively. Messages are nodes with only one port and their names have the form $(rec \ @ \ domain \ \# \ m)$ where rec is the identifier of the recipient client, $domain$ is the identifier of the server domain, and m the body of the message. If redundant, the domain and/or the client identifiers are removed (when arrived in the server domain or at the client). In the system, the server identified by 5 is disconnected from the network node, hence it is in a crashed state.

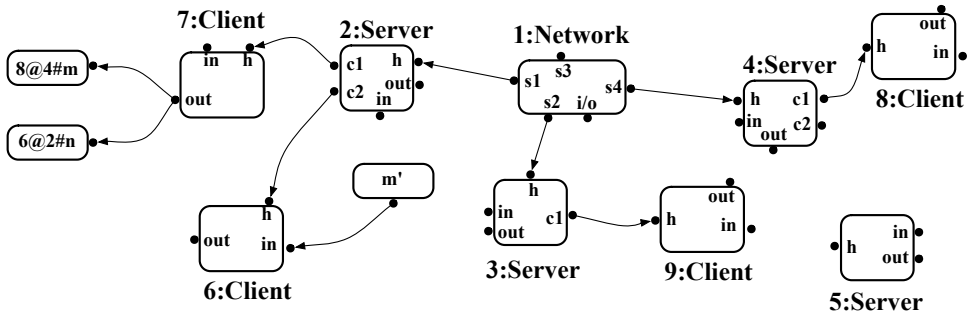


Fig. 12.1: A mail system configuration

The evolution of the distributed system is modeled via port graph transformations, themselves expressed by port graph rewrite rules and the generated rewriting relation. To support intuition, in the mail system example, rules express what happens when a client sends a mail to a client in the same network.

We illustrate in Figure 12.2 the basic rules for the mail system. Since the correspondence between the left- and right-hand sides of the rules are the identities, we simplify the graphs by not detailing the arrow node. A mail sent by a client goes to its server : if the mail is sent to a client in the same server domain then it goes to the input port by

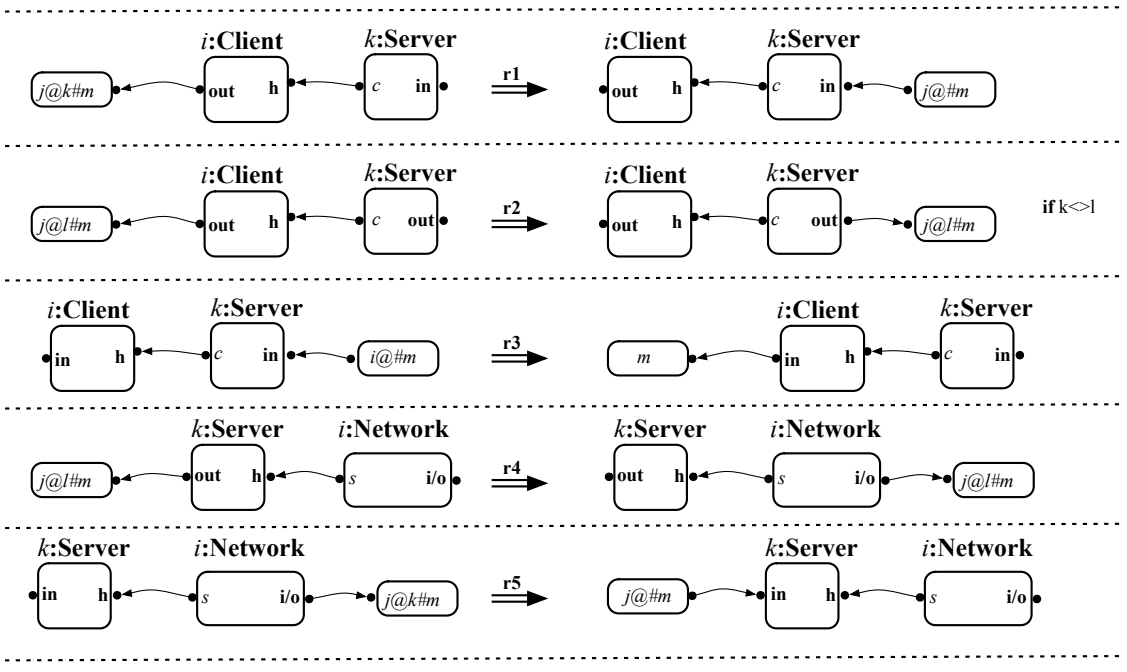


Fig. 12.2: Basic rules for the mail delivery system

r1, otherwise to the outgoing port by r2. By rule r3 a server forwards a mail to a client if he is the recipient. Rule r4 specifies that a server forwards a mail to the network if its recipient is not in the domain, while rule r5 specifies that the network forwards a mail to the appropriate server according to the server domain information contained in the mail.

Self-configuration

The self-configuration is simply described by the concurrent application of the five rules given in Figure 12.2 using the reduction semantics of the ρ_{pg}-calcul.

Rule r6 in Figure 12.3 specifies that a client recipient of a mail telling him to migrate to a server k' does so if the server k' has a free client port. We model a free client port on the server k' by a condition on its degree, specifying that there is no incident edge to that port. In a more visual way, we pictured this condition with a slashed edge. But how to deal with mails that will probably arrive later on the server k for the client i? The problem is solved by introducing in the system a new rule that will update the address of the migrating clients. It is possible that the application of a rewrite rule on a port graph introduces, besides modifying the port graph, a new rewrite rule. In addition, before a client migrates, he must get all mails addressed to him that are already on the server; this is modeled via the strategy first(repeat(r3), r6).

Another interesting problem may concern the operations of replacing a server by two or more servers or, the other way around, replace a group of servers by one server. Then biologically inspired port graph rules from Figure 10.3 (a) and (d) for splitting and merging nodes could be applied as well for servers.

Self-healing

An autonomic system detects when a server crashes and the connection of the crashed server to the network is cut. It is expected to repair the problem of the clients connected to the crashed server and of the mails that were about to be sent from that particular server. Rule r7 creates a temporary server named TServer as a copy of the crashed server and connects it to the network (assuming there is a procedure checking if a server failed). The arrow node of the rule encodes the correspondence of all ports of the server node k, and consequently, all connections with the crashed server are recovered by the temporary server. Note that, for simplifying the graphical representation, we do not include all edges incident to the arrow node, but just the relevant ones.

Since the server replacing the crashed one is temporary, all the clients must try rapidly to connect to other servers which are not fully occupied (rule r8). We graphically express that the server node j has a free client port, i.e., with no incident edges. The mails are forwarded to the network node via the rules r9 and r10. But all this must not happen before sending the mails already there to the clients of the crashed server (rule r11). Instead of simply adding all these rules to the system, we add the strategy r7; repeat(r11); repeat(first(r8, r9, r10)). By composing these rules in a strategy, the recovery from the server crash is assured. A rule should delete the temporary server when it no longer has clients nor pending messages.

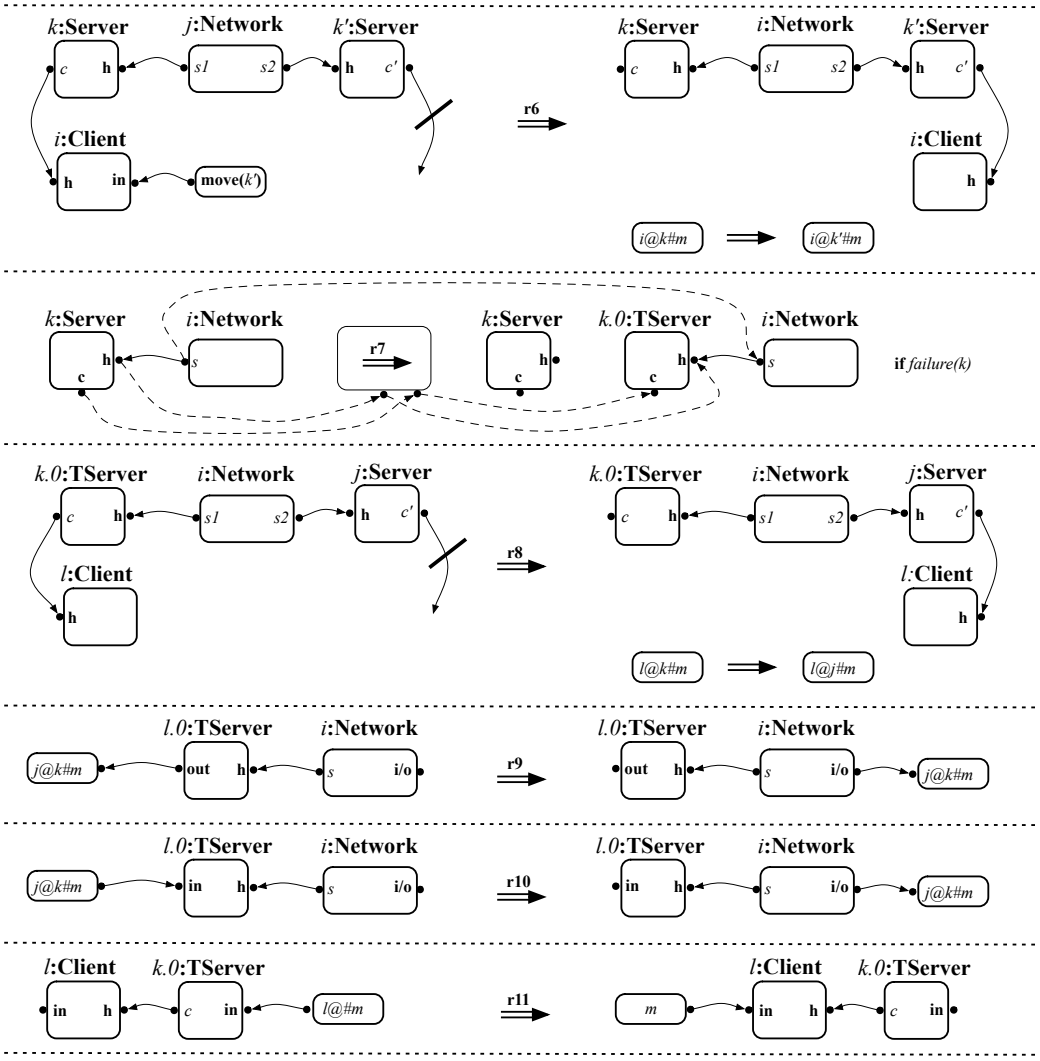


Fig. 12.3: Rules for self-configuration and self-healing in the mail delivery system

Self-protection

When a spam arrives at a server node, the filtering rule r12 deletes it, assuming that the server has a procedure for deciding when a mail is a spam. The rules r13 and r14 are analogous to r12 but for a client node and a network node, assuming as well that both entities have their own spam detection procedure. In order to limit spam sending, the rule r14 should have a higher priority than r5, and the rule r12 a higher priority than r3. Then we replace r3 and r12 by try(r12); r3, and r5 and r13 by try(r14); r5.

When a client receives a mail and, based on a spam decision procedure, concludes that the mail is a spam, it deletes the mail and provides the server with a new rule specifying that from now on the server node should delete all mails of this kind. This behavior is

specified by the rule r15 in Figure 12.4.

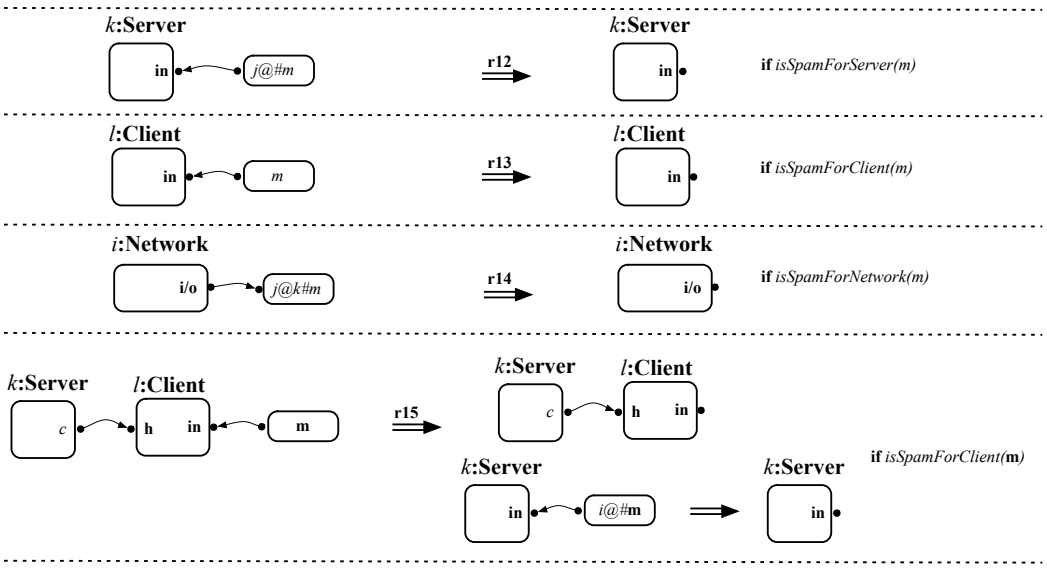


Fig. 12.4: Rules for self-protection in the mail delivery system

Self-optimization

Assuming that a server can determine when it is saturated, i.e., when it reaches the maximal load of incoming messages, a particular type of server for equilibrating the load of the saturated server is created by rule r16. Such an auxiliary server has a handler for connecting to the network node and to the saturated server node, and a port for incoming messages. We call a server with an associated server for equilibrating the load, an optimized server. Then, when the network has a message to dispatch to an optimized server, if the number of incoming messages $in(k)$ on the optimized server is smaller that the incoming messages $in(k.0)$ on its associated server, then the message goes to k (rule r17), else it goes to $k.0$ (rule r18). Since an auxiliary server is created due to an overload of incoming messages, it is obvious that the next message(s) from the network node will be dispatched to the auxiliary node ; hence rule r18 will be executed before rule r17, which is expressed by the strategy first(r18, r17). A message is dispatched from $k.0$ to k (rule r19) when $in(k) < in(k.0)$. If an optimized server fails, then the rule r20 creates a temporary server similarly to rule r7 which in addition recovers all messages on the auxiliary server which it deletes.

12.1.2 Towards Embedding Runtime Verification in the Model

We have shown in the previous section how a particular autonomic system can be modeled using the ρ_{pg}-calcul. The model should also ensure formally that the intended

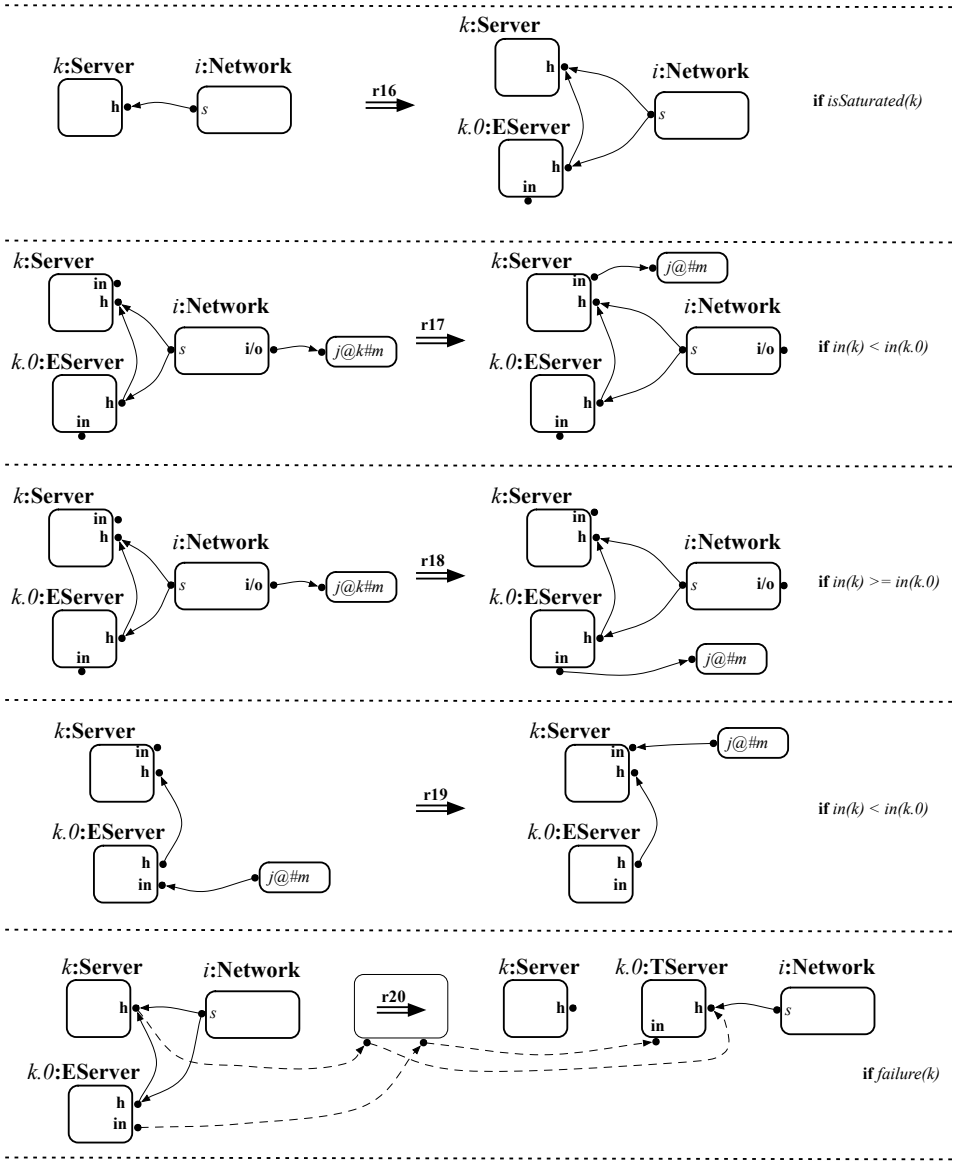


Fig. 12.5: Rules for self-optimization in the mail delivery system

self-managing specification of the system helps indeed preserving the properties of the system. Some properties can be verified by checking the presence of particular port graphs and we can easily encode them as object port graphs, abstractions, or strategies, hence as entities of the ρ_{pg}-calcul. Consequently, the properties can be placed at the same level as the specification of the modeled system and they can be tested at any time.

One possibility of embedding verification in the model consists in associating a recovery strategy for each strategy modeling a behavior of the system. Let $recovery(S)$ denote the recovery strategy for S. Then instead of S we have $\text{first}(S, recovery(S))$. If the recovery

strategy is id then we obtain try(S) which avoids a failure, whereas using fail as recovery strategy does not change the failure if the strategy S fails. Of course, what needs to be done is to determine which are the recovery strategies for each strategy of the system.

Another possiblity of embedding verification in the model consists in expressing an invariant of the system as a port graph rewrite rule with identical sides, $G \Rightarrow G$, testing the presence of a port graph G. The failure of the invariant is handled by a failure port graph STK that does not allow the execution to continue. The strategy verifying such an invariant is then :

$$\text{first}(G \Rightarrow G, X \Rightarrow \text{STK})!$$

For instance in our running example, this strategy is useful to ensure the persistency of a given critical server of the network, or may be used also to check that there is always a minimal number of servers available in the network. From another perspective, we express the unwanted occurrence of a object port graph G in the system using the strategy :

$$(G \Rightarrow \text{STK})!$$

In both cases above, instead of yielding the failure STK signaling that a property of the system is not satisfied, the problem can be "repaired" by associating to each property the necessary rules or strategies to be inserted in the system in case of failure. Such ideas need to be further explored since they open a wide field of possibilities for combining runtime verification and self-healing in ρ_{pg}-calcul. In particular, we show in Chapter 14 how we can increase the expressivity of the ρ_{pg}-calcul by embedding a particular set of formulas in a suitable temporal logic to the syntax and adjusting correspondingly the reduction semantics in order to verify such formula in parallel with the evolution of the modeled system.

12.2 Molecular Graphs. Biochemical Networks

Port graphs provide a modeling formalism for molecular complexes by restricting the connectivity of a port (called site in the biological model) to at most one other port ; we call such restricted port graphs molecular graphs. We instantiate the Abstract Biochemical Calculus with the structure of molecular graphs to obtain a Calculus of Molecular Graphs (CMG) [AK08a]. The first-citizens of the CMG are molecular graphs, molecular graph rewrite rules, and their interactions.

In [AK07], we already introduced the basic ideas of the graph rewriting and strategic rewriting for modeling biochemical networks. We also gave an encoding via term rewriting which provided us for free a term rewriting calculus for biochemical systems. In the same paper, we illustrated the biological motivation for using this formalism with an example based on the epidermal growth factor receptor (EGFR) signaling pathway. With respect to the result of this preliminary work, in this section we present the calculus of molecular graphs where the rules and the strategies are first-class elements. We also illustrate its expressive power provided by its higher-order features that may capture behaviors of systems capable of self-management. However further work is needed to understand whether this corresponds to actual biological models.

12.2.1 Modeling Molecular Complexes as Port Graphs

The behavior of a protein is given by its functional domains that determine which other protein it can bind to or interact with. These domains are usually abstracted as sites that can be bound or free, visible or hidden. A protein is characterized by the collection of interaction sites on its surface. Proteins can connect at specific sites by low energy bounds forming molecular complexes. A biochemical system is represented as a discrete system consisting of interacting components which give rise to structural and behavioral transformation of the components and of the system as a whole. Such a system is dynamic, has an emergent behavior, is highly concurrent and non-deterministic.

In the following we show how molecular complexes can be represented by some particular graphs, and how their interactions can be modeled by graph rewrite rules and a rewriting relation on such graphs.

Molecular Graphs

We represent molecular complexes as a particular class of port graphs. For a given biological model, we can extract a p-signature by associating to each protein name its site names.

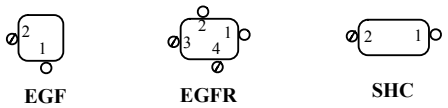
Définition 55 (Molecular graph). A molecular graph is a port graph over a p-signature whose ports have states and each port can be the endpoint of at most one edge. The ports are called sites and the edges bonds. Graphically, the state of a site is represented as a filled circle for bound, an empty circle for free, and a slashed circle for hidden.

Exemple 14. We consider a fragment of the EGFR signaling pathway, an example oftenly studied by various formalisms, for instance the κ-calculus [DL04, LT07]. The protagonists of this model are :

- the signal protein EGF situated outside the cell acting as a ligand,
- the transmembrane protein EGFR with two extracellular sites and two intracellular sites as a receptor, and
- the adapter protein SHC situated inside the cell.

Using the same graphical representation as for port graphs, we represent a protein as an empty box having the identifier placed at the exterior and the sites as small points on the surface of the box.

Then the three types of proteins above are represented graphically as below :



In Figure 12.6 we illustrate in the left side a molecular graph representing the initial state of the system that will be studied throughout this section. The molecular graph

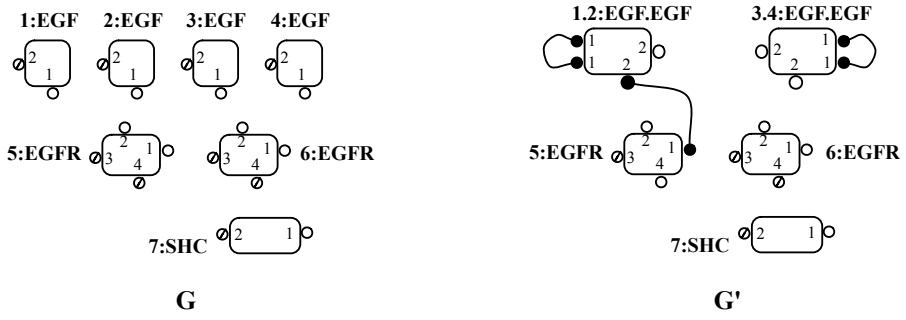


Fig. 12.6: Initial and intermediate molecular graphs in the EGFR model

in the right side represents an intermediary state where two signal proteins are already bound forming a dimer which in turn binds to a receptor.

Rewriting Molecular Graphs

Définition 56 (Molecular graph rewrite rule). A molecular graph rewrite rule is a port graph rewrite rule where the left- and right-hand sides are molecular graphs. A molecular graph rewrite systems is a finite set of molecular graph rewrite rules.

We note that a molecular graph rewrite rule is not a molecular graph, but a port graph, since the arrow node does not satisfy the constraint of the maximum one incidence degree for its ports.

In the case of molecular graph rewrite rules we represent the edges incident to the arrow node only if the correspondence it embeds is ambiguous.

Exemple 15. In Figure 12.7 we present the molecular graphs rewrite rules for the EGFR signaling pathway :

(r1) two signaling proteins form a dimer represented as a single node ;
(r2) an EGF dimer and a receptor bind on free sites ;
(r3) two receptors activated by the same EGF dimer bind creating an active dimer RTK ;
(r4) an active dimer RTK activates itself by attaching phosphate groups ;
(r5) an activated RTK binds to an adapter protein activating it as well.

The rewriting relation induced by a set of molecular graph rewrite rules is similar to the port graph rewrite relation up to the constraints imposed on molecular graphs as particular port graphs. In a similar way, the strategic molecular graph rewriting is defined based on the strategic port graph rewriting.

Exemple 16. We illustrate in Figure 12.8 two possible results of applying the rewrite rule r2 on the molecular graph G'.

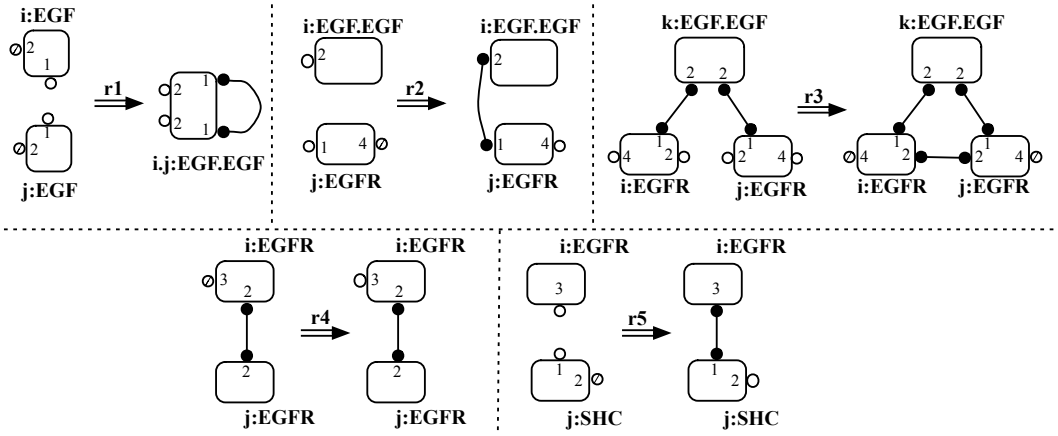


Fig. 12.7: The reaction patterns in the EGFR signaling pathway fragment

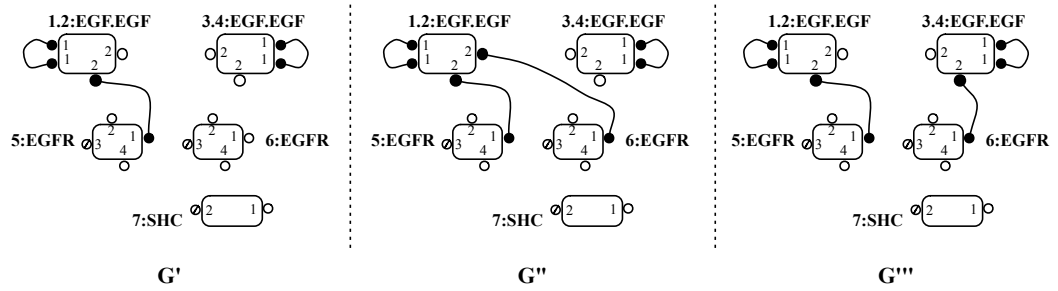


Fig. 12.8: G" and G"' are obtained from G' by rewriting using the rule r2 on different subgraphs

12.2.2 Biochemical Network Generation by Strategic Rewriting

Strategic rewriting is a suitable formalism for modeling the highly concurrent and non-deterministic behavior of complex systems in general. Strategies were already used for a chemical application, more precisely for modeling automated generation of the kinetic mechanism in the GasEl project [BCC+03, BIK06, Iba04] in an ELAN-based implementation and then in a TOM-based implementation. A first link between strategies and computation models inspired by biology is presented in [ACL06] where highly parallel control mechanisms in membrane systems are expressed by means of rewrite strategies.

A biochemical system as a particular complex system is not completely described by its components and the way they interact by means of reactions, but also by the behavior of the system as a whole. Modeling the generation of a biochemical network amounts to defining how a set of reaction patterns is applied on a collection (set or multiset) of molecules. Strategic rewriting provides a formal model for expressing the control on the reaction rule application.

Usually, given an initial molecular graph describing the structure a biological system and the set of reaction patterns (as rewrite rules) describing its behavior, the correspon-

ding biochemical network is constructed by repeatedly applying the reaction patterns to every state of the system until no new states are obtained or a termination condition is satisfied.

Exemple 17. In the calculus of molecular graphs, for the system corresponding to the EGFR pathway fragment described in the previous examples, the initial state of the is a world consisting of the molecular graph G and several strategies built upon the five reaction rules. G can be also written as the following juxtaposition of molecular graphs (or nodes in this particular situation) :

$$1 :\text{EGF } 2 :\text{EGF } 3 :\text{EGF } 4 :\text{EGF } 5 :\text{EGFR } 6 :\text{EGFR } 7 :\text{SHC},$$

We see in the following a few examples of strategies for generating the biochemical network. The most straightforward way of modeling the biochemical network for the EGFR signaling pathway fragment presented here is to consider the juxtaposition of the reaction rules as persistent strategies. Hence the initial state of the system is the simple world [r1! r2! r3! r4! r5! G]. Then any of the five rules is applied exhaustively. We can easily prove that the system will reach a stable state since the number of free binding sites decreases or remains constant with every successful rule application.

The strategy first can be used to specify a higher priority in the application of two rules ; for instance, $\text{first}(\text{r}_2, \text{r}_1)$ is saying that an EFG dimer binds a receptor as soon as it is created. Then the initial state is [first(r2, r1)! r3! r4! r5! G]. We can slightly modify this state such that r3 is not persistent : [first(r2, r1)! r3 r4! r5! G]. This means that the rule r3 is consumed when creating the active dimer RTK ; having only one instance of such rule allows the creation of only one active dimer RTK.

The execution can be separated in two stages : the first one is concerned with the extracellular interactions between the signals and the receptors, hence the reactions r1, r2 and r3, while the second one with the RTK pathway, hence the reactions r4, r5. If we consider that r2 has a higher application priority over r1, and any reaction from the first stage has priority over any reaction from the second stage, then the initial world of the system is :

$$[\text{first}(\text{first}(\text{r2}, \text{r1}), \text{r3}) \;\; \text{first}(\text{first}(\text{r2}, \text{r1}), \text{r4}) \;\; \text{first}(\text{first}(\text{r2}, \text{r1}), \text{r5}) \;\; \text{G}]$$

For every such initial state, the interactions take place non-deterministically and concurrently, and all will reduce to a state of equilibrium where no more rules can be applied. For each of the states above, an equilibrium state contains the molecular graph H given in Figure 12.9 or G"' from Figure 12.8.

Modeling Self-Management Properties of Biological System

In this paragraph we simply suggest some capabilities made available by the expressive power of the calculus of molecular graphs. They need to be further explored in the context of biological systems.

The occurrence of a molecular graph pattern P in the state of the system may indicate that a specific action should be included in the system behavior along with a possible

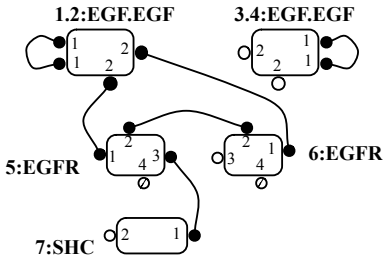


Fig. 12.9: The molecular graph H for the EGFR signaling pathway fragment in the equilibrium state

change of the molecular graph pattern. Such adaptive situation can be easily specified in the calculus by an abstraction $(P \Rightarrow G'\ A)$ that creates a new abstraction A to handle the situation. For instance, if a virus characterized by a pattern PV appears in the system, then the application of the abstraction PV $\Rightarrow A$ will delete the virus and introduce an abstraction to repair the possible damage the virus produced.

Another high-level capability of the calculus is represented by second-order abstractions which can transform abstractions on molecular graphs ; for instance, at a certain moment all such abstractions involving a particular protein must be changed to take into account a mutation of the protein.

12.2.3 Comparisons with Related Formalisms

Using graph rewriting and rule-based formalisms for modeling biological systems has already been done by several authors. A review of such rule-based formalisms can be found in [HFB+06] with emphasis on the capability of representing the topology of complexes.

An inspiring starting point for our work was the graphical formalism BioNetGen presented in [BYFH06] for modeling biochemical networks where the protein complexes are represented by typed attributed graphs, and classes of reactions are modeled by graph transformation rules. This model considers also quantitative information on reactions. Our approach, developed on a rule-based modeling framework and extensively using expressive graphical representations (like the one of [BYFH06]), focuses on providing in addition a strategic rewriting dimension and higher-order capabilities. These two aspects allow a flexible modeling for the generation of biochemical networks.

The κ-calculus [DL04] is a language of formal proteins which models complexes as graphs-with-sites and their interactions as a particular graph-rewriting operation. The bonds are represented in complexes by shared label names. A reaction is restricted to at most one complexation or decomplexation. Such a requirement is quite restrictive, but expressive enough to allow non-linear reaction like modeling the synthesis or degradation of proteins. The algebraic notation used for the κ-calculus is based on the π-calculus [Mil99] rather that on graphs in order to express the combinatorics of the interactions between proteins. A step forward, bioκ-calculus [LT07] combines the κ-calculus with the brane calculi [Car05b] providing a more expressive formalism by modeling the effects of protein

interactions on the interaction capabilities of membranes.

The two formalisms BioNetGen and the κ-calculus are both based on a visual graph representation and on a rule-based semantics as the calculus we propose. We use the same approach to get a new calculus that differentiates itself through the control defined by strategies and the higher-order capabilities.

Pathway Logic [EKL+04] is a rewriting system formalism, where proteins and cells are modeled by algebraic terms, and reactions by term rewrite rules. It is designed to work on two levels of abstraction, one concerning the protein states, and the other concerning the protein-protein interactions handled by means of graph rewriting. The Maude[1] [CDE+02] system is used for implementing Pathway Logic, providing executability of the specifications and analytic tools. The first-order term encoding for port graphs defined in Chapter 13 is close to the Pathway Logic approach to use algebraic terms and rewrite rules for modeling molecules and reactions respectively. There is a difference in the representation of graphs : we use a set of nodes and a set of adjacency lists, whereas in Pathway Logic the graph representation is based on a set of nodes and a set of edges.

We also provide a term-rewriting implementation using TOM[2] based on a very similar idea used for the encoding in Chapter 13. In addition, we use the pointers introduced in TOM for rewriting term-graphs [BM08] ; this way each bound site (or port) in the interface of a protein has an associated pointer to the node and site it is connected. The pointers ensure the well-formedness of graphs since the condition of no dangling edges is equivalent to the condition of no dangling pointers. We also benefit from the rewrite strategies in TOM for expressing the biochemical network generation as well as to express more complex rules. We give details of the implementation in Appendix A.

In addition to the formalism introduced by Pathway Logic for modeling protein interactions, we propose a higher-order calculus which permits the expression of control of rewrite rule applications inside the calculus.

While on the one side the ρ_{pg}-calcul is capable of modeling interactions between biochemical entities like proteins, on the other side we are obviously able to model as well chemical reactions (like the ones in [AIK06]) : atoms represent the nodes, the valence of an atom gives the number of identical ports, and chemical bonds between atoms are edges.

12.3 Conclusions

In the first part of this chapter we saw that the biochemical calculus instantiated for the structure of port graphs exhibits, as expected, the same expressivity power for modeling autonomous systems as the higher-order chemical language. HOCL is a well-suited model of computation for the specification of complex computing infrastructures such as Grids or large autonomous systems. As a future direction, we should consider the expressivity the ρ_{pg}-calcul can provide by considering a port graph structure for the resources and by controlling the order of interactions.

[1] http://maude.cs.uiuc.edu
[2] http://tom.loria.fr

In the second part of the chapter, we showed that the ρ_{pg}-calcul is not only a biologically-inspired calculus but it is well-suited for modeling biochemical networks. We described the evolution of a biochemical system using several "different" strategies. The plurality of such strategies raises questions on the possibility of defining a comparison relation on strategies with the aim of finding the smaller, canonical or most efficient strategy between two strategies. In order to handle these problems one has to make intensive use of theoretical tools and skills in rewriting theory. A starting point for approaching this problem is the simplification process of strategies introduced in [FGK03] concerning the termination problem for strategic rewriting. The usefulness of solving such problems, both from the theoretical and application point of view, makes this direction interesting for future study.

Strategic rewriting opens many possibilities for modeling different aspects of biological systems at different abstraction levels. In particular, we plan to explore the capabilities of this formalism to model and reason about the adaptability and flexibility of cell behavior. We do not limit our aim to modeling some well-known biological systems, but to help understand their behavior and deduce new organization and behavioral principles. In the same vein as Păun in [Pau06], we consider that reasoning at the level of strategies of computing (rewrite strategies), rather than at the tactic level (rewrite rules), is an incentive direction of formally studying biological systems.

We focused in this section on interactions between proteins at the level of functional domains. However, the model we propose can be easily tuned to represent other types of biomolecular interactions, such as protein-DNA or protein-lipid interactions [FBH05].

13 Term Rewriting Semantics for Port Graph Rewriting

In this chapter we define a function encoding port graphs as algebraic terms, port graphs rewrite rules as term rewrite rules, and the port graph rewrite relation defined in Chapter 10 as a term rewrite relation. The motivation of this encoding is to obtain an implementable operational semantics for port graph rewriting.

An axiomatization of labeled graphs as algebraic data types was introduced in [Mes96] in the context of presenting the Rewriting Logic, and its implementation Maude in particular, as a semantic framework for different models of concurrency. Using an object-oriented approach, a node is an object with a name (or identifier) and two attributes : the first one for the data element representing the label, and the second one for the adjacency list consisting of the the immediate neighbors in the graph. The nodes of a graph must fulfill two requirements : different node objects must have different node names (as usual for any object-oriented system), and the node names occurring in the adjacency lists of objects must be present in the graph (to avoid dangling pointers). For the more delicate problems of node creation and node deletion, some solutions are presented. For node creation the solution presented considers creating fresh names by conveniently appending numbers to the node names. Whereas for node deletion, a new attribute is added to each node object which counts the references to itself ; then a node with zero reference count becomes garbage, and then deleted. However, this solution is not always appropriate since it assumes that a graph cannot contain a disconnected node.

As we will see in this chapter, for encoding a port graph we separate the set of nodes from the set of edges grouped in adjacency lists. However, we could as well consider to include the adjacency list in the nodes, but the representation would then be quite complicated for a node with many ports and many outgoing edges.

Concerning the node creation, we consider the same solution as the one presented in [Mes96] to preserve the uniqueness of the identifiers of the nodes. Related to the problem of dangling edges, we have showed in Chapter 10 how in a port graph rewrite rule each node from the left-hand side deleted by the rule is mapped to the empty set of nodes. For the term encoding we introduce a special node $\bullet$ called the black hole (or sink) and we replace a deleted node in an intermediate step by a black hole. The behavior of a black hole consists in deleting itself along with the incident edges. In a similar way we use the black hole to replace in an intermediary step a deleted port as well. Hence, the dangling edges are deleted using some particular intermediate and transparent operation as we will see later in this chapter.

This chapter is structured as follows. We start by defining in Section 13.1 an algebraic signature and the encoding of port graphs as terms over this signature with appropriate

conditions for well-formedness and canonical form. In Section 13.2 we encode port graph rewrite rules as term rewrite rules. We prepare the definition of the encoding of the port graph rewriting relation by extending the term rewrite rules with variables in order to handle the context within a rule is applied (Section 13.3) and by defining some auxiliary operations and reduction relations necessary for in the replacement process (Section 13.4). In Section 13.5 we define the encoding of the port graph rewriting relation and we prove it correct and complete with respect to the port graph rewriting relation in Section 13.6. In Section 13.7 we embed the term approach on port graph rewriting in the rewriting calculus obtaining a term rewriting calculus for port graphs.

13.1 Term Encoding of Port Graphs

13.1.1 An Algebraic Signature for Port Graphs

We define an order-sorted signature $\Sigma = (\mathcal{S}, <, \mathcal{F})$ for encoding port graphs. In order to eliminate redundancies, we represent a port graph as a pair consisting of the set of node labels and the set of adjacency lists instead of the sets of nodes labels and edges. An adjacency list for a node is the list of its neighbors with the corresponding edges as pairs of ports.

The sort set $\mathcal{S}$ consists of sorts for each component or set of components needed for encoding a port graph as an algebraic term :

$$\mathcal{S} = \{\text{Id, Name, Port, Node, Edge, Neighbor, AdjacencyEq, PortSet, NodeSet, EdgeSet, NeighborSet, AdjacencyEqSet, PGraph}\}$$

The subsort relation is defined by X $<$ XSet for X $\in$ {Port, Node, Edge, Neighbor, AdjacencyEq}, i.e. each term of sort X can be seen as a set with a single element.

The set of operation symbols $\mathcal{F}$ allowing to describe the port graph structure is given in Figure 13.1 where X takes sort values from the set {Node, Edge, Neighbor, AdjacencyEq}. The associative-commutative operator _ , _ (union) is overloaded on each of the set sorts, and ϵ_{X} denotes the identity element (the empty set) for the union operation _ , _ on the sort X. We use ϵ instead of ϵ_{X} whenever the sort X can be easily deduced from the context. The constant operator $\bullet$ is overloaded as well, it can be an Id-, a Port- or a Node-sorted term.

13.1.2 A Term Algebra for Port Graphs

Let $\mathcal{X}$ be an $(\mathcal{S}, <)$-sorted family of variables. Let $\mathcal{T}_\Sigma$ and $\mathcal{T}_\Sigma(\mathcal{X})$ denote the algebra of ground terms and the algebra of terms with variables in $\mathcal{X}$ respectively generated by the signature Σ.

Définition 57 (Encoding port graphs as terms). We encode a port graph $G = (V, E)$ as an algebraic term $\mathcal{E}(G) = T_1(\!|T_2|\!)$ of sort PGraph where :

– $T_1 \in \mathcal{T}_{\Sigma,\text{NodeSet}}(\mathcal{X})$ represents the set of all node labels in G, and

$\bullet$: $\longrightarrow$ Id	$\bullet$: $\longrightarrow$ Port	$\bullet$: $\longrightarrow$ Node $\quad \epsilon_{\mathrm{X}}$: $\longrightarrow$ XSet
$\_,\_$	:	XSet XSet $\longrightarrow$ XSet $[ACU(\epsilon_{\mathrm{X}})]$
$\langle \_ : \_ \parallel \_ \rangle$	:	Id Name PortSet $\longrightarrow$ Node
$(\_,\_)$	:	Port Port $\longrightarrow$ Edge
$\_ \frown \_$	:	Id EdgeSet $\longrightarrow$ Neighbor
$\_ \doteq \_$	:	Id NeighborSet $\longrightarrow$ AdjacencyEq
$\_(\!\|\_\|\!)$	:	NodeSet AdjacencyEqSet $\longrightarrow$ PGraph

Fig. 13.1: The operation set $\mathcal{F}$

– $T_2 \in \mathcal{T}_{\Sigma,\mathrm{AdjacencyEqSet}}(\mathcal{X})$ is the set of adjacency equations providing the neighbors for each node in V (if any) and the pairs of ports corresponding to the incident edges.

Exemple 18. The port graph G illustrated in Figure 10.1 is encoded as the following term :

$$\begin{aligned}\mathcal{E}(G) \quad = \quad & (\langle 1 : A \parallel a, b, c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a, b, c\rangle, \langle 4 : C \parallel d\rangle)(\!| \\ & 1 \doteq (2 \frown (a, e), (b, e)), (3 \frown (b, a))), \\ & 2 \doteq \epsilon, \\ & 3 \doteq 4 \frown (a, d), \\ & 4 \doteq 1 \frown (d, c) \; |\!)\end{aligned}$$

Additionally, algebraic terms encoding port graphs must satisfy two structural properties in order to be considered well-formed. These two properties are also mentioned in [Mes96] for the axiomatizating graphs as algebraic data types.

Définition 58 (Well-formed terms). A term $t \in \mathcal{T}_{\Sigma,\mathrm{PGraph}}(\mathcal{X})$ is well-formed if :
– each node identifier occurs at most once in the node set, in the adjacency equation set as left-hand side of an adjacency equation, in the neighbor set of a node identifier ;
– each node identifier or port occurring in the adjacency equation set must also occur in the node set (i.e., there should be no "dangling edges").

We also impose a canonical form (a representative of each equivalence class modulo ACU) for the terms encoding port graphs, in order to eliminate useless information as follows :

Définition 59 (Canonical form). A term $t \in \mathcal{T}_{\Sigma,\mathrm{PGraph}}(\mathcal{X})$ is in canonical form if :
– the right-hand sides of adjacency equations are non-empty sets of neighbors ;
– only non-empty sets of edges occur in neighbor terms.

Exemple 19. The term in canonical form corresponding to G is the following :

$$\begin{aligned}& (\langle 1 : A \parallel a, b, c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a, b, c\rangle, \langle 4 : C \parallel d\rangle)(\!| \\ & \quad 1 \doteq (2 \frown (a, e), (b, e)), (3 \frown (b, a))), \\ & \quad 3 \doteq 4 \frown (a, d), \\ & \quad 4 \doteq 1 \frown (d, c) \; |\!)\end{aligned}$$

13.2 pg-Rewrite Rules

For all rewrite rules over $\mathcal{T}_{\Sigma,\mathrm{PGraph}}(\mathcal{X})$, according to Definition 45, we impose node identifiers occurring in the left-hand side to be variables. We say that a rewrite rule over $\mathcal{T}_{\Sigma,\mathrm{PGraph}}(\mathcal{X})$ is well-formed if both t_1 and t_2 are well-formed. We call pg-rewrite rule a well-formed rewrite rule over $\mathcal{T}_{\Sigma,\mathrm{PGraph}}(\mathcal{X})$.

Définition 60 (Encoding port graph rewrite rules as term rewrite rules). Given a labeled (weak) port graph rewrite rule $L \rightsquigarrow R$, we encode it as a term rewrite rule $\mathcal{E}(L \rightsquigarrow R) = t \twoheadrightarrow t'$ with t and t' the canonical forms of $\mathcal{E}(L)$ and $\mathcal{E}(R)$ respectively.

The encoding of a (weak) port graph rewrite rule is a pg-rewrite rule since, by definition, the term encoding a port graph is well-formed.

The node-morphism from nodes of the left-hand side to nodes of the right-hand side of a pg-rewrite rule can be extracted automatically by means of an analysis on each identifier occurrence. We call this procedure GetMap. It produces a set of elementary mappings from $\mathcal{T}_{\Sigma,\mathrm{Node}}(\mathcal{X})$ to $\mathcal{T}_{\Sigma,\mathrm{NodeSet}}(\mathcal{X})$ for each node occurring in the left-hand side of the rule. Identity mappings are usually omitted. The node-morphism for a port graph rewrite rule is encoded as in the following example.

Exemple 20. The encoding of the port graph rewrite rule (a) given in Figure 10.3 is :

$$(\langle i : X \parallel x, z\rangle, \langle j : Y \parallel y\rangle)(\!|i \doteq j^\frown(x, y), (z, y)|\!) \twoheadrightarrow$$
$$(\langle i.1 : X.1 \parallel x\rangle, \langle i.2 : X.2 \parallel z\rangle, \langle j : Y \parallel y\rangle)(\!|(i.1 \doteq j^\frown(x, y)), (i.2 \doteq j^\frown(z, y))|\!)$$

with the node-morphism

$$\begin{aligned} \xi \quad = \quad & \{\langle i : X \parallel x, z\rangle \mapsto (\langle i.1 : X.1 \parallel x\rangle, \langle i.2 : X.2 \parallel z\rangle)), \\ & \langle j : Y \parallel y\rangle \mapsto \langle j : Y \parallel y\rangle\} \end{aligned}$$

13.3 Extending the pg-Rewrite Rules

After encoding port graphs and their transformation rules, we continue with some preparatory steps before encoding the port graph rewriting relation into a term rewriting relation.

In a first step we customize the rewrite rules on $\mathcal{T}_{\mathrm{PGraph}}(\mathcal{X})$ before applying them. In order to model port graph rewriting using algebraic terms, we need to handle the context of the port graph in which the replacement is performed. This is done by a systematic enrichment of rewrite rules with extension variables that help storing the context and applying rewrite steps in subterms. This is a usual method employed when performing rewriting modulo associativity and commutativity [KM01]. For instance, let us consider two constant operators a and b, and a set operator $\_|\_$, i.e., associative and commutative with neutral element. Then a rule $a \to b|b$ is extended to $a|X \to b|b|X$ with X a set variable such that it can be applied on different subterms of a non-constant term, like $a|b|b|a|b|a$.

In the context of encoding port graphs and rewrite rules over port graphs, we usually denote by W an extension variable and by $\overline{t}$ the extension of term t. For each rewrite rule $t_1 \rightarrowtriangle t_2$, extension variables are appended to set-sorted terms to produce the extended rule $(\overline{t_1 \rightarrowtriangle t_2})$. An extension variable is added to each set-sorted subterm in the left-hand side (and accordingly in the right-hand side).

Exemple 21. The extension of the rule given in Example 20 is :

$$(\langle i : X \parallel x, z, W_1^p, W_2^p\rangle, \langle j : Y \parallel y, W_3^p\rangle, W_4^n)(\!|(i \doteq (j^\frown((x,y),(z,y),W_5^e)), W_6^h), W_7^a|\!) \rightarrowtriangle$$
$$(\langle i.1 : X.1 \parallel x, W_1^p\rangle, \langle i.2 : X.2 \parallel z, W_2^p\rangle, \langle j : Y \parallel y, W_3^p\rangle, W_4^n)(\!|(i.1 \doteq j^\frown(x,y)),$$
$$(i.2 \doteq j^\frown(z,y)), (i \doteq (j^\frown W_5^e), W_6^h), W_7^a|\!)$$

with the node-morphism $\xi = (\langle i : X \parallel x, z, W_1^p, W_2^p\rangle) \mapsto \langle i.1 : X.1 \parallel x, W_1^p\rangle, \langle i.2 : X.2 \parallel z, W_2^p\rangle)$, $(\langle j : Y \parallel y, W_3^p\rangle) \mapsto \{\langle j : Y \parallel y, W_3^p\rangle)$ where the extension variables W_i have appropriate set sorts. The exponents used for the extension variables indicate their set sort : p for PortSet, n for NodeSet, e for EdgeSet, h for NeighborSet, a for AdjacencyEqSet.

13.4 Auxiliary Operations and Reduction Relations

In this section we define some operations necessary for the definition of the rewriting relation.

Instantiation of a Node-Morphism

Let σ be a substitution and ξ a node-morphism. We denote by ξ^σ the instantiation by σ of variables occurring in ξ computed component-wise :

$$\{t^i \mapsto t_1^i, \ldots, t_{k_i}^i \mid i \in \mathcal{I}\}^\sigma = \{\sigma(t^i) \mapsto \sigma(t_1^i), \ldots, \sigma(t_k^i) \mid i \in \mathcal{I}\}$$

Node-Morphism Application

The application of a node-morphism ξ to a term encoding a port graph as we have introduced in in Definition 41 consists in applying sequentially each elementary node mapping of ξ on the term. This is achieved by reducing the term $\xi(N(\!|A|\!))$ using the term rewrite system $\mathcal{A}$ from Figure 13.2. In the following we explain each rule from $\mathcal{A}$. An elementary node mapping is propagated inside an PGraph-sorted term using the rule (Propagate), inside the set of adjacency equations of neighbors using the rule (Distribute), and then applied on each of them. The application of a node-morphism on an adjacency equation using the rule (ApplySrc) (or a neighbor using the rule (ApplyTar)) transforms it in n adjacency equations (neighbors respectively), one for each corresponding node in the right-hand side of the mapping, and propagates the node-morphism application on the set of neighbors. We illustrate the node-morphism application in Example 22.

i : Id; $t, t_1, \ldots, t_n$: Node; N, T : NodeSet; A : AdjacencyEqSet; $S_1, \ldots, S_k$: XSet; V : NeighborSet; E : EdgeSet

(Propagate)	$\{t \mapsto T\}N(\!\|A\|\!) \to N(\!\|\{t \mapsto T\}A\|\!)$
(Distribute)	$\{t \mapsto T\}(S_1, \ldots, S_k) \to \{t \mapsto T\}S_1, \ldots, \{t_1 \mapsto t_2\}S_k$
(ApplySrc)	$\{t \mapsto t_1, \ldots, t_n\}(i \doteq V) \to$
	if $id(t) \neq i$ then $i \doteq (\{t \mapsto t_1, \ldots, t_n\}V)$
	else $id(t_1) \doteq (\{t \mapsto t_1, \ldots, t_n\}V), \ldots, id(t_n) \doteq (\{t \mapsto t_1, \ldots, t_n\}V)$
(ApplyTar)	$\{t \mapsto t_1, \ldots, t_n\}(i^\frown E) \to$
	if $id(t) \neq i$ then $i^\frown E$ else $id(t_1)^\frown E, \ldots, id(t_n)^\frown E$

Fig. 13.2: Rules for node-morphism application ($\mathcal{A}$)

Proposition 7. $\mathcal{A}$ is strongly terminating and confluent.

After applying a node-morphism on a PGraph-term, the resulting term may be either not well-formed, or not in canonical form, or it may contain black holes. For this purpose we define in the following reductions for cleaning and computing the canonical form.

Rules for Ensuring Well-Formedness

Let $\mathcal{W}$ be the rewrite system defined by the rules presented in Figure 13.3. These rules transform terms with respect to the condition of well-formedness of PGraph-sorted terms specified in Definition 58 as follows :

(w_1) deletes the adjacency equations for black holes ;

(w_2) deletes the black hole neighbors ;

(w_3) deletes extra-edges (edges whose endpoints do not appear among the ports of the connected nodes).

The extra-edges may occur after the application of the node-morphism according to (ApplySrc) and (ApplyTar) on adjacency equations and neighbors respectively, without checking the connectivity between the new nodes. For v a node identifier, $ports(v)$ will return the set of ports of the node. This condition could be avoided by considering PGraph-sorted terms in both sides of the rule (w_3) with the set of nodes containing the term encoding the node identified by v, and the set of adjacency equations containing the sides of the rule (w_3) respectively. In the particular case of the black hole, $ports(\bullet) = \emptyset$.

In Example 22 we illustrate a reduction using the rules in $\mathcal{W}$.

Proposition 8. $\mathcal{W}$ is strongly terminating and confluent.

A direct consequence of the result above is that any term t has a unique normal form with respect to the reduction relation induced by $\mathcal{W}$, which we denote by $t{\downarrow_{\mathcal{W}}}$.

u, v : Id, t_1, t_2 : NeighborSet, t_3, t_4 : EdgeSet, p, r : Port

(w_1) $\bullet \doteq t_1 \rightarrow \epsilon_{AdjacencyEq}$

(w_2) $\bullet \frown t_3 \rightarrow \epsilon_{Neighbor}$

(w_3) $(v \doteq t_1, (u \frown t_3, (p, r), t_4), t_2) \rightarrow (v \doteq t_1, (u \frown t_3, t_4), t_2)$
if $p \notin ports(v)$ or $r \notin ports(u)$

Fig. 13.3: Rules for reducing a term to a well-formed term ($\mathcal{W}$)

Computing the Canonical Form

Let $\mathcal{C}$ be the rewrite system defined by the rules in Figure 13.4. These rules transform terms in the canonical form specified by Definition 59 as follows :

(c_1) merges nodes having the same identifier into one node by merging their port sets ;

(c_2) deletes a neighbor with empty set of edges ;

(c_3) merges the associated sets of edges for identical neighbors ;

(c_4) deletes adjacency equations with empty set of neighbors ;

(c_5) merges adjacency equations having the same identifier in the first component into one adjacency equation by merging the sets in the second component.

v : Id, n : Name, t_1, t_2 : PortSet, t_3, t_4 : NeighborSet, t_5, t_6 : EdgeSet

(c_1) $\langle v : n \parallel t_1 \rangle, \langle v : n \parallel t_2 \rangle \rightarrow \langle v : n \parallel t_1, t_2 \rangle$

(c_2) $v \frown \epsilon_{Edge} \rightarrow \epsilon_{Neighbor}$

(c_3) $(v \frown t_5), (v \frown t_6) \rightarrow v \frown t_5, t_6$

(c_4) $v \doteq \epsilon_{Neighbor} \rightarrow \epsilon_{AdjacencyEq}$

(c_5) $(v \doteq t_3), (v \doteq t_4) \rightarrow v \doteq t_3, t_4$

Fig. 13.4: Rules for computing a canonical form ($\mathcal{C}$)

Using the same proof technique as for Proposition 8, we can prove the following result :

Proposition 9. $\mathcal{C}$ is strongly terminating and confluent.

We denote by $t \downarrow_{\mathcal{C}}$ the unique normal form of a term t with respect to the reduction relation induced by $\mathcal{C}$.

13.5 The pg-Rewriting Relation

We are now ready to define the pg-rewriting relation. Operationally, we apply extended rewrite rules which allow us to deal only with rule application at the root position of terms.

Définition 61 (pg-rewriting relation). A term t of sort PGraph rewrites to a term t' using a pg-rewrite rule $r : t_1 \twoheadrightarrow t_2$ with $\overline{r} : \overline{t_1 \twoheadrightarrow t_2}$ and $\xi = \mathrm{GetMap}(\overline{r})$, which is denoted by $t \xrightarrow{\overline{r}} t'$, if there exists a substitution σ, a solution of the ACU-matching problem $\overline{t_1} \ll t$, such that $t' = \xi^\sigma(\sigma(\overline{t_2}))\downarrow_{\mathcal{W}}\downarrow_{\mathcal{C}}$. We call this relation pg-rewriting and we say that t pg-rewrites to t' by r.

Exemple 22. We present here a result of pg-rewriting the term $t = \mathcal{E}(G)$ from Example 18 encoding the port graph from Figure 10.1, using the rewrite rule $t_1 \twoheadrightarrow t_2$ given in Example 20 and extended in Example 21 which encodes the port graph rewrite rule (a) from Figure 10.3. This result of the pg-rewriting corresponds to the port graph rewriting depicted in Figure 10.6.

1. one solution of the matching problem $\overline{t_1} \ll t$ is given by the substitution :

$$\begin{aligned}\sigma \;=\; \{&i \mapsto 1,\ X \mapsto A,\ x \mapsto a,\ z \mapsto b,\ j \mapsto 2,\ Y \mapsto B,\ y \mapsto e,\ W_1^p \mapsto \epsilon,\\ &W_2^p \mapsto c,\ W_3^p \mapsto \epsilon,\ W_4^n \mapsto (\langle 3 : A \parallel a, b, c\rangle, \langle 4 : C \parallel d\rangle),\ W_5^e \mapsto \epsilon,\\ &W_6^h \mapsto 3^\frown(b, a),\ W_7^a \mapsto ((3 \doteq 4^\frown(a, d)), (4 \doteq 1^\frown(d, c))\}\end{aligned}$$

2. we instantiate $\overline{t_2}$ by σ :

$$\begin{aligned}\sigma(\overline{t_2}) = (&\langle 1.1 : A.1 \parallel a\rangle, \langle 1.2 : A.2 \parallel b, c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a, b, c\rangle, \langle 4 : C \parallel d\rangle)(\!|\\ &1.1 \doteq 2^\frown(a, e),\\ &1.2 \doteq 2^\frown(b, e),\ (3^\frown(b, a)),\\ &3 \doteq 4^\frown(a, d),\\ &4 \doteq 1^\frown(d, c)\ |\!)\end{aligned}$$

   and we note the occurrence of the node identifier 1 in the adjacency equation set which is no longer a valid node identifier in this term ;

3. we instantiate the node-morphism ξ by σ :

$$\begin{aligned}\xi^\sigma \;=\; &\langle 1 : A \parallel a, b, c\rangle \mapsto \langle 1.1 : A.1 \parallel a\rangle, \langle 2.2 : A.2 \parallel b, c\rangle,\\ &\langle 2 : B \parallel e\rangle \mapsto \{\langle 2 : B \parallel e\rangle\end{aligned}$$

4. we apply the instantiated node-morphism on $\sigma(\overline{t_2})$:

$$\begin{aligned}\xi^\sigma(\sigma(\overline{t_2})) \;\to_{\mathcal{A}}^{+}\; &(\langle 1.1 : A.1 \parallel a\rangle, \langle 1.2 : A.2 \parallel b, c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a, b, c\rangle,\\ &\langle 4 : C \parallel d\rangle)(\!|\\ &1.1 \doteq 2^\frown(a, e),\\ &1.2 \doteq 2^\frown(b, e),\ 3^\frown(b, a),\\ &3 \doteq 4^\frown(a, d),\\ &4 \doteq 1.1^\frown(d, c),\ 1.2^\frown(d, c)\ |\!)\end{aligned}$$

   with (ApplyTar) being the last rule used in the reduction on $4 \doteq 1^\frown(d, c)$;

5. we compute the well-formed term using $\mathcal{W}$:

$$\begin{aligned}\xi^{\sigma}(\sigma(\overline{t_2}))\downarrow_{\mathcal{W}} \;=\; & (\langle 1.1 : A.1 \parallel a\rangle, \langle 1.2 : A.2 \parallel b, c\rangle, \langle 2 : B \parallel e\rangle, \langle 3 : A \parallel a, b, c\rangle,\\ & \langle 4 : C \parallel d\rangle)(\!|\\ & 1.1 \doteq 2^\frown(a, e),\\ & 1.2 \doteq 2^\frown(b, e),\ 3^\frown(b, a),\\ & 3 \doteq 4^\frown(a, d),\\ & 4 \doteq 1.2^\frown(d, c)\ |\!)\end{aligned}$$

using the reduction $4 \doteq 1.1^\frown(d, c), 1.2^\frown(d, c) \xrightarrow{(ExtraEdges)} 4 \doteq 1.2^\frown(d, c)$ since the port c does not occur in the port set of the node identified by 1.1, but it occurs in the port set of the node identified by 1.2 ;

6. in the end we compute the canonical form using $\mathcal{C}$; however no rule is applied :

$$\xi^{\sigma}(\sigma(\overline{t_2}))\downarrow_{\mathcal{W}}\downarrow_{\mathcal{C}} \;=\; \xi^{\sigma}(\sigma(\overline{t_2}))\downarrow_{\mathcal{W}}$$

The above solution σ of the matching problem leads to the result term $\xi^{\sigma}(\sigma(\overline{t_2}))\downarrow_{\mathcal{W}}\downarrow_{\mathcal{C}}$ which in fact is the term encoding of the port graph G' in Figure 10.6.

Proposition 10. If t pg-rewrites to t' and t is a well-formed term in canonical form then t' is well-formed and in canonical form.

Remarque 7. We saw that a port graph rewrite rule $O_1 \Rightarrow O_2$ has a port graph representation ; therefore the abstractions of the forms $(O_1 \Rightarrow O_2) \Rightarrow (O_1' \Rightarrow O_2')$ and $O \Rightarrow G$ have a port graph representation. Since here we defined an encoding for port graphs and the port graph rewriting relation handles, then it encompasses all types of port graph molecules considered in the ρ_{pg}-calcul (Chapter 11).

13.6 Operational Correspondence

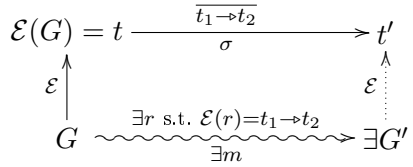
In this section we show that the encoding of the weak port graph rewrite relation from Chapter 10 as a term rewriting relation is sound and complete.

Théorème 5 (Operational correspondence).

1. Let G, G' be two port graphs, r a weak port graph rewrite rule and m a port graph morphism such that $G \overset{r}{\rightsquigarrow} G'$ using m. Then there exists a substitution σ and a term t' such that $\mathcal{E}(G) \xrightarrow{\overline{\mathcal{E}(r)}} t'$ using the substitution σ and $t' = \mathcal{E}(G')$.

$$\begin{array}{ccc} G & \overset{r}{\underset{m}{\rightsquigarrow}} & G' \\ \downarrow{\scriptstyle \mathcal{E}} & & \vdots{\scriptstyle \mathcal{E}} \\ \mathcal{E}(G) = t & \underset{\exists\sigma}{\xrightarrow{\overline{\mathcal{E}(r)}}} & \exists t' \end{array}$$

2. Let G be port graph, $t = \mathcal{E}(G)$, $t_1 \twoheadrightarrow t_2$ a pg-rewrite rule, and t' such that $t \xrightarrow{\overline{t_1 \twoheadrightarrow t_2}} t'$ with σ the solution of the matching $\overline{t_1} \ll t$ used in the rewriting. Then there exist
   a) a weak port graph rewrite rule r satisfying $\mathcal{E}(r) = t_1 \twoheadrightarrow t_2$,
   b) a port graph morphism that can be constructed using σ and the structures of t and G, and
   c) a port graph G' such that $G \overset{r}{\rightsquigarrow} G'$ using the matching morphism m and $\mathcal{E}(G') = t'$.



There is also a correspondence between all possible results of rewriting a port graph G using a rule $G_1 \rightsquigarrow G_2$ and a morphism m, and possible results of rewriting $t = \mathcal{E}(G)$ using $t_1 \twoheadrightarrow t_2 = \mathcal{E}(G_1 \rightsquigarrow G_2)$ since all solutions of the matching $\overline{t_1} \ll t$ have as common basis the encoding of m, but different mappings for the extension variables. Hence, while the application for port graphs of the node-morphism on unmatched partial nodes produces k results, the application of node-morphism for a term produces a term and the k solutions arise from different solutions of the ACU-matching problem with the extension variables.

13.7 Relation to the ρ-Calculus

Based on the encoding of the port graph rewriting relation as defined in Section 13.5, we instantiate the ρ-calcul for terms encoding port graphs as follows :

- take for $\mathcal{K}$ the operation symbols in $\mathcal{F}$ with the partial ordered set of sorts $(\mathcal{S}, <)$ and for $\mathcal{X}$ an $(\mathcal{S}, <)$-sorted family of variables ;
- consider as patterns well-formed terms in canonical form in $\mathcal{T}_{\Sigma,\mathrm{PGraph}}(\mathcal{X})$;
- consider only port graph rewrite rules corresponding to the possible types of abstraction in the ρ_{pg}-calcul (Chapter 11).

We benefit in addition of the structure operator which allows grouping rules or results of applications.

As for the semantics, while (δ) dealing with the distributivity of the application over structures is taken as such from the ρ-calcul, we need a new rule for the application of a rewrite rule $t_1 \twoheadrightarrow t_2$ on a well-formed term t_3 in canonical form as follows :

$$(\rho_{tpg}) \quad (\overline{t_1 \twoheadrightarrow t_2})\ t_3 \to_\rho S(\varsigma_1(\overline{t_2})) \wr \ldots \wr S(\varsigma_n(\overline{t_2})),$$
$$\text{if } \mathcal{S}ol(\overline{t_1} \ll t_3) = \{\sigma_1, \ldots, \sigma_n\}, \xi = \mathrm{GetMap}(\overline{t_1 \twoheadrightarrow t_2}), \varsigma_i = \sigma_i \circ \xi^{\sigma_i}$$

where $\overline{t_1 \twoheadrightarrow t_2}$ is the extended rule associated to $t_1 \twoheadrightarrow t_2$, the matching problem is solved using an ACU-matching algorithm, and S is a strategy which reduces a term to its normal form with respect to rewriting systems $\mathcal{W}$ and $\mathcal{C}$, i.e., to a canonical well-formed term.

We obtain this way a term rewriting calculus for port graphs, which is an instance of the ρ-calcul, and we call it the ρ_{tpg}-calculus.

Comparison with the Higher-Order Calculus for Graph Transformation

M. Fernandez, I. Mackie, and J. S. Pinto introduced in [FMP07] a higher-order calculus for graph Transformation Using a syntax based on the Combinatory Reduction Systems (CRSs) [Klo80] and on the equational notation for term-graph rewriting. In this calculus a (hyper)graph is a term of the form $\eta[x_1, x_2, \ldots, x_n].\{s \mid t\}$ where s is a set of variables representing the interface of the graph (the nodes where the graph may be glued with other graphs), t the list of edges (with each edge given by its label and endpoints), and the binder η hiding all nodes $x_1, x_2, \ldots, x_n$ that are not in the interface. Then for $L \Rightarrow R$ a graph transformation rule, L and R are encoded by two terms l and r as above such that their interfaces define a mapping between the nodes of L and R. The graph transformation rule is transformed basically into a rule of the form $Z(l) \Rightarrow Z(r)$ with Z a metavariable corresponding to the context.

In this chapter, we have encoded port graphs as algebraic terms over a first-order signature. However, by adding context variables, we have obtained an extension of a port graph rewrite rule in the similar way as in the higher-order calculus for graph transformation.

The Relation between the ρ_{pg}-Calculus and the ρ_{tpg}-Calculus

In the following we review the relations between the calculi we developed until now :
- the $\rho_{\langle\Sigma\rangle}$-calcul generalizing the γ-calculus based on the pattern matching idea from the ρ-calcul on an arbitrary structure described by Σ ;
- the ρ_{pg}-calcul instantiating the $\rho_{\langle\Sigma\rangle}$-calcul using the port graph structure and the port graph rewriting relation ;
- the ρ_{tpg}-calculus as a rewriting calculus on algebraic terms encoding port graphs.

Therefore the ρ_{pg}-calcul is based on the ρ_{tpg}-calculus in a similar way the $\rho_{\langle\Sigma\rangle}$-calcul is based on the ρ-calcul.

13.8 Conclusions

In this chapter we provided a sound and complete axiomatization of port graphs and port graph rewriting using terms from a suitable first-order algebra and a term rewriting relation. A term encoding a port graph consists of two subterms, one for the set of node labels, and the other for the set of adjacency equations for each node. We have defined a rewriting relation such that the rewrite rules are applied at the top position of terms. As a consequence, we have instantiated the ρ-calcul with the algebraic signature for port graphs, the class of patterns and abstractions corresponding to the abstractions used in the ρ_{pg}-calcul, and the pg-rewriting relation, and we obtained a rewriting calculus for terms encoding port graphs.

In the implementation of port graphs using TOM [BBK+07b] for modeling the biochemical example presented in Chapter 12, we preferred a representation similar to the one in [Mes96] : a set of terms encoding the nodes as identifier, name, and list of immediate neighbors. The main lines of this implementation are presented in Appendix A.

We have used an encoding more efficient for programming by representing a port graph by its node set and each edge by a pointer to the target port associated to the source port. The gain in efficiency is provided by the pointers introduced in [BB07] for a term graph rewriting implementation which handles cyclic term graphs as well. In addition, the use of pointers and TOM's maximal sharing is very important in order to cope with computer representation of large terms. A further development of this implementation would be to generalize to rewriting terms encoding port graphs. We could benefit from the traversal strategies in TOM to improve the implementation, in particular the research of particular patterns.

The encoding we presented here is focused on port graphs and port graph rewriting, nevertheless it can be used for graphs and graph rewriting. The syntax proposed by higher-order calculus for graph transformation based on CRSs is more general and expressive since the aim of the calculus is to encompass graph and term-graph rewriting systems [Plu99], Interaction Nets [Laf90], Interaction Systems [AL94], non-deterministic nets [Ale99], and process calculi. In consequence, based on the similarity between the ways of encoding graphs as as we did and using the CRS-based syntax in [FMP07], we could imagine a calculus based on port graph rewriting for simulating the other graphical formalisms mentioned above. In particular, the Interaction Nets represent a formalism that could be easily encoded by port graphs since an agent (a node) has distinguished ports.

14 Runtime Verification in the ρ_{pg}-Calculus

In Chapter 10 we have defined the structure of port graph, port graph rewrite rules and a rewriting relation for port graphs. We have designed this formalism for modeling complex systems with dynamical topology whose components interact in a concurrent and distributed manner. Nodes with ports represent components (objects), while edges communication channels. Then port graph rewrite rules are used for modeling the interactions between some entities or nodes capable of creating connections among them at specific points (ports), breaking connections, merging, splitting, or deleting nodes, or any combination of these operations. A dynamic system whose initial state is represented by a port graph structure evolves according to a set of port graph rewrite rules leading to change in the its structure over time. In Chapter 11 we have defined a higher-order formalism, the ρ_{pg}-calcul, where we reason about the description of the state and the description of the system's behavior at the same level. This allowed us to add more expressive power in modeling a system via port graph rewrite rules that may introduce other port graph rewrite rules and via strategies for controlling the application order of a set of port graph rewrite rules. We then have shown the capabilities of the ρ_{pg}-calcul to model autonomous systems and biochemical networks. After defining a formal specification framework for such systems, the next step that comes naturally is to provide an automated method for validating the behavior of the system with respect to some initial design requirements or properties.

In this chapter our aim is to endow the calculus with a method of verifying a given set of requirements for a modeled system. A particularly relevant formalism for expressing properties of dynamic systems is temporal logic. Therefore we express the requirements a system behavior has to meet as temporal formulas in a suitable logic and we put them at the same level as the system description. Then the behavior of the system is dynamically verified to satisfy the given requirements based on the semantics of the chosen temporal logic. We obtain a runtime verification technique which allows the running system to detect its own failures. Usually, this verification technique increases the confidence in the correctness of the system behavior with respect to its formal specification. In particular, for autonomous systems, runtime verification is useful for recovering from problematic situation, hence for the self-healing property.

In order to reason about the evolutions of the structure of system states in time, we consider a standard temporal logic that is well-suited for reasoning on port graph reduction, the Computational Tree Logic (CTL) [CGP00]. The atomic propositions are structural formulas which we encode by means of some adequate rewrite strategies and we verify that the modeled system satisfies them using the evaluation mechanism of the rewrite strategies. Then the CTL formulas constructed on these atomic propositions allow us to formulate requirements about the dynamic properties of the modeled system.

For instance we can express an invariant property p as the CTL formula AG(p) (i.e., for all possible executions, at any moment, p is true), whereas a fatality property as AF(p) for p an atomic proposition (i.e., for all possible executions, there is a moment when p is true).

In the following we motivate the choice of CTL as temporal logic for performing runtime verification in the ρ_{pg}-calcul. On one hand, the idea of using of the strategy formalism for ensuring the satisfiability of CTL formulas came from [BMR07]. In this paper TOM strategies are used for expressing temporal properties of the method code in a program with predicates as basic strategies (set of rules). Then a terminating non-failing strategy ensures that the encoded CTL formula is true. On the other hand, CTL has already been used for querying and validation of molecular interactions [CRCFS04, PC03, BRJ$^+$05, MRM$^+$08]. For instance, in [CRCFS04] the expresiveness of CTL for formalizing a wide variety of biological queries on the possible behaviors of a biochemical system. Some typical biological queries are identified and they concern reachability of particular states from an initial state, properties about pathways (reachability of a state under a certain constraint), stability properties. Later, in [MRM$^+$08] the authors identified temporal logic patterns for expressing biological queries concerning occurrence/exclusion, consequence, sequence, and invariance of cellular events. These patterns cover all biological queries identified in [CRCFS04].

The structure of the chapter is as follows. We start with an overview of the concepts from CTL adjusted for a system whose initial state has the structure of a port graph and whose behavior is described by a port graph rewrite system (Section 14.1). The particularity comes from the treatment of atomic propositions as structural formulas over port graphs. Then we model the evolution of such a system by a transition system where the states are port graphs and the transition relation is defined using the (strategic) port graph rewrite relation. In Section 14.2 we extend the syntax of the ρ_{pg}-calcul with a set of CTL formulas that guard the worlds and multiverses. Then we extend the semantics of the ρ_{pg}-calcul for testing the satisfiability of formulas guarding the worlds and the multiverses with respect to the initial world of the modeled system. In particular, we define the satisfaction of a structural formula on a state of the system as the successful application of an appropriate strategy on the state. At the end of the section we illustrate the approach by some formulas encoded in the newly defined calculus, the ρ^v_{pg}-calcul, for a biological system modeled using the ρ_{pg}-calcul. We end the chapter with some conclusions and perspectives.

14.1 CTL for Port Graphs and Port Graph Rewriting

In this chapter we review the syntax and semantics of CTL from a port graph rewriting perspective. The difference from the classical definition comes from the particular set of atomic propositions we consider, as well as from the associated definition of the satisfaction relation.

In a first stage we define a set of formulas for reasoning locally about subgraphs. An elementary formula is represented by a port graph expression; then we combine port

graph expressions using Boolean connectors to obtain structural formulas. The satisfaction problem of a port graph expression is equivalent to a matching problem ; in consequence, the satisfaction problem for a structural formula is decomposed to several satisfaction problems for port graph expressions. In a second stage, we review the CTL formulas [CGP00] using path quantifiers and temporal operators together with the corresponding satisfaction relation for reasoning on port graph rewriting.

14.1.1 Port Graph Expressions

We consider the same domains for the node identifiers, node names, and port names given by a p-signature ∇ as in Chapter 10. We briefly recall their construction in the following. Node identifiers are constructed based on integers, variables, and sequential composition (hence Id^*, where $\mathrm{Id} = \mathrm{Int} \cup \mathrm{VarId}$). Let $\nabla = \langle \nabla_{\mathscr{N}}, \nabla_{\mathscr{P}} \rangle$ be a p-signature, and $\mathcal{X} = (\mathcal{X}_{\mathscr{N}}, \mathcal{X}_{\mathscr{P}})$ a pair of sets of variables names for nodes and ports. Then $\nabla^{\mathcal{X}}$ denotes the p-signature $\langle \nabla_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{N}}, \nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}} \rangle$. The node names can be constant strings (capital letters) from $\nabla_{\mathscr{N}}$, variables from $\mathcal{X}_{\mathscr{N}}$, and sequential compositions of a node name and an integer, or sequential composition of two node names. The port names can be constant strings from $\nabla_{\mathscr{P}}$ or variables (lower case letters) from $\mathcal{X}_{\mathscr{P}}$.

A port graph expression can be associated to each construct of a port graph defined in Chapter 11 : the empty graph, a node, composition of graphs by juxtaposition and connection between two ports, and self-connection on different ports and on a same port. Figure 14.1 summarizes these constructs.

Let $\sigma : \mathcal{X} \to \mathrm{Int}^* \cup \nabla$ be a variable assignement from variables to constants for node identifiers, node names, and port names respectively. We usually denote by σ^* the extension of σ over port graph expressions. We define the satisfaction relation of port graph expressions using the structural congruence relation $\equiv$ defined on port graph molecules given by Definition 54.

Définition 62 (Satisfaction of port graph expressions). A port graph G satisfies a port graph expression γ if and only if there exists a variable assignement σ such that $G \equiv \sigma^*(\gamma)$, which we denote by $G \models^{\sigma} \gamma$.

Remark that a port graph expression γ is satisfied by a port graph G if γ corresponds to a port graph which matches G, i.e., $\mathcal{S}ol(\gamma \ll G) \neq \emptyset$. It follows that a port graph expression γ is not satisfied by G, denoted by $G \not\models^{\sigma} \gamma$, if $\mathcal{S}ol(\gamma \ll G) = \emptyset$.

14.1.2 Structural Formulas

Based on the port graph expressions defined above and on the Boolean connectors, we introduce in the following the structural formulas whose satisfaction we intend to verify on the states of a system. These formulas represent the atomic propositions for a model.

Définition 63 (Structural formulas). Given $\nabla^{\mathcal{X}}$ a p-signature, the set of structural formulas, denoted by $\mathcal{FS}(\nabla^{\mathcal{X}})$, is constructed inductively as follows :

– $\top$ and $\bot$ are structural formulas ;

Node identifier expressions	ξ	::=	n, $n \in Int$	constant node identifier
		|	i, $i \in$ VarId	variable node identifier
		|	$\xi_1.\xi_2$	composed node identifier
Node name expressions	α	::=	A, $A \in \nabla_{\mathscr{N}}$	constant node name
		|	X, $X \in \mathcal{X}_{\mathscr{N}}$	variable node name
		|	$\alpha_1.\alpha_2$	composed node name
		|	$\alpha.n$, $n \in Int$	composed node name
Port name expressions	β	::=	a, $a \in \nabla_{\mathscr{P}}$	constant port name
		|	x, $x \in \mathcal{X}_{\mathscr{P}}$	variable port name
Port name list expressions	$\overline{\beta}$	::=	ε	empty list
		|	β	port name expression
		|	$\overline{\beta}, \overline{\beta}$	list concatenation
Port graph expressions	γ	::=	ε	empty graph
		|	$\xi : \alpha : \overline{\beta}$	node
		|	$\gamma_1\ \gamma_2$	juxtaposition
		|	γ_1  γ_2	connection
		|	γ 	loop
		|	γ 	self connection

Fig. 14.1: Port graph expressions

- any port graph expression γ is a structural formula ;
- if φ, φ_1, and φ_2 are structural formulas, then $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, and $\diamondsuit\varphi$ are structural formulas.

In Figure 14.2 we give the grammar generating the set of structural formulas $\mathcal{FS}(\nabla^{\mathcal{X}})$.

The Boolean operators $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\rightarrow$ are the usual operators from propositional logic for "true", "false", "not", "and", "or", and "implies" respectively. The somewhere operator $\diamondsuit$ requires that the property holds on a fragment or port subgraph of the state. As already known, we can consider only the Boolean operators $\bot$, $\neg$, and $\vee$; then combinations of these operators define the other Boolean operators, $\top$, $\wedge$, and $\rightarrow$.

Définition 64 (Structural satisfaction). The satisfiability of a structural formula $\varphi \in \mathcal{FS}(\nabla^{\mathcal{X}})$ by a port graph G, denoted by $G \models \varphi$, is defined inductively as follows :

$$
\begin{array}{lcl}
G \models \top & & \\
G \not\models \bot & & \\
G \models \gamma & \Leftrightarrow & \exists\sigma \text{ such that } G \models^{\sigma} \gamma
\end{array}
$$

Structural formulas	φ	::=	$\top$	true
		|	$\bot$	false
		|	γ	port graph expression
		|	$\neg\varphi$	negation
		|	$\varphi_1 \wedge \varphi_2$	conjunction
		|	$\varphi_1 \vee \varphi_2$	disjunction
		|	$\varphi_1 \rightarrow \varphi_2$	implication
		|	$\diamondsuit\varphi$	somewhere modality

Fig. 14.2: Structural formulas for port graphs

$G \models \neg\varphi$	$\Leftrightarrow$	$G \not\models \varphi$
$G \models \varphi_1 \wedge \varphi_2$	$\Leftrightarrow$	$G \models \varphi_1$ and $G \models \varphi_2$
$G \models \varphi_1 \vee \varphi_2$	$\Leftrightarrow$	$G \models \varphi_1$ or $G \models \varphi_2$
$G \models \varphi_1 \rightarrow \varphi_2$	$\Leftrightarrow$	$G \not\models \varphi_1$ or $G \models \varphi_2$
$G \models \diamondsuit\varphi$	$\Leftrightarrow$	$\exists G' \sqsubseteq G$ such that $G' \models \varphi$

Whereas a structural formula γ is satisfied by a port graph G if γ corresponds to a port graph matching G, G structurally satisfies a formula $\diamondsuit\gamma$ is γ corresponds to a port graph submatching G.

Proposition 11. $G \models \diamondsuit\gamma$ if and only if $\mathcal{S}ol(\gamma \lessdot G) \neq \emptyset$.

The following proposition states that the structural satisfaction is up to the structural congruence relation on port graphs given by Definition 54. The proof is immediate using induction over the structure of φ.

Proposition 12. If $G \models \varphi$ and $G \equiv G'$ then $G' \models \varphi$.

14.1.3 State and Path Formulas

The models for CTL are labeled transitions systems or Kripke structures where each state is labeled by the set of the satisfied atomic proposition.

Définition 65 (Kripke structure [CGP00]). Given a set of atomic propositions AP, a Kripke structure over AP is a construct $\mathcal{K} = (S, R_t, L)$ where :

1. S is a finite set of states,
2. $R_t \subseteq S \times S$ is a total relation called the transition relation, i.e., for all $s \in S$, $\exists s' \in S$ such that sR_ts',
3. $L : S \rightarrow 2^{AP}$ is the labeling function associating to each state $s \in S$ the set of those atomic proposition in AP that hold in the state s.

Sometimes the Kripke structure must specify a set of initial sets of states. A path in the structure $\mathcal{K}$ from a state s is an infinite sequence of states $\pi = s_0 s_1 s_2 \ldots$ such that $s = s_0$ and $R_t(s_i, s_{i+1})$ holds for all $i \geq 0$. The notation π^i refers to the suffix of π starting from the element of the i-th position.

We define the semantics of CTL for port graphs with respect to an adequate Kripke structure. In this case, the atomic propositions for a state are structural formulas on port graphs. However, we do not associate to a state a label consisting of the set of the satisfied atomic propositions, since we can test the satisfaction of such a proposition by using a submatching algorithm.

We can easily associate a Kripke structure to a (strategic) rewriting system by following the same lines as for rewrite theories in Rewriting Logic [EMS04].

Définition 66 (Model for CTL for port graph rewriting). A model for CTL for port graphs over a p-signature $\nabla^{\mathcal{X}}$ and the port graph rewriting system $\mathcal{R}$ is a Kripke structure system $\mathcal{M} = (Ob(\text{PGraph}), \rightarrow_{\mathcal{R}}^{\bullet}, L_{\text{PGraph}})$ where :

- $Ob(\text{PGraph})$ is the set of states given by the set of objects in the category of port graphs PGraph,
- $\rightarrow_{\mathcal{R}}$ is the one-step rewriting relation defined by $\mathcal{R}$ on $Ob(\text{PGraph})$,
- $\rightarrow_{\mathcal{R}}^{\bullet}$ is the total relation associated to $\rightarrow_{\mathcal{R}}$ which adds a self-loop for each irreducible or terminal state G (i.e., G cannot be further rewritten using rules in $\mathcal{R}$) ;
- $L_{\text{PGraph}} : Ob(\text{PGraph}) \rightarrow \mathcal{P}(\mathcal{FS}(\nabla^{\mathcal{X}}))$ that maps each port graph G to a set of structural formulas that hold in G, that is, $L_{\text{PGraph}}(G) = \{\varphi \in \mathcal{FS}(\nabla^{\mathcal{X}}) \mid G \models \varphi\}$.

If in the definition of an abstract reduction system we consider that the objects are equivalence classes of port graphs and the steps are one-step port graph rewritings defining the transition relation, we obtain a port graph reduction system (PGRS). We remark that every model for CTL for port graphs as above has an underlying port graph reduction system generated by the port graph rewriting system $\mathcal{R}$. Therefore we can borrow all definitions from abstract reduction systems on paths, derivations, concatenation of derivations, strategy, strategy application, and strategy derivation.

Let G be the initial port graph. The computational tree associated to a model $\mathcal{M}$ over $\nabla^{\mathcal{X}}$ and $\mathcal{R}$ with G the initial state, denoted by $\mathcal{M}(\mathcal{R}, G)$, is obtained by unwinding the structure into a possibly infinite tree with G as the root. Then whenever we consider a path, we refer to a path in this tree starting from the root if not otherwise specified. The state formulas and the path formulas refer to structural properties of states and to properties of the branching structure of the underlying graph respectively. The path quantifiers A and E help describing the branching structure in the computation tree, while the path operators (X, F, G, U, R) help describing properties of a path in such a tree. A property of a path is formulated using five basic operators :

- X (neXt time) requires that a property has to hold in the next state of the path ;
- G (Globally or always) requires that a property has to hold at every state on the path ;
- F (Finally, eventually, or in the Future) requires that a property eventually has to hold somewhere on the path ;

- U (Until) takes two properties as arguments, and it requires that the first property has to hold until some state where the second property holds ;
- R (Release) is also binary and it requires that the second property holds along the path until and including the state where the first property holds.

In the following we give the usual syntax of CTL [CGP00] where we consider the structural formulas defined previously as atomic propositions. The syntax of the CTL formulas for port graphs is summarized in Figure 14.3.

Définition 67 (State and path formulas). The sets of state formulas and path formulas are defined as follows :

- any structural formula is a state formula ;
- if ψ is a path formula, then $\mathrm{A}\psi$ and $\mathrm{E}\psi$ are state formulas ;
- if ϕ, ϕ_1, ϕ_2 are state formulas, then $\neg\phi$, $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\phi_1 \rightarrow \phi_2$ are state formulas ;
- if ϕ, ϕ_1, ϕ_2 are state formulas, then $\mathrm{X}\phi$, $\mathrm{G}\phi$, $\mathrm{F}\phi$, $\phi_1 \mathrm{U} \phi_2$, $\phi_1 \mathrm{R} \phi_2$ are path formulas.

In CTL there is a minimal set of operators such that all formulas can be expressed using them. One such minimal set is $\{\bot, \neg, \vee, \mathrm{EX}, \mathrm{EG}, \mathrm{EU}\}$.

State formulas	ϕ	::=	φ	structural formula
		|	$\neg\phi$	negation
		|	$\phi_1 \wedge \phi_2$	conjunction
		|	$\phi_1 \vee \phi_2$	disjunction
		|	$\phi_1 \rightarrow \phi_2$	implication
		|	$\mathrm{A}\psi$	for all paths
		|	$\mathrm{E}\psi$	exists a path
Path formulas	ψ	::=		
		|	$\mathrm{X}\phi$	on the next state in the path
		|	$\mathrm{G}\phi$	globally on the path
		|	$\mathrm{F}\phi$	somewhere on the path
		|	$\phi_1 \mathrm{U} \phi_2$	until
		|	$\phi_1 \mathrm{R} \phi_2$	release

Fig. 14.3: Formulas in CTL for port graphs

The semantics of CTL for port graphs with respect to a model $\mathcal{M}$ is similar to the one of CTL. If ϕ is a state formula, the notation $\mathcal{M}, G \models \phi$ means that ϕ holds at the state G in the model $\mathcal{M}$. Similarly, if ψ is a path formula, then the notation $\mathcal{M}, \pi \models \psi$ means that ψ holds along the path π in the model $\mathcal{M}$, while the notation $\mathcal{M}, \pi^k \models \phi$ means that ϕ holds in the state on the k-th position of the path π in $\mathcal{M}$.

Définition 68 (Temporal satisfaction). The satisfiability of formulas in CTL for port graphs in a model $\mathcal{M}$ is defined inductively as follows, for G a state in $\mathcal{M}$ and π a path :

$$
\begin{array}{lcl}
\mathcal{M}, G \models \neg\phi & \Leftrightarrow & \mathcal{M}, G \not\models \phi \\
\mathcal{M}, G \models \phi_1 \wedge \phi_2 & \Leftrightarrow & \mathcal{M}, G \models \phi_1 \text{ and } \mathcal{M}, G \models \phi_2 \\
\mathcal{M}, G \models \phi_1 \vee \phi_2 & \Leftrightarrow & \mathcal{M}, G \models \phi_1 \text{ or } \mathcal{M}, G \models \phi_2 \\
\mathcal{M}, G \models \phi_1 \rightarrow \phi_2 & \Leftrightarrow & \mathcal{M}, G \not\models \phi_1 \text{ or } \mathcal{M}, G \models \phi_2 \\
\mathcal{M}, G \models \mathrm{A}\psi & \Leftrightarrow & \text{for every path } \pi \text{ starting from G, } \mathcal{M}, \pi \models \psi \\
\mathcal{M}, G \models \mathrm{E}\psi & \Leftrightarrow & \text{there is a path } \pi \text{ starting from G such that } \mathcal{M}, \pi \models \psi \\
\mathcal{M}, \pi \models \mathrm{X}\phi & \Leftrightarrow & \mathcal{M}, \pi^1 \models \phi \\
\mathcal{M}, \pi \models \mathrm{G}\phi & \Leftrightarrow & \text{for all } k \geq 0, \ \mathcal{M}, \pi^k \models \phi \\
\mathcal{M}, \pi \models \mathrm{F}\phi & \Leftrightarrow & \text{there exists } k \geq 0 \text{ such that } \mathcal{M}, \pi^k \models \phi \\
\mathcal{M}, \pi \models \phi_1 \mathrm{U} \phi_2 & \Leftrightarrow & \text{there exists a } k \geq 0 \text{ such that } \mathcal{M}, \pi^k \models \phi_2 \text{ and} \\
 & & \text{for all } 0 \leq i < k, \ \mathcal{M}, \pi^i \models \phi_1 \\
\mathcal{M}, \pi \models \phi_1 \mathrm{R} \phi_2 & \Leftrightarrow & \text{for all } j \geq 0, \text{ if for every } i < j \ \mathcal{M}, \pi^i \not\models \phi_1 \text{ then} \\
 & & \mathcal{M}, \pi^j \models \phi_2
\end{array}
$$

14.2 Embedding Verification in the ρ_{pg}-Calculus : the ρ^v_{pg}-Calculus

We extend the ρ_{pg}-calcul by including a subset of CTL formulas in the syntax and by defining a new big-step reduction semantics which extends the big-step reduction relation of the ρ_{pg}-calcul with verification of formulas ; we call this calculus the ρ^v_{pg}-calcul. The structural formulas are represented as strategies, hence their satisfiability is tested based on the evaluation mechanism available for strategies. For temporal formulas new objects need to be added to the syntax.

14.2.1 Syntax

We add to the syntax of the ρ_{pg}-calcul the formulas to be verified along the reduction of structures of worlds. We define the state and path formulas as objects of the calculus.

Structural Formulas

The structural formulas we consider for ρ^v_{pg}-calcul correspond to structural formulas in CTL for port graphs in disjunctive or conjunctive normal form constructed inductively from port graph expressions and usual logical connectives from predicate logic :

$$\varphi ::= \top \mid \ \bot \ \mid \Diamond\gamma \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2$$

Any port graph expression γ must be quantified by a somewhere modality in order to be a structural formula since a formula $\Diamond\gamma$ supposes solving a submatching problem which is supported currently in the ρ_{pg}-calcul.

The port graph expressions used in the structural formulas presented for CTL for port graphs in Section 14.1 correspond to object port graphs. But they can be easily extended to port graph molecules since the arrow and application operators, $\Rightarrow$ and @, are represented as nodes.

Verification of Structural Formulas using Strategies

We represent a structural formula by a strategy. Assuming that any port graph expression γ can be seen as a port graph molecule in the calculus, we define the mapping τ from structural formulas to strategies (or extended abstractions) as follows :

$$\begin{array}{lcl}
\tau(\top) & = & \mathrm{id} \\
\tau(\bot) & = & \mathrm{fail} \\
\tau(\diamondsuit\gamma) & = & \gamma \Rightarrow \gamma \\
\tau(\neg\varphi) & = & \mathrm{not}(\tau(\varphi)) \\
\tau(\varphi_1 \wedge \varphi_2) & = & \mathrm{seq}(\tau(\varphi_1), \tau(\varphi_2)) \\
\tau(\varphi_1 \vee \varphi_2) & = & \mathrm{first}(\tau(\varphi_1), \tau(\varphi_2)) \\
\tau(\varphi_1 \rightarrow \varphi_2) & = & X \Rightarrow \mathrm{seq}(\tau(\varphi_1), \mathrm{first}(\mathrm{stk} \Rightarrow X, \tau(\varphi_2)))@X
\end{array}$$

Proposition 13. Let φ be a structural formula in the ρ^v_{pg}-calcul and G a port graph molecule. Then $\tau(\varphi)@G$ reduces either to $\{[G]\}$ or to $\{[\mathrm{stk}]\}$.

Intuitively, if the application of the strategy encoding φ on a port graph G fails, then the formula is not satisfied by G.

Proposition 14 (Correctness of the encoding). Let G be a port graph molecule and φ a structural formula in the ρ^v_{pg}-calcul. If $G \models \varphi$ then $\tau(\varphi)@G \longrightarrow^* \{[G]\}$, otherwise, if $G \not\models \varphi$ then $\tau(\varphi)@G \longrightarrow^* \{[\mathrm{stk}]\}$.

Proposition 15 (Completeness of the encoding). Let G be a port graph molecule and φ a structural formula in the ρ_{pg}-calcul. If $\tau(\varphi)@G \longrightarrow^* \{[G]\}$ then $G \models \varphi$, otherwise, if $\tau(\varphi)@G \longrightarrow^* \{[\mathrm{stk}]\}$ then $G \not\models \varphi$.

Remarque 8. Since the conjunction is commutative then, intuitively, we have the equality $\tau(\varphi_1 \wedge \varphi_2) = \tau(\varphi_2 \wedge \varphi_1)$, hence :

$$\mathrm{seq}(\tau(\varphi_1), \tau(\varphi_2)) \quad = \quad \mathrm{seq}(\tau(\varphi_2), \tau(\varphi_1))$$

Such strategy equality is possible due to the particular encoding of a port graph expression. Obviously, this equality does not imply that the strategy operator seq is commutative. The same discussion can be made for the disjunction encoded using the strategy operator first.

Exemple 23 (Verifying the absence of a virus in a biological system). A usual problem in modeling a biological system possibly exposed to viruses is to detect their presence as soon they enter the system. In other words, we do not want that a particular virus

described by a pattern γ occurs in a state of the system, hence the state should not satisfy the formula $\neg\gamma$.

This problem has also an alternative solution which does not use the logical negation. In the conditions of heating with recovery, we add in the world corresponding to the state of the system an abstraction $\gamma \Rightarrow \gamma$. If at any moment the virus pattern γ occurs in the system, the abstraction will eventually successfully apply, hence it will no longer occur in the sequel worlds since it is consumed by the application. Then the absence of the virus pattern γ in a state is assured by the presence of the abstraction $\gamma \Rightarrow \gamma$, in consequence we do not test the satisfiability of the formula $\neg\gamma$, but the satisfiability of the "positive" formula $\diamondsuit(\gamma \Rightarrow \gamma)$. Remark that this procedure is possible since we allow abstractions to be applied on other abstractions.

CTL Formulas in the ρ^v_{pg}-calcul

We include in the ρ^v_{pg}-calcul the temporal formulas built upon the structural formulas defined previously and the CTL operators : AX, EX, AF, EF, AG, EG, AU, EU, AR, ER. In addition, since φ is a structural formula verifiable on the port graph molecule contained by a world, and we have also the concept of a structure of worlds or multiverse, we use φ as structural formula on worlds, and for a multiverse we use the path quantifiers A and E for quantifying over a structural formula on worlds. Hence the set of formulas we define for the ρ^v_{pg}-calcul is the following :

$$
\begin{array}{lcl}
\mathcal{F}_1 & ::= & \varphi \\
\mathcal{F}_2 & ::= & \mathrm{A}\varphi \mid \mathrm{E}\varphi \\
\mathcal{F}_3 & ::= & \mathrm{AX}\varphi \mid \mathrm{EX}\varphi \mid \mathrm{AF}\varphi \mid \mathrm{EF}\varphi \mid \mathrm{AG}\varphi \mid \mathrm{EG}\varphi \mid \mathrm{A}(\varphi \mathrm{U} \varphi) \mid \mathrm{E}(\varphi \mathrm{U} \varphi) \mid \mathrm{A}(\varphi \mathrm{R} \varphi) \mid \mathrm{E}(\varphi \mathrm{R} \varphi) \\
\mathcal{F}_v & ::= & \mathcal{F}_1 \mid \mathcal{F}_3 \mid \neg\mathcal{F}_v \mid \mathcal{F}_v \vee \mathcal{F}_v \mid \mathcal{F}_v \wedge \mathcal{F}_v \\
\mathcal{F}_w & ::= & \mathcal{F}_2 \mid \mathcal{F}_3 \mid \neg\mathcal{F}_w \mid \mathcal{F}_w \vee \mathcal{F}_w \mid \mathcal{F}_w \wedge \mathcal{F}_w
\end{array}
$$

Guarded Worlds and Multiverses

We guard the worlds and the multiverses with formulas from $\mathcal{F}$:

$$
\begin{array}{lcl}
\mathcal{D}_v & ::= & \models \mathcal{F}_v \mid \models^? \mathcal{F}_v \mid \not\models \mathcal{F}_v \mid \mathcal{D}_v\ \mathcal{D}_v \\
\mathcal{FV} & ::= & \mathcal{V}\ \mathcal{D}_v \\
\mathcal{D}_w & ::= & \models \mathcal{F}_w \mid \models^? \mathcal{F}_w \mid \not\models \mathcal{F}_w \mid \mathcal{D}_w\ \mathcal{D}_w \\
\mathcal{FW} & ::= & \mathcal{W}\ \mathcal{D}_w \mid \{\mathcal{FV} \ldots \mathcal{FV}\} \mid \{\mathcal{FW} \ldots \mathcal{FW}\}
\end{array}
$$

where :

- $V \models F$ ($W \models F$) corresponds to a world V (resp. multiverse W) where F holds,
- $V \models^? F$ ($W \models^? F$) corresponds to a world V (resp. multiverse W) on which the satisfaction of a formula F being tested for satisfaction, and

– $V \not\models F$ ($W \not\models F$) corresponds to a world V (resp. multiverse W) where F does not hold.

The structural congruence relation on worlds and multiverses extends naturally on guarded worlds and guarded multiverses :

$$V_1\ D \equiv V_2\ D \ \text{ if } \ V_1 \equiv V_2$$
$$W_1\ D \equiv W_2\ D \ \text{ if } \ W_1 \equiv W_2$$

for any D and D' guards on a world or on a multiverse respectively.

Since any formula $F \in \mathcal{F}_v$ can be expressed as a conjunction of formulas from $\mathcal{F}_1 \cup \mathcal{F}_3$, $F = F_1 \wedge \ldots \wedge F_k$, $k \geq 1$, we can decompose the guard considering F as a formula into a juxtaposition of k guards with the formulas $F_1, \ldots, F_k$. For instance,

$$\models^? (F_1 \wedge \ldots \wedge F_k) \quad \equiv \quad \models^? F_1 \ldots \models^? F_k$$

In the following we consider only guards over elementary formulas in $\mathcal{F}_1 \cup \mathcal{F}_3$ and $\mathcal{F}_2 \cup \mathcal{F}_4$ since the satisfaction of a composed formula is reduced to the satisfaction in parallel of all component elementary formulas.

A multiverse guarded by a formula F which does not have the form $\mathrm{A}\varphi$ or $\mathrm{E}\varphi$ is equivalent to the juxtaposition of the component worlds guarded by F :

$$\{W_1 \ldots W_n\} \models^? F \equiv \{W_1 \models^? F \ldots W_n \models^? F\} \text{ if } F \neq \mathrm{A}\varphi \vee F \neq \mathrm{E}\varphi$$

14.2.2 Semantics

We consider that for each evolution step the choice of a strategy to apply among all available in a world is deterministic. This choice must also be fair. If a world contains more than one strategy, we do not want only one strategy to be chosen for interaction ; in particular, if it is not applicable, the system will never evolve, never giving the chance to the other strategies to be (possibly successfully) applied.

Satisfaction of Structural Formulas on a World and on a Multiverse

The satisfaction of a structural formula φ by a port graph molecule in a world can be easily verified by evaluating the encoding strategy $\tau(\varphi)$ on the port graph molecule. We extend the use of a structural formula for a world by defining $[G] \models \varphi$ if $G \models \varphi$. We encode the formula φ as a strategy on a world :

$$S^v_\varphi = [X] \Rightarrow \mathrm{ifThenElse}(\tau(\varphi), X_1 \Rightarrow [X] \models \varphi, X_2 \Rightarrow [X] \not\models \varphi)@X$$

Proposition 16. Let $[G]$ be a world and φ a structural formula. Then $S^v_\varphi@V$ reduces to $[G] \models \varphi$ if $G \models \varphi$ and to $[G] \not\models \varphi$ otherwise.

We consider that a world V guarded by a structural formula φ is equivalent to the guarded world resulting from the application of the strategy S^v_φ on V :

$$V \models^? \varphi \equiv S^v_\varphi@V$$

If the molecular port graph G contained by the world V satisfies the structural formula φ, i.e., $\tau(\varphi)@G \longrightarrow^* \{[G]\}$, then the satisfaction problem $V \models^? \varphi$ is reduced to $V \models \varphi$ by computing $S^v_\varphi@V$. Otherwise, if V does not satisfy φ, i.e., $\tau(\varphi)@G \longrightarrow^* \{[\text{stk}]\}$, then the satisfaction problem $V \models^? \varphi$ is reduced to $V \not\models \varphi$.

When we deal with a multiverse, we need to universally or existentially quantify a structural formula. We use the path quantifiers A and E also for quantifying over the worlds in a multiverse to say that :

- $\text{A}\varphi$ is satisfied by a multiverse W if φ is satisfied in each world from W ;

$$\{V_1 \dots V_n\} \models \text{A}\varphi \Leftrightarrow \forall i,\ 1 \leq i \leq n, V_i \models \varphi$$

- $\text{E}\varphi$ is satisfied by a multiverse W if φ is satisfied in one world from W :

$$\{V_1 \dots V_n\} \models \text{E}\varphi \Leftrightarrow \exists i, 1 \leq i \leq n, V_i \models \varphi$$

We encode the formulas $\text{A}\varphi$ and $\text{E}\varphi$ as strategies on worlds and multiverses. In order to test the satisfiability of φ on W we define a strategy whose application on W reduces to a positive or negative answer. Let this be $S^w_{\text{A}\varphi}$ defined as a left-biased choice between a strategy defined on a multiverse containing a single world and a strategy defined on an arbitrary multiverse :

$$\begin{aligned}
&S^w_{\text{A}\varphi} = \text{first}(S_1, S_2)\\
&S_1 = \{[X]\} \Rightarrow \text{ifThenElse}(\tau(\varphi), X_1 \Rightarrow \{[X]\} \models \text{A}\varphi, X_2 \Rightarrow \{[X]\} \not\models \text{A}\varphi)@X\\
&S_2 = \{[X]\ Z\} \Rightarrow\\
&\qquad \text{ifThenElse}(\tau(\varphi), X_1 \Rightarrow \{[X] \models \varphi\ S^w_{\text{A}\varphi}@Z\}, X_2 \Rightarrow \{[X]\ Z\} \not\models \text{A}\varphi)@X
\end{aligned}$$

We define the strategy $S^w_{\text{E}\varphi}$ for testing the satisfiability of $\text{E}\varphi$ in a multiverse W also as a left-biased choice similarly to $S^w_{\text{A}\varphi}$:

$$\begin{aligned}
&S^w_{\text{E}\varphi} = \text{first}(S_3, S_4)\\
&S_3 = \{[X]\} \Rightarrow \text{ifThenElse}(\tau(\varphi), X_1 \Rightarrow \{[X]\} \models \text{E}\varphi, X_2 \Rightarrow \{[X]\} \not\models \text{E}\varphi)@X\\
&S_4 = \{[X]\ Z\} \Rightarrow\\
&\qquad \text{ifThenElse}(\tau(\varphi), X_1 \Rightarrow \{[X]\ Z\} \models \text{E}\varphi, X_2 \Rightarrow \{[X] \not\models \varphi\ S^w_{\text{E}\varphi}@Z\})@X
\end{aligned}$$

Proposition 17. Let $W = \{[G_1] \dots [G_n]\}$ be a multiverse, and φ a structural formula. Then $S^w_{\text{A}\varphi}@W$ reduces to either $W \models \text{A}\varphi$ if $G_i \models \varphi$ for all i, $1 \leq i \leq n$, and to $W \not\models \text{A}\varphi$ otherwise.

Proposition 18. Let $W = \{[G_1] \dots [G_n]\}$ be a multiverse, and φ a structural formula. Then $S^w_{\text{E}\varphi}@W$ reduces to either $W \models \text{E}\varphi$ if there exists i, $1 \leq i \leq n$, such that $G_i \models \varphi$, and to $W \not\models \text{E}\varphi$ otherwise.

The proofs of the previous two propositions use the result stated by Proposition 14 and the definition of the strategy operator ifThenElse.

Then, similarly for a guarded world, we consider that a multiverse W guarded by a formula $\mathrm{A}\varphi$ or $\mathrm{E}\varphi$ is equivalent to the result of applying the strategy $S^w_{\mathrm{A}\varphi}$ or $S^w_{\mathrm{E}\varphi}$ respectively on the multiverse :

$$W \models^? \mathrm{A}\varphi \equiv S^w_{\mathrm{A}\varphi}@W \qquad\qquad W \models^? \mathrm{E}\varphi \equiv S^w_{\mathrm{E}\varphi}@W$$

The satisfiability test of a structural formula for a multiverse W is propagated by the strategies $S^w_{\mathrm{A}\varphi}$ and $S^w_{\mathrm{E}\varphi}$ inside the multiverse such that we obtain a multiverse of worlds guarded by satisfied or non-satisfied formulas. We then regroup the information on the satisfaction based the following congruences :

$$\begin{array}{l}
\{V \models \varphi \; W \models \mathrm{A}\varphi\} \equiv \{V \; W\} \models \mathrm{A}\varphi \\
\{V \not\models \varphi \; W \models \mathrm{A}\varphi\} \equiv \{V \models \varphi \; W \not\models \mathrm{A}\varphi\} \equiv \{V \; W\} \not\models \mathrm{A}\varphi \\
\{V \not\models \varphi \; W \not\models \mathrm{E}\varphi\} \equiv \{V \; W\} \not\models \mathrm{E}\varphi \\
\{V \not\models \varphi \; W \models \mathrm{E}\varphi\} \equiv \{V \models \varphi \; W \not\models \mathrm{E}\varphi\} \equiv \{V \; W\} \models \mathrm{E}\varphi
\end{array}$$

Reduction Relation on Guarded Worlds and Multiverses

We define now the reduction relation $\Rrightarrow$ between guarded worlds and guarded multiverses. For $V \models^? F$ the initial guarded world, if at any step in the reduction we have $W \models F$ ($W \not\models F$), it means that the initial world V satisfies F (does not satisfy F respectively).

$\boxed{\mathrm{AX}\varphi \text{ and } \mathrm{EX}\varphi}$

If a world V having a guard $\models^? \mathrm{AX}\varphi$ reduces in a step to the multiverse W, the guard reduces as well to $\models \mathrm{AX}\varphi$ if every world in W satisfies φ, or $\not\models \mathrm{AX}\varphi$ otherwise. We reason similarly for a world guarded by $\models^? \mathrm{EX}\varphi$ by testing if the multiverse the world reduces to in a step contains a world satisfying φ.

$$V \models^? \mathrm{AX}\varphi \Rrightarrow \mathrm{first}(W \models \mathrm{A}\varphi \Rightarrow W \models \mathrm{AX}\varphi, W \not\models \mathrm{A}\varphi \Rightarrow W \not\models \mathrm{AX}\varphi)@(W \models^? \mathrm{A}\varphi) \quad \text{if } V \mapsto W \tag{14.1}$$

$$V \models^? \mathrm{EX}\varphi \Rrightarrow \mathrm{first}(W \models \mathrm{E}\varphi \Rightarrow W \models \mathrm{EX}\varphi, W \not\models \mathrm{E}\varphi \Rightarrow W \not\models \mathrm{EX}\varphi)@(W \models^? \mathrm{E}\varphi) \quad \text{if } V \mapsto W \tag{14.2}$$

$\boxed{\mathrm{AG}\varphi}$

If a formula $\mathrm{AG}\varphi$ is being tested on a simple world $[G]$ then in the next state it remains under testing if the formula φ is satisfied by G, or it becomes unsatisfied otherwise.

$$[G] \models^? \mathrm{AG}\varphi \Rrightarrow \mathrm{ifThenElse}(\tau(\varphi), X_1 \Rightarrow W \models^? \mathrm{AG}\varphi, X_2 \Rightarrow W \not\models \mathrm{AG}\varphi)@G \quad \text{if } [G] \mapsto W \tag{14.3}$$

Remarque 9. The satisfaction problem of a formula of the type AGφ, EGφ, AFφ, or EFφ for a world $[G]$ irreducible with respect to the reduction relation $\mapsto$ is equivalent to the satisfaction problem of the structural formula φ for G. If we consider for instance the formula AGφ and the irreducible world $[G]$, then we have :

$$[G] \models^{?} \mathrm{AG}\varphi \equiv \mathrm{ifThenElse}(\tau(\varphi), X_1 \Rightarrow [G] \models \mathrm{AG}\varphi, X_2 \Rightarrow [G] \not\models \mathrm{AG}\varphi)@G$$
$$\text{if } [G] \text{ is } \mapsto\text{-irreducible}$$

Since a formula guarding a multiverse is distributed to each component worlds, at any moment we can regroup the information on the satisfaction of the formula on each world in order to find the satisfaction information on the multiverse in the following situations :

$$\{V \not\models \mathrm{AG}\varphi\ W \models^{?} \mathrm{AG}\varphi\} \equiv \{V \models^{?} \mathrm{AG}\varphi\ W \not\models \mathrm{AG}\varphi\} \equiv \{V\ W\} \not\models \mathrm{AG}\varphi$$
$$\{V \not\models \mathrm{AG}\varphi\ W \models \mathrm{AG}\varphi\} \equiv \{V \models \mathrm{AG}\varphi\ W \not\models \mathrm{AG}\varphi\} \equiv \{V\ W\} \not\models \mathrm{AG}\varphi$$
$$\{V \not\models \mathrm{AG}\varphi\ W \not\models \mathrm{AG}\varphi\} \equiv \{V\ W\} \not\models \mathrm{AG}\varphi$$
$$\{V \models \mathrm{AG}\varphi\ W \models \mathrm{AG}\varphi\} \equiv \{V\ W\} \models \mathrm{AG}\varphi$$

EGφ

The reduction in the ρ^{v}_{pg}-calcul of a world guarded by a formula EGφ is similar to the reduction of a world guarded by AGφ. This similarity is due to the reasoning at the level of one world, hence the universal and existential quantification are equivalent. Obviously, this kind of similarity no longer happens in the case of a guarded mutiverse.

$$[G] \models^{?} \mathrm{EG}\varphi \Rrightarrow \mathrm{ifThenElse}(\tau(\varphi), X_1 \Rightarrow W \models^{?} \mathrm{EG}\varphi, X_2 \Rightarrow W \not\models \mathrm{EG}\varphi)@G$$
$$\text{if } [G] \mapsto W \tag{14.4}$$

The information on the satisfaction or in-satisfaction of a formula EGφ in a multiverse is regrouped as follows :

$$\{V \not\models \mathrm{EG}\varphi\ W \not\models \mathrm{EG}\varphi\} \equiv \{V\ W\} \not\models \mathrm{EG}\varphi$$
$$\{V \not\models \mathrm{EG}\varphi\ W \models \mathrm{EG}\varphi\} \equiv \{V \models \mathrm{EG}\varphi\ W \not\models \mathrm{EG}\varphi\} \equiv \{V\ W\} \models \mathrm{EG}\varphi$$

AFφ

When testing the satisfiability of a formula AFφ on a world $[G]$, if G satisfies φ, then we conclude that the formula is satisfied at this point, otherwise the formula continues to be tested.

$$[G] \models^{?} \mathrm{AF}\varphi \Rrightarrow$$
$$\mathrm{ifThenElse}(\tau(\varphi), X_1 \Rightarrow W \models \mathrm{AF}\varphi, X_2 \Rightarrow W \models^{?} \mathrm{AF}\varphi)@G \text{ if } [G] \mapsto W \tag{14.5}$$

We reassemble the information on the satisfaction of a formula AFφ on a multiverse as follows :

$$\{V \models \mathrm{AF}\varphi\ W \models \mathrm{AF}\varphi\} \equiv \{V\ W\} \models \mathrm{AF}\varphi$$
$$\{V \models \mathrm{AF}\varphi\ W \not\models \mathrm{AF}\varphi\} \equiv \{V \not\models \mathrm{AF}\varphi\ W \models \mathrm{AF}\varphi\} \equiv \{V\ W\} \not\models \mathrm{AF}\varphi$$
$$\{V \not\models \mathrm{AF}\varphi\ W \not\models \mathrm{AF}\varphi\} \equiv \{V\ W\} \not\models \mathrm{AF}\varphi$$
$$\{V \models^? \mathrm{AF}\varphi\ W \not\models \mathrm{AF}\varphi\} \equiv \{V \not\models \mathrm{AF}\varphi\ W \models^? \mathrm{AF}\varphi\} \equiv \{V\ W\} \not\models \mathrm{AF}\varphi$$

$\boxed{\mathrm{EF}\varphi}$

The reduction in ρ^v_{pg}-calcul of a world guarded by a formula EFφ is similar to the reduction of a world guarded by AFφ on the same basis as for EGφ and AGφ.

$$[G] \models^? \mathrm{EF}\varphi \Rrightarrow$$
$$\mathrm{ifThenElse}(\tau(\varphi), X_1 \Rightarrow W \models \mathrm{EF}\varphi, X_2 \Rightarrow W \models^? \mathrm{EF}\varphi)@G \text{ if } [G] \mapsto W \qquad (14.6)$$

The satisfaction information for EFφ is grouped in a multiverse as follows :

$$\{V \models \mathrm{EF}\varphi\ W \models^? \mathrm{EF}\varphi\} \equiv \{V \models^? \mathrm{EF}\varphi\ W \models \mathrm{EF}\varphi\} \equiv \{V\ W\} \models \mathrm{EF}\varphi$$
$$\{V \models \mathrm{EF}\varphi\ W \not\models \mathrm{EF}\varphi\} \equiv \{V \not\models \mathrm{EF}\varphi\ W \models \mathrm{EF}\varphi\} \equiv \{V\ W\} \models \mathrm{EF}\varphi$$
$$\{V \models \mathrm{EF}\varphi\ W \models \mathrm{EF}\varphi\} \equiv \{V\ W\} \models \mathrm{EF}\varphi$$
$$\{V \not\models \mathrm{EF}\varphi\ W \not\models \mathrm{EF}\varphi\} \equiv \{V\ W\} \not\models \mathrm{EF}\varphi$$

$\boxed{\mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)}$

If the second structural formula φ_2 is satisfied by G, then the formula A(φ_1Uφ_2) is satisfied. Otherwise, if in addition φ_1 is satisfied by G then we are still in a state where the first property holds whereas the second does not, hence we keep testing the formula in the next states. If both structural formulas φ_1 and φ_2 are do not hold in G, then the formula A(φ_1Uφ_2) does not hold.

$$\begin{aligned}
&[G] \models^? \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2) \Rrightarrow \\
&\mathrm{ifThenElse}(\tau(\varphi_2), \\
&\qquad X_1 \Rightarrow W \models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2), \\
&\qquad X_2 \Rightarrow (\mathrm{ifThenElse}(\tau(\varphi_1), \\
&\qquad\qquad X_3 \Rightarrow W \models^? \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2), \\
&\qquad\qquad X_4 \Rightarrow W \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2))@G))@G \\
&\quad \text{if } [G] \mapsto W
\end{aligned} \qquad (14.7)$$

If the current world $[G]$ is irreducible with respect to $\mapsto$, then the formula A(φ_1Uφ_2) holds if and only if φ_2 holds :

$$\begin{aligned}
&[G] \models^? \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2) \equiv \\
&\mathrm{ifThenElse}(\tau(\varphi_2), X_1 \Rightarrow [G] \models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2), X_2 \Rightarrow [G] \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2))@G \\
&\quad \text{if } [G] \text{ is } \mapsto\text{-irreducible}
\end{aligned}$$

We reassemble partial information on the satisfaction of a formula $\mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)$ inside a multiverse :

$$\begin{aligned}
\{V \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\ W \models^? \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\} &\equiv \{V \models^? \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\ W \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\} \\
&\equiv \{V\ W\} \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2) \\
\{V \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\ W \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\} &\equiv \{V\ W\} \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2) \\
\{V \models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\ W \models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\} &\equiv \{V\ W\} \models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2) \\
\{V \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\ W \models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\} &\equiv \{V \models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\ W \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)\} \\
&\equiv \{V\ W\} \not\models \mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)
\end{aligned}$$

$\boxed{\mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)}$

Again the reasoning over the satisfaction of the formula $\mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)$ on a world is similar to the one for the counterpart universally quantified formula.

$$\begin{aligned}
&[G] \models^? \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2) \Rrightarrow \\
&\mathrm{ifThenElse}(\tau(\varphi_2), \\
&\qquad X_1 \Rightarrow W \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2), \\
&\qquad X_2 \Rightarrow (\mathrm{ifThenElse}(\tau(\varphi_1), \\
&\qquad\qquad X_3 \Rightarrow W \models^? \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2), \\
&\qquad\qquad X_4 \Rightarrow W \not\models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2))@G))@G \\
&\quad \text{if } [G] \Mapsto W
\end{aligned} \tag{14.8}$$

The result on the satisfaction of $\mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)$ on a $\Mapsto$-irreducible world is similar to case of $\mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)$.

The satisfaction of a formula $\mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)$ on a multiverse is deduced from the satisfaction information on the component worlds as follows :

$$\begin{aligned}
\{V \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\ W \models^? \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\} &\equiv \{V \models^? \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\ W \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\} \\
&\equiv \{V\ W\} \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2) \\
\{V \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\ W \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\} &\equiv \{V \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\ W \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\} \\
&\equiv \{V\ W\} \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2) \\
\{V \not\models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\ W \not\models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\} &\equiv \{V\ W\} \not\models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2) \\
\{V \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\ W \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)\} &\equiv \{V\ W\} \models \mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)
\end{aligned}$$

$\boxed{\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)}$

The formula $\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)$ does not hold if both φ_1 and φ_2 hold or do not hold in the same time. If φ_2 holds in the current world $[G]$ and φ_1 does not, then the formula $\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)$ continues to be tested in the next state. Whereas if φ_2 does not hold in the current world, but held in the previous one if we are not in the initial state,

and φ_1 holds, then $\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)$ holds.

$$
\begin{array}{l}
[G] \models^{?} \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2) \Rrightarrow \\
\mathrm{ifThenElse}(\tau(\varphi_2), \\
\qquad X_1 \Rightarrow (\mathrm{ifThenElse}(\tau(\varphi_1), \\
\qquad\qquad X_2 \Rightarrow W \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2), \\
\qquad\qquad X_3 \Rightarrow W \models^{?} \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2))@G), \\
\qquad X_4 \Rightarrow \mathrm{ifThenElse}(\tau(\varphi_1), \\
\qquad\qquad X_5 \Rightarrow W \models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2), \\
\qquad\qquad X_6 \Rightarrow W \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2))@G))@G \\
\qquad \text{if } [G] \Mapsto W
\end{array}
\tag{14.9}
$$

If the current world $[G]$ is irreducible with respect to $\Mapsto$, then the formula $\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)$ holds if and only if φ_2 holds and φ_1 does not hold :

$$
\begin{array}{l}
[G] \models^{?} \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2) \equiv \\
\mathrm{ifThenElse}(\tau(\neg\varphi_1 \wedge \varphi_2), X_1 \Rightarrow [G] \models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2), X_2 \Rightarrow [G] \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2))@G \\
\qquad \text{if } [G] \text{ is } \Mapsto\text{-irreducible}
\end{array}
$$

The information on the satisfaction of a formula $\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)$ inside a multiverse is regrouped as follows :

$$
\begin{array}{l}
\{V \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\ W \models^{?} \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\} \equiv \{V \models^{?} \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\ W \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\} \\
\qquad\qquad\qquad\qquad\qquad\qquad \equiv \{V\ W\} \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2) \\
\{V \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\ W \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\} \equiv \{V\ W\} \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2) \\
\{V \models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\ W \models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\} \equiv \{V\ W\} \models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2) \\
\{V \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\ W \models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\} \equiv \{V \models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\ W \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)\} \\
\qquad\qquad\qquad\qquad\qquad\qquad \equiv \{V\ W\} \not\models \mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)
\end{array}
$$

$\boxed{\mathrm{E}(\varphi_1 \mathrm{R} \varphi_2)}$

The reduction is defined similarly to the one for $\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)$ where we replace the formula $\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)$ by $\mathrm{E}(\varphi_1 \mathrm{R} \varphi_2)$.

Model

If we instantiate an abstract reduction system with port graph molecules as objects and the big-one-step reduction relation from the ρ_{pg}-calcul as the transition relation, we obtain a higher-order port graph reduction system. We stress that the obtained reduction system is of higher-order in order to distinguish from the port graph reduction system associated to a port graph rewriting system. In the following we define a model for the CTL based on a higher-order port graph reduction system.

Définition 69 (Model). A model over $\nabla^{\mathcal{X}}$ for the ρ_{pg}-calcul is a Kripke structure system $\mathcal{K}_{pg} = (\mathcal{Q}, \Mapsto^{\bullet}, L_{\mathcal{Q}})$ where :

- $(\mathcal{Q}, \Mapsto)$ is a higher-order port graph reduction with :
  - $\mathcal{Q}$ is the set of states, each state consisting of a multiverse ;
  - $\Mapsto^{\bullet}$ is the big-step reduction relation of the ρ_{pg}-calcul which adds a self-loop for every irreducible state ;
- $L_{\mathcal{Q}} : \mathcal{Q} \to \mathcal{P}(\mathcal{FS}(\nabla^{\mathcal{X}}))$ that maps to each state W the set of structural formulas it satisfies, i.e., $L_{\mathcal{Q}}(W) = \{\mathrm{A}\varphi \mid \varphi \in \mathcal{FS}(\nabla^{\mathcal{X}})$ such that $W \models \mathrm{A}\varphi\} \cup \{\mathrm{E}\varphi \mid \varphi \in \mathcal{FS}(\nabla^{\mathcal{X}})$ such that $W \models \mathrm{E}\varphi\}$.

In the following we define the satisfaction of a formula ϕ in a model $\mathcal{K}_{pg}$ with W the initial state :

- for $\phi \in \mathcal{F}_3$, we have $\mathcal{K}_{pg}, W \models \phi$ if there exists W' irreducible with respect to $\Mapsto$ such that $W \models^{?} \phi \Rrightarrow^{*} W' \models \phi$;
- for ϕ either $\mathrm{AG}\varphi$, $\mathrm{A}(\varphi_1 \mathrm{U} \varphi_2)$, or $\mathrm{A}(\varphi_1 \mathrm{R} \varphi_2)$, we have $\mathcal{K}_{pg}, W \models \phi$ until reaching any state W' such that $W \models^{?} \phi \Rrightarrow^{*} \{W_1 \models^{?} \phi \dots W_n \models^{?} \phi\}$ and $W' \equiv \{W_1 \dots W_n\}$;
- for ϕ either $\mathrm{EG}\varphi$, $\mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)$, or $\mathrm{E}(\varphi_1 \mathrm{R} \varphi_2)$, we have $\mathcal{K}_{pg}, W \models \phi$ until reaching any state W' such that $W \models^{?} \phi \Rrightarrow^{*} \{W_1 \models^{?} \phi \ FW\}$ and W' consists of W_1 and all multiverses occurring in the guarded multiverse FW ;
- for ϕ either $\mathrm{AF}\varphi$ or $\mathrm{EF}\varphi$, we have $\mathcal{K}_{pg}, W \models \phi$ if there exists W' such that $W \models^{?} \phi \Rrightarrow^{*} W' \models \phi$;
- for ϕ either $\mathrm{AX}\varphi$ or $\mathrm{EX}\varphi$, we have $\mathcal{K}_{pg}, W \models \phi$ if there exist W' such that $W \models^{?} \phi \Rrightarrow W' \models \phi$.

14.2.3 Application in Modeling Autonomous Systems

In an autonomous system, if a failure occurs, the system has the capability of auto-repairing himself. In the context of the ρ^{v}_{pg}-calcul, a failure can be tested by the insatisfaction of a temporal formula. Then, as soon as the formula is marked as unsatisfied, a reparation rule becomes applicable and it may remove the unsatisfied formula, introduce some new port graph objects or strategies or modify the existing one, or even add a new formula for guarding the execution from that moment on.

In a first step, when an evaluation rule of the ρ^{v}_{pg}-calcul finds the satisfaction or insatisfaction of a formula, this information has to be pumped into the current worlds. Let us consider for instance the rule (14.3) concerning the reduction with the formula $\mathrm{AG}\varphi$. We replace the rule $X_2 \Rightarrow W \not\models \mathrm{AG}\varphi$ by $X_2 \Rightarrow (X' \Rightarrow (X' \not\models \mathrm{AG}\varphi))@W$. This is possible if we extend the syntax of the calculus by adding guards from $\mathcal{D}_v$ in worlds.

Then, each world has to be equipped with a strategy to handle such information on the (in)satisfaction of a formula. A typical example of an abstraction handling the insatisfaction of a formula ϕ is $X \not\models \varphi \Rightarrow G\ X \models^{?} \phi$ where G may consist of object port graphs or strategies for repairing the problem that produced the insatisfaction of the formula. Therefore each formula guarding the evolution of the modeled system should come with the corresponding port graph molecules for repairing or improving the system. Let us see below two examples from a biological system where the port graph respect the conditions imposed for molecular graphs :

1. We can handle the insatisfaction of an invariant stating that there is no virus described by a structural formula Virus ; then the invariant has the form AG¬Virus and the associated repairing strategy could be :

$$X \not\models \text{AG}\neg\text{Virus} \Rightarrow X\text{Antibiotic} \models^{?} \text{AG}\neg\text{Virus}$$

   which introduces an antibiotic in the system, or, something simpler :

$$X\text{Virus} \not\models \text{AG}\neg\text{Virus} \Rightarrow X \models^{?} \text{AG}\neg\text{Virus}$$

   which removes the virus.

2. A formula of the type $\text{A}(\varphi_1\text{U}\varphi_2)$ can be used to ensure the constant existence of a signal protein described by φ_1 before a molecular complex described by φ_2 is formed. If at any moment we have $V \not\models \text{A}(\varphi_1\text{U}\varphi_2)$, it means that the signal protein disappeared from the system. Then a repairing strategy would be :

$$X \not\models \text{A}(\varphi_1\text{U}\varphi_2) \Rightarrow X\ M \models^{?} \text{A}(\varphi_1\text{U}\varphi_2)$$

   which introduces the molecule M for ensuring the presence of the signal (for M such that $M \models \varphi_1$).

These examples illustrate our initial motivation of integrating temporal formulas to worlds and multiverses such that we can not only see if a formula is satisfied, but also to give the possibility to the system to react to such information, either positive or negative, i.e., to self-manage its behavior as response to some tests.

14.3 Conclusions

In this chapter we have embedded in the ρ_{pg}-calcul a runtime verification technique for a particular set of CTL formulas. As a consequence, beyond a simulation formalism for a dynamic systems, the obtained calculus, the ρ^{v}_{pg}-calcul, provides an automatic method for modeling self-management properties. This result can be easily generalized for the Abstract Biochemical Calculus provided that we make precise the set of structural formulas for abstract molecules.

There are several extensions worth considering for increasing the expressivity of the ρ^{v}_{pg}-calcul. We summarize them in the following.

The structural formulas for the ρ^{v}_{pg}-calcul do not contain a port graph expression γ. A formula γ supposes a matching of the current state with the pattern described by γ, which translates into the global application concept already mentioned in the possible extensions for the semantics of the ρ_{pg}-calcul, but not currently supported. Whereas a formula $\diamond\gamma$ supposes a submatching verification, hence a local application of a particular abstraction as defined in the ρ_{pg}-calcul. A solution of this problem is to consider the possible approach mentioned in the perspectives of the Chapter I, i.e, to consider two ways of applying an abstraction on a molecule : locally (corresponding to the current approach by solving a submatching problem) or/and globally (corresponding to a matching problem). If we

allow the two types of application by considering two distinct operators for application (to see if this information should be encoded in the abstraction itself to be then passed to the application operator), then we can verify a formula γ on a port graph G by applying globally on G the strategy $\gamma \Rightarrow \gamma$; then for the somewhere modality, i.e., $\Diamond\gamma$, the strategy $\gamma \Rightarrow \gamma$ should be applied locally.

For the current form of the ρ^v_{pg}-calcul we have considered only a particular set of CTL formulas. In a first stage we should investigate which formulas from CTL can be added as guards in the ρ^v_{pg}-calcul. Then, in a second stage, we should move the more expressive logic CTL* and investigate the possibility of embedding the satisfaction relation for some formulas in the calculus.

A Implementation of the EGFR Signaling Pathway Fragment using TOM

The GOM Syntax for Molecular Graphs

A node is specified by an identifier, a name and a list of ports, a port by a name, a list of neighbors, and a state, a neighbor by a reference to node and a set of pointers to ports. The node substitution is a pair of a node and a list of nodes ; it corresponds to an elementary mapping defined by a node morphism. A port graph is then a list of nodes and a structure collects all possible resulting port graph during rewriting. Each list is constructed using a associative variadic operator with neutral element.

```
module term
  imports String int
  abstract syntax

  Structure = struct( Structure* )
            | Nodes( n:NodeList )

  NodeList = concN( Node* )
  Node = node( id:Uid, plist:PortList )

  Uid = uid( uid:String, name:String )

  PortList = concP( Port* )
  Port = port( name:String, neighlist:NeighbourList, state:State )
  State = b() | v() | h()

  NeighbourList = concNG( Neighbour* )
  Neighbour = neighbour( node:Node, ports:PortList )

  NodeSubstitution = nodeSubstitution( before:Node, after:NodeList)
  NodeSubstitutionList = concNS( NodeSubstitution* )
```

In a molecular graph, a port (or site) can only be connected to maximum one other port. We consider a list of neighbors for a port in order to cover the case when the port is not connected which corresponds to an empty list of neighbors, concNG(). The restriction of the incidence degree on a port is maintained thanks to the state associated to the port. If the state of a port is b() for bound, it means that the port is already connected, and in consequence it is not considered for an interaction where it would be connect to another port.

When a node list is constructed or an insertion or appending operation is performed on a node list, we make sure that no duplicates are introduced. This is possible by adding hooks for modifying the creation operation for a list of nodes. In a similar way we forbid the insertion of duplicates in lists of ports and neighbors.

```
concN:make_insert(node, nodes) {
    %match(Node node, NodeList nodes) {
        n@LabNode(i, node(uid(i,name), _)),
        concN(x@LabNode(i1, node(uid(i1,name1), _)), list*) -> {
          if (‘name.compareTo(‘name1) < 0) {
              return ‘realMake(n, realMake(x, list*));
          }
          else if (‘name.compareTo(‘name1) > 0) {
              return ‘realMake(x, concN(n, list*));
          }
          else {
              if (‘i.compareTo(‘i1) < 0) {
                return ‘realMake(n, realMake(x, list*));
              }
              else {
                return ‘realMake(x, realMake(n, list*));
              }
          }
        }
        n@node(uid(_,name), _),
        concN(x@node(uid(_,name1), _), list*) -> {
     if (‘name.compareTo(‘name1) <= 0) {
              return ‘realMake(n, realMake(x, list*));
           }
           else {
              return ‘realMake(x, concN(n, list*));
           }
        }
    }
}
```

For restructuring the terms after an arrow translation operation we use the rules hook. This construct enables to define a set of conditional rewrite rules over the current module signature. These rules are applied systematically using a leftmost innermost strategy. Thus, the only terms that can be produced and manipulated in the Tom program are normal with respect to the defined system.

```
module term:rules() {

  // merge the port lists of two identical nodes
  concN(a*, LabNode(id, node(uid(id, name), concP(sl1*))), b*,
       LabNode(id, node(uid(id, name), concP(sl2*))), c*)
    ->
  concN(a*, LabNode(id, node(uid(id, name), concP(sl1*, sl2*))), b*, c*)
```

```
  // merge the neighbours of the same port
  concP(a*, LabPort(id, port(p, concNG(nl1*), s)), b*,
        LabPort(id, port(p, concNG(nl2*), s)))
    ->
  concP(a*, LabPort(id, port(p, concNG(nl1*, nl2*), s)), b*)

  // merge the references to ports of two identical neighbours
  concNG(nl1*, neighbour(RefNode(x), concP(sl1*)), nl2*,
         neighbour(RefNode(x), concP(sl2*)), nl3*)
    ->
  concNG(nl1*, neighbour(RefNode(x), concP(sl1*, sl2*)), nl2*, nl3*)

  // remove a neighbour with an empty list of ports
  concNG(nl1*, neighbour(RefNode(x), concP()), nl2*)
    ->
  concNG(nl1*, nl2*)

  // eliminate duplicates in a structure of nodes
  struct(a*, x, b*, x, c*) -> struct(a*, x, b*, c*)
}
```

A signal (EGF), a receptor (EGFR) and an adapter (SHC) are represented in TOM as follows :

```
node(uid("1", "EGF"), concP(port("s1", concNG(), v())),
                      port("s2", concNG(), h()))

node(uid("5", "EGFR"), concP(port("s1", concNG(), v()),
                       port("s2", concNG(), v()),
                       port("s3", concNG(), h()),
                       port("s4", concNG(), h())))

node(uid("7", "SHC"), concP(port("s1", concNG(), v()),
                      port("s2", concNG(), h())))
```

The initial molecular graph contains in addition to the above proteins three more signal proteins identified by uid("2", "EGF"), uid("3", "EGF"), uid("4", "EGF"), and a receptor protein uid("6", "EGFR").

For each protein we add labels to each node and port in order to use pointers for representing bonds. For instance, a signal protein is labeled as follows :

```
Node nodeEGF1 =
    LabNode("1", node(uid("1", "EGF"),
                  concP(LabPort("s1", port("s1", concNG(), v())),
                        LabPort("s2", port("s2", concNG(), h())))));
```

The molecular graph is constructed using seven nodes with labels as above :

```
Structure mgraph = `struct(Nodes(concN(nodeEGF1,nodeEGF2,nodeEGF3,nodeEGF4,
                                 nodeEGFR5, nodeEGFR6, nodeSHC7)));
```

Arrow Translation

We implement the application of a node morphism on both on the source and the target of an edge based on the rules described in Figure 13.2. For each pair of node – set of nodes in the node morphism, we first reduce the term with a strategy innermost for applying the node substitution on the sources, and then, in a second step, we reduce the obtained term using the innermost strategy for the targets.

```
NodeList interm = rhs;

%match(NodeSubstitutionList nodeSubsts) {
  concNS(_*, x, _*) -> {
    try {
     interm = (NodeList)‘InnermostId(ApplyNodeSubst1(x)).visit(interm);
     interm = (NodeList)‘InnermostId(ApplyNodeSubst2(x)).visit(interm);
    } catch (VisitFailure e) {
System.out.println("Error at ApplyNodeSubst: " + e);
    }
   }
 }

 return interm;
```

We do not give the details here of the strategies ApplyNodeSubst1(e:NodeSubstitution) and ApplyNodeSubst2(e:NodeSubstitution) since they are too technical.

Reaction Rules

For each of the five reaction patterns we followed the same steps in the implementation. We give some details here for the first reaction, the dimerization of two EGF proteins.

The reaction pattern as described graphically in Figure 12.7 is implemented as the following strategy :

```
%strategy EGFDimerizationStrat(c:Collection) extends Identity() {
 visit Structure {
  Nodes(concN(
   nl1*,
 LabNode(i, node(uid(i,"EGF"),
                 concP(LabPort("s1", port("s1", concNG(), v())),
                       LabPort("s2", port("s2", concNG(), h())))))),
       nl2*,
       LabNode(j, node(uid(j,"EGF"),
                 concP(LabPort("s1", port("s1", concNG(), v())),
                 LabPort("s2", port("s2", concNG(), h())))))),
       nl3*)) -> {

      NodeSubstitutionList nodeSubsts = ‘concNS();
      NodeList rhs =
        ‘concN(
```

```
        nl1*,
        LabNode(i, node(uid(i,"EGF"),
                concP(LabPort("s1",
                    port("s1", concNG(neighbour(RefNode(j),
                                concP(RefPort("s1")))), b())),
                    LabPort("s2", port("s2", concNG(), v()))))),
        nl2*,
        LabNode(j, node(uid(j,"EGF"),
                concP(LabPort("s1",
                    port("s1", concNG(neighbour(RefNode(i),
                                concP(RefPort("s1"))),b())),
                    LabPort("s2", port("s2", concNG(), v()))))),
        nl3*);

       c.add(`Nodes(nodeSubstApp.applyAll(rhs, nodeSubsts)));
      }
    }
}
```

It searches two nodes EGF with the first port visible, and it connects them by adding to each visible port a reference to the other one. The collection taken as argument is used for gathering all possible results of the matching. In the implementation for each rule we provide as well the associated node morphism, hence there is no need to define the procedure GetMap used in the term encoding defined in Chapter 13. In the rule described above the node morphism is the identity, hence it corresponds to an empty list of elementary mappings (or node substitutions as they are called in the implementation).

This strategy is included in a class EGFDimerization. This class contains also a method apply which takes a structure and returns it reduced with the strategy EGFDimerization.

Then in the main Java program we have the following strategy that tries to apply the first reaction rule and returns the structure of all possible results if any, otherwise it returns the input structure :

```
%strategy Rule1() extends Identity() {
  visit Structure {
    x@Nodes[] -> {
      EGFDimerization r = new EGFDimerization();
      Structure s = `struct(r.apply(x));
      %match(s) {
        struct() -> { return `x; }
        _ -> { return s; }
      }
    }
  }
}
```

In a similar way we implement the strategies Rule2, Rule3, Rule4, Rule5 for the reaction patterns r2, r3, r4, r5 respectively.

Generating the Biochemical Network using Strategies

One strategy that generates the biochemical networks is :

$$\text{seq}(\text{repeat}(\text{first}(\text{r2},\ \text{r1})),\ \text{seq}(\text{r3},\ \text{seq}(\text{r4},\ \text{r5})))$$

Taking into account that in TOM the sequence strategy operator is variadic and that Choice is the same as first, the implementation of this strategy is the following :

```
Structure result = (Structure)`Sequence(
                        RepeatId(_struct(ChoiceId(Rule2(), Rule1()))),
                        RepeatId(_struct(Rule3())),
                        RepeatId(_struct(Rule4())),
                        RepeatId(_struct(Rule5()))).visit(mgraph);
```

We tested other strategies mentioned in the Chapter 12. For instance, after reducing only with the first strategy Rule1() we obtained three possible molecular graphs corresponding to the possibilities of connected two by two the EGF proteins.

Let us present here the result. We implemented also in the TOM program a pretty-printer method, hence the initial molecular graph is printed as follows :

```
<1:EGF||(s1->v),(s2->h)>, <2:EGF||(s1->v),(s2->h)>,
<3:EGF||(s1->v),(s2->h)>, <4:EGF||(s1->v),(s2->h)>,
<5:EGFR||(s1->v),(s2->v),(s3->h),(s4->h)>,
<6:EGFR||(s1->v),(s2->v),(s3->h),(s4->h)>, <7:SHC||(s1->v),(s2->h)>
```

The successful result of reducing the initial molecular graphs with the strategy Rule1() is :

```
1)
<1:EGF||(s1->2^s1),(s2->v)>,
<2:EGF||(s1->1^s1),(s2->v)>, <3:EGF||(s1->4^s1),(s2->v)>,
<4:EGF||(s1->3^s1),(s2->v)>,
<5:EGFR||(s1->v),(s2->v),(s3->h),(s4->h)>,
<6:EGFR||(s1->v),(s2->v),(s3->h),(s4->h)>, <7:SHC||(s1->v),(s2->h)>
2)
<1:EGF||(s1->3^s1),(s2->v)>, <2:EGF||(s1->4^s1),(s2->v)>,
<3:EGF||(s1->1^s1),(s2->v)>, <4:EGF||(s1->2^s1),(s2->v)>,
<5:EGFR||(s1->v),(s2->v),(s3->h),(s4->h)>,
<6:EGFR||(s1->v),(s2->v),(s3->h),(s4->h)>, <7:SHC||(s1->v),(s2->h)>
3)
<1:EGF||(s1->4^s1),(s2->v)>, <2:EGF||(s1->3^s1),(s2->v)>,
<3:EGF||(s1->2^s1),(s2->v)>, <4:EGF||(s1->1^s1),(s2->v)>,
<5:EGFR||(s1->v),(s2->v),(s3->h),(s4->h)>,
<6:EGFR||(s1->v),(s2->v),(s3->h),(s4->h)>, <7:SHC||(s1->v),(s2->h)>
```

The result of the biochemical network generation consists of 24 molecular graphs. We give below the first ones.

1)
<1:EGF||(s1->2^s1),(s2->v)>, <2:EGF||(s1->1^s1),(s2->v)>,
<3:EGF||(s1->4^s1),(s2->5^s1)>, <4:EGF||(s1->3^s1),(s2->6^s1)>,
<5:EGFR||(s1->3^s2),(s2->6^s2),(s3->7^s1),(s4->h)>,
<6:EGFR||(s1->4^s2),(s2->5^s2),(s3->v),(s4->h)>,
<7:SHC||(s1->5^s3),(s2->v)>
2)
<1:EGF||(s1->2^s1),(s2->v)>, <2:EGF||(s1->1^s1),(s2->v)>,
<3:EGF||(s1->4^s1),(s2->5^s1)>, <4:EGF||(s1->3^s1),(s2->6^s1)>,
<5:EGFR||(s1->3^s2),(s2->6^s2),(s3->v),(s4->h)>,
<6:EGFR||(s1->4^s2),(s2->5^s2),(s3->7^s1),(s4->h)>,
<7:SHC||(s1->6^s3),(s2->v)>
3)
<1:EGF||(s1->2^s1),(s2->v)>, <2:EGF||(s1->1^s1),(s2->v)>,
<3:EGF||(s1->4^s1),(s2->6^s1)>, <4:EGF||(s1->3^s1),(s2->5^s1)>,
<5:EGFR||(s1->4^s2),(s2->v),(s3->7^s1),(s4->h)>,
<6:EGFR||(s1->3^s2),(s2->v)> &&(s2->v),(s3->v),(s4->h)>,
<7:SHC||(s1->5^s3),(s2->v)>
4)
<1:EGF||(s1->2^s1),(s2->v)>, <2:EGF||(s1->1^s1),(s2->v)>,
<3:EGF||(s1->4^s1),(s2->6^s1)>, <4:EGF||(s1->3^s1),(s2->5^s1)>,
<5:EGFR||(s1->4^s2),(s2->v),(s3->v),(s4->h)>,
<6:EGFR||(s1->3^s2),(s2->v),(s3->7^s1),(s4->h)>,
<7:SHC||(s1->6^s3),(s2->v)>
5)
<1:EGF||(s1->3^s1),(s2->5^s1)>, <2:EGF||(s1->4^s1),(s2->v)>,
<3:EGF||(s1->1^s1),(s2->6^s1)>, <4:EGF||(s1->2^s1),(s2->v)>,
<5:EGFR||(s1->1^s2),(s2->6^s2),(s3->7^s1),(s4->h)>,
<6:EGFR||(s1->3^s2),(s2->5^s2),(s3->v),(s4->h)>,
<7:SHC||(s1->5^s3),(s2->v)>
6)
<1:EGF||(s1->3^s1),(s2->5^s1)>, <2:EGF||(s1->4^s1),(s2->v)>,
<3:EGF||(s1->1^s1),(s2->6^s1)>, <4:EGF||(s1->2^s1),(s2->v)>,
<5:EGFR||(s1->1^s2),(s2->6^s2),(s3->v),(s4->h)>,
<6:EGFR||(s1->3^s2),(s2->5^s2),(s3->7^s1),(s4->h)>,
<7:SHC||(s1->6^s3),(s2->v)>
7)
...

Bibliographie

[ACL06] Oana Andrei, Gabriel Ciobanu and Dorel Lucanu – « Expressing Control Mechanisms in P systems by Rewriting Strategies », Workshop on Membrane Computing (H. J. Hoogeboom, G. Paun, G. Rozenberg and A. Salomaa, eds.), Lecture Notes in Computer Science, vol. 4361, Springer, 2006, p. 154–169. 113

[AIK06] Oana Andrei, Liliana Ibanescu and Hélène Kirchner – « Non-intrusive Formal Methods and Strategic Rewriting for a Chemical Application. », Algebra, Meaning, and Computation, Essays Dedicated to Joseph A. Goguen on the Occasion of His 65th Birthday (K. Futatsugi, J.-P. Jouannaud and J. Meseguer, eds.), Lecture Notes in Computer Science, vol. 4060, Springer, 2006, p. 194–215. 8, 116

[AK07] Oana Andrei and Hélène Kirchner – « Graph Rewriting and Strategies for Modeling Biochemical Networks. », SYNASC '07 : Proceedings of the Ninth International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, IEEE Computer Society, 2007, p. 407–414. 8, 110

[AK08a] — , « A Biochemical Calculus Based on Strategic Graph Rewriting », Preproceedings of the 3rd International Conference on Algebraic Biology, 2008. 9, 110

[AK08b] — , « A Higher-Order Graph Calculus for Autonomic Computing », Graph Theory, Computational Intelligence and Thought. A Conference Celebrating Martin Charles Golumbic's 60th Birthday, 2008. 9, 104

[AK08c] — , « A Rewriting Calculus for Multigraphs with Ports. », Proceedings of RULE'07, Electronic Notes in Theoretical Computer Science, vol. 219, 2008, p. 67–82. 8

[AK08d] — , « Strategic Port Graph Rewriting for Autonomic Computing », TFIT, 2008. 9, 104

[AL94] Andrea Asperti and Cosimo Laneve – « Interaction Systems », HOA (J. Heering, K. Meinke, B. Möller and T. Nipkow, eds.), Lecture Notes in Computer Science, vol. 816, Springer, 1994, p. 1–19. 130

[Ale99] Vladimir Alexiev – « Non-deterministic Interaction Nets », Thesis, University of Alberta, 1999. 130

[Bar84] Henk Barendregt – The Lambda-Calculus, its syntax and semantics, second edition ed., Studies in Logic and the Foundation of Mathematics., Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. 4, 39

[BB92] Gérard Berry and Gérard Boudol – « The Chemical Abstract Machine. », Theoretical Computer Science 96 (1992), no. 1, p. 217–248. 3, 29

[BB07] Emilie Balland and Paul Brauner – « Term-graph rewriting in Tom using relative positions. », Pre-proceedings of 4th International Workshop on Computing with Terms and Graphs - TERMGRAPH, 2007. 130

[BBCK05] Clara Bertolissi, Paolo Baldan, Horatiu Cirstea and Claude Kirchner – « A Rewriting Calculus for Cyclic Higher-order Term Graphs. », Electronic Notes in Theoretical Computer Science 127 (2005), no. 5, p. 21–41. 32

[BBK+07a] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau and Antoine Reilles – « Tom : Piggybacking rewriting on java », Proceedings of the 18th Conference on Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 4533, Springer-Verlag, 2007, p. 36–47. 26

[BBK+07b] Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau and Antoine Reilles – « Tom : Piggybacking Rewriting on Java. », RTA'07 (F. Baader, ed.), Lecture Notes in Computer Science, vol. 4533, Springer, 2007, p. 36–47. 129

[BBK+07c] — , Tom Manual, LORIA, Nancy (France), version 2.5 ed., July 2007. 26

[BCC+03] Olivier Bournez, Guy-Marie Côme, Valérie Conraud, Hélène Kirchner and Liliana Ibanescu – « A Rule-Based Approach for Automated Generation of Kinetic Chemical Mechanisms. », Rewriting Techniques and Applications (RTA 2003) (R. Nieuwenhuis, ed.), Lecture Notes in Computer Science, vol. 2706, Springer, 2003, p. 30–45. 8, 113

[BDGM07] Lars Birkedal, Troels Christoffer Damgaard, Arne J. Glenstrup and Robin Milner – « Matching of Bigraphs », Electronic Notes in Theoretical Computer Science 175 (2007), no. 4, p. 3–19. 83

[BFR04] Jean-Pierre Banâtre, Pascal Fradet and Yann Radenac – « Higher-Order Chemical Programming Style », UPP (J.-P. Banâtre, P. Fradet, J.-L. Giavitto and O. Michel, eds.), Lecture Notes in Computer Science, vol. 3566, Springer, 2004, p. 84–95. 4, 29

[BFR06a] — , « A Generalized Higher-Order Chemical Computation Model », Electronic Notes in Theoretical Computer Science 135 (2006), no. 3, p. 3–13. 4, 30, 103

[BFR06b] — , « Generalised multisets for chemical programming », Mathematical Structures in Computer Science 16 (2006), no. 4, p. 557–580. 59

[BFR06c] — , « Towards chemical coordination for grids », SAC (H. Haddad, ed.), ACM, 2006, p. 445–446. 4, 30

[BFR07] — , « Programming Self-Organizing Systems with the Higher-Order Chemical Language », International Journal of Unconventional Computing 3 (2007), no. 3, p. 161–177. 4, 30, 103

[BG06] Olivier Bournez and Florent Garnier – « Proving Positive Almost Sure Termination Under Strategies », RTA (F. Pfenning, ed.), Lecture Notes in Computer Science, vol. 4098, Springer, 2006, p. 357–371. 57

[BIK06] Olivier Bournez, Liliana Ibanescu and Hélène Kirchner – « From Chemical Rules to Term Rewriting. », Electronic Notes in Theoretical Computer Science 147 (2006), no. 1, p. 113–134. 8, 113

[BK02] Olivier Bournez and Claude Kirchner – « Probabilistic Rewrite Strategies. Applications to ELAN », RTA (S. Tison, ed.), Lecture Notes in Computer Science, vol. 2378, Springer, 2002, p. 252–266. 57

[BKKR01a] Peter Borovanský, Claude Kirchner, Hélène Kirchner and Christophe Ringeissen – « Rewriting with Strategies in ELAN : A Functional Semantics », Int. J. Found. Comput. Sci. 12 (2001), no. 1, p. 69–95. 57

[BKKR01b] Peter Borovanský, Claude Kirchner, Helene Kirchner and Christophe Ringeissen – « Rewriting with strategies in ELAN : a functional semantics », International Journal of Foundations of Computer Science 12 (2001), no. 1, p. 69–98. 26

[BKN87] Dan Benanav, Deepak Kapur and Paliath Narendran – « Complexity of Matching Problems », J. Symb. Comput. 3 (1987), no. 1/2, p. 203–216. 22

[BM86] Jean-Pierre Banatre and Daniel Le Metayer – « A New Computational Model and Its Discipline of Programming. », Tech. Report RR-566, INRIA, 1986. 3, 29

[BM08] Emilie Balland and Pierre-Etienne Moreau – « Term-Graph Rewriting Via Explicit Paths », RTA (A. Voronkov, ed.), Lecture Notes in Computer Science, vol. 5117, Springer, 2008, p. 32–47. 116

[BMR07] Emilie Balland, Pierre-Etienne Moreau and Antoine Reilles – « Bytecode Rewriting in Tom », Electronic Notes in Theoretical Computer Science 190 (2007), no. 1, p. 19–33. 132

[BN98] Franz Baader and Tobias Nipkow – Term Rewriting and All That., Cambridge University Press, 1998. 15, 18, 21

[BRF04] Jean-Pierre Banâtre, Yann Radenac and Pascal Fradet – « Chemical Specification of Autonomic Systems », IASSE, ISCA, 2004, p. 72–79. 32, 103, 104

[BRJ+05] Grégory Batt, Delphine Ropers, Hidde de Jong, Johannes Geiselmann, Radu Mateescu, Michel Page and Dominique Schneider – « Validation of qualitative models of genetic regulatory networks by model checking : analysis of the nutritional stress response in scherichia coli », Bioinformatics 21 (2005), no. 1, p. 19–28. 132

[BS01] Franz Baader and Wayne Snyder – « Unification Theory », Handbook of Automated Reasoning (J. A. Robinson and A. Voronkov, eds.), Elsevier and MIT Press, 20001, p. 445–532. 20

[Bun79] Horst Bunke – « Programmed Graph Grammars », Graph-Grammars and Their Application to Computer Science and Biology (V. Claus, H. Ehrig and G. Rozenberg, eds.), Lecture Notes in Computer Science, vol. 73, Springer, 1979, p. 155–166. 78

[BYFH06] M. L. Blinov, J. Yang, J. R. Faeder and W. S. Hlavacek – « Graph Theory for Rule-Based Modeling of Biochemical Networks. », Transactions on Computational Systems Biology VII (C. Priami, A. Ingólfsdóttir, B. Mishra and H. R. Nielson, eds.), Lecture Notes in Computer Science, vol. 4230, Springer, 2006, p. 89–106. 8, 115

[Car05a] Luca Cardelli – « Abstract Machines of Systems Biology. », Transactions on Computational Systems Biology III (C. Priami, E. Merelli, P. P. Gonzalez and A. Omicini, eds.), Lecture Notes in Computer Science, vol. 3737, Springer, 2005, p. 145–168. 5

[Car05b] — , « Brane Calculi. », Computational Methods in Systems Biology, International Conference CMSB 2004, Paris, France, May 26-28, 2004, Revised Selected Papers (V. Danos and V. Schächter, eds.), Lecture Notes in Computer Science, vol. 3082, Springer, 2005, p. 257–278. 103, 115

[CDE+02] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer and Jose F. Quesada – « Maude : specification and programming in rewriting logic. », Theoretical Computer Science 285 (2002), no. 2, p. 187–243. 116

[CF07] Horatiu Cirstea and Germain Faure – « Confluence of Pattern-Based Calculi », RTA (F. Baader, ed.), Lecture Notes in Computer Science, vol. 4533, Springer, 2007, p. 78–92. 31

[CG00] Luca Cardelli and Andrew D. Gordon – « Mobile ambients. », Theoretical Computer Science 240 (2000), no. 1, p. 177–213. 103

[CGP00] Edmund M. Clarke, Orna Grumberg and Doron A. Peled – Model Checking, MIT Press, 2000. 10, 56, 131, 133, 135, 137

[CK98] Horatiu Cirstea and Claude Kirchner – « The Rewriting Calculus as a Semantics of ELAN », ASIAN (J. Hsiang and A. Ohori, eds.), Lecture Notes in Computer Science, vol. 1538, Springer, 1998, p. 84–85. 31

[CK01] — , « The Rewriting Calculus - Part I and II », Logic Journal of the IGPL 9 (2001), no. 3, p. 427—498. 7, 30, 31, 32, 40, 48, 52, 59

[CKL02] Horatiu Cirstea, Claude Kirchner and Luigi Liquori – « Rewriting Calculus with(out) Types », Electronic Notes in Theoretical Computer Science 71 (2002), p. 3–19. 31

[CKLW03] Horatiu Cirstea, Claude Kirchner, Luigi Liquori and Benjamin Wack – « Rewrite strategies in the rewriting calculus. », Electronic Notes in Theoretical Computer Science 86 (2003), no. 4, p. 593–624. 31, 47

[CMR+97] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel and Michael Löwe – « Algebraic Approaches to Graph Transformation - Part I : Basic Concepts and Double Pushout Approach. », Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1 : Foundations (G. Rozenberg, ed.), World Scientific, 1997, p. 163–246. 24

[CRCFS04] Nathalie Chabrier-Rivier, Marc Chiaverini, Vincent Danos François Fages and Vincent Schächter – « Modeling and querying biomolecular interaction networks. », Theoretical Computer Science 325 (2004), no. 1, p. 25–44. 10, 132

[CZ08] Luca Cardelli and Gianluigi Zavattaro – « On the Computational Power of Biochemistry », AB'08 (K. Horimoto, G. Regensburger, M. Rosenkranz and H. Yoshida, eds.), Lecture Notes in Computer Science, vol. 5147, Springer-Verlag, 2008, p. 65–80. 5, 32

[Dau92] Max Dauchet – « Simulation of Turing Machines by a Regular Rewrite Rule », Theoretical Computer Science 103 (1992), no. 2, p. 409–420. 20

[Der82] Nachum Dershowitz – « Orderings for Term-Rewriting Systems », Theoretical Computer Science 17 (1982), p. 279–301. 21

[DL04] Vincent Danos and Cosimo Laneve – « Formal Molecular Biology. », Theoretical Computer Science 325 (2004), no. 1, p. 69–110. 8, 111, 115

[DP04] Vincent Danos and Sylvain Pradalier – « Projective Brane Calculus. », CMSB (V. Danos and V. Schächter, eds.), Lecture Notes in Computer Science, vol. 3082, Springer, 2004, p. 134–148. 103

[DZB01] Peter Dittrich, Jens Ziegler and Wolfgang Banzhaf – « Artificial Chemistries - A Review. », Artificial Life 7 (2001), no. 3, p. 225–275. 4

[EEKR97] H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg (eds.) – Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2 : Applications, Languages, and Tools, World Scientific, 1997. 23, 25, 65

[EHK+97] Hartmut Ehrig, Reiko Heckel, Martin Korff, Michael Löwe, Leila Ribeiro, Annika Wagner and Andrea Corradini – « Algebraic Approaches to Graph Transformation - Part II : Single Pushout Approach and Comparison with Double Pushout Approach », Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1 : Foundations (G. Rozenberg, ed.), World Scientific, 1997, p. 247–312. 24

[EHKPP90] Hartmut Ehrig, Annegret Habel, Hans-Jörg Kreowski and Francesco Parisi-Presicce – « From Graph Grammars to High Level Replacement Systems », Graph-Grammars and Their Application to Computer Science (H. Ehrig, H.-J. Kreowski and G. Rozenberg, eds.), Lecture Notes in Computer Science, vol. 532, Springer, 1990, p. 269–291. 84

[Ehr79] Hartmut Ehrig – « Introduction to the Algebraic Theory of Graph Grammars (A Survey). », Graph-Grammars and Their Application to Computer Science and Biology (V. Claus, H. Ehrig and G. Rozenberg, eds.), Lecture Notes in Computer Science, vol. 73, Springer, 1979, p. 1–69. 81

[EKL+04] Steven Eker, Merrill Knapp, Keith Laderoute, Patrick Lincoln and Carolyn L. Talcott – « Pathway Logic : Executable Models of Biological Networks. », Electronic Notes in Theoretical Computer Science 71 (2004), p. 144—161. 116

[EKMR97] H. Ehrig, H.-J. Kreowski, U. Montanari and G. Rozenberg (eds.) – Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3 : Concurrency, Parallelism, and Distribution, World Scientific, 1997. 23, 65

[EMS04] Steven Eker, Jose Meseguer and Ambarish Sridharanarayanan – « The Maude LTL Model Checker », Electronic Notes in Theoretical Computer Science 71 (2004), p. 162–187. 136

[EP05] Hartmut Ehrig and Ulrike Prange – « Modeling with Graph Transformation », Advances in Multiagent Systems, Robotics and Cybernetics : Theory and Practice. Proceedings of Intern. Conf. on Systems Research, Informatics and Cybernetics (G. Lalsker and J. Pfalzgraf, eds.), 2005. 65

[EPPH06] Hartmut Ehrig, Julia Padberg, Ulrike Prange and Annegret Habel – « Adhesive High-Level Replacement Systems : A New Categorical Framework for Graph Transformation », Fundam. Inform. 74 (2006), no. 1, p. 1–29. 85

[ERT97] Claudia Ermel, Michael Rudolf and Gabriele Taentzer – « The AGG Approach : Language and Environment », Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2 : Applications, Languages, and Tools (H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg, eds.), World Scientific, 1997, p. 551–603. 25, 74

[FB96] Walter Fontana and Leo W. Buss – « The barrier of objects : From dynamical systems to bounded organizations. », Boundaries and Barriers. (J. Casti and A. Karlqvist, eds.), Addison-Wesley, 1996, p. 56–116. 4, 43

[FBH05] James R. Faeder, Michael L. Blinov and William S. Hlavacek – « Graphical rule-based representation of signal-transduction networks. », Proceedings of the 2005 ACM Symposium on Applied Computing (SAC), Santa Fe, New Mexico, USA (H. Haddad, L. M. Liebrock, A. Omicini and R. L. Wainwright, eds.), ACM, 2005, p. 133–140. 117

[FGK03] Olivier Fissore, Isabelle Gnaedig and Hélène Kirchner – « Simplification and termination of strategies in rule-based languages. », PPDP, ACM, 2003, p. 124–135. 117

[FM98] Pascal Fradet and Daniel Le Métayer – « Structured Gamma. », Sci. Comput. Program. 31 (1998), no. 2-3, p. 263–289. 3

[FMP07] Maribel Fernández, Ian Mackie and Jorge Sousa Pinto – « A Higher-Order Calculus for Graph Transformation », Electronic Notes in Theoretical Computer Science 72 (2007), no. 1, p. 45–58. 129, 130

[FMS06] Maribel Fernández, Ian Mackie and François-Régis Sinot – « Interaction Nets vs. the rho-calculus : Introducing Bigraphical Nets. », Electronic Notes in Theoretical Computer Science 154 (2006), no. 3, p. 19–32. 93, 94

[Gia03] Jean-Louis Giavitto – « Invited Talk : Topological Collections, Transformations and Their Application to the Modeling and the Simulation of Dynamical Systems », RTA (R. Nieuwenhuis, ed.), Lecture Notes in Computer Science, vol. 2706, Springer, 2003, p. 208–233. 4, 61

[Gil77] Daniel T. Gillespie – « Exact stochastic simulation of coupled chemical reactions. », J. Phys. Chem. 81 (1977), no. 25, p. 2340–2361. 57

[GM92] Joseph A. Goguen and Jose Meseguer – « Order-Sorted Algebra I : Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations. », Theoretical Computer Science 105 (1992), no. 2, p. 217–273. 18

[GM01] Jean-Louis Giavitto and Olivier Michel – « MGS : a Rule-Based Programming Language for Complex Objects and Collections. », Electronic Notes in Theoretical Computer Science 59 (2001), no. 4, p. 286–304. 4

[GM02a] — , « Data Structure as Topological Spaces », UMC (C. Calude, M. J. Dinneen and F. Peper, eds.), Lecture Notes in Computer Science, vol. 2509, Springer, 2002, p. 137–150. 61

[GM02b] — , « The Topological Structures of Membrane Computing », Fundam. Inform. 49 (2002), no. 1–3, p. 123–145. 61

[HFB+06] William S. Hlavacek, James R. Faeder, Michael L. Blinov, Richard G. Posner, Michael Hucka and Walter Fontana – « Rules for Modeling Signal-Transduction Systems. », Science STKE 334 (2006). 115

[HHT96] Annegret Habel, Reiko Heckel and Gabriele Taentzer – « Graph Grammars with Negative Application Conditions », Fundam. Inform. 26 (1996), no. 3/4, p. 287–313. 104

[HP01] Annegret Habel and Detlef Plump – « Computational Completeness of Programming Languages Based on Graph Transformation », FoSSaCS (Furio Honsell and Marino Miculan, ed.), Lecture Notes in Computer Science, vol. 2030, Springer, 2001, p. 230–245. 78

[Iba04] Liliana Ibanescu – « Programmation par règles et stratégies pour la génération automatique de mécanismes de combustion d'hydrocarbures polycycliques », Thesis, Institut National Polytechnique de Lorraine, 2004. 8, 113

[IYD05] Oscar H. Ibarra, Hsu-Chun Yen and Zhe Dang – « On various notions of parallelism in P Systems », Int. J. Found. Comput. Sci. 16 (2005), no. 4, p. 683–705. 57

[JK86] Jean-Pierre Jouannaud and Hélène Kirchner – « Completion of a set of rules modulo a set of equations. », SIAM Journal of Computing 15 (1986), no. 4, p. 1155 – 1194. 22

[JM04] Ole Hogh Jensen and Robin Milner – « Bigraphs and mobile processes (revised) », Tech. Report 580, University of Cambridge, February 2004. 83

[KC03] Jeffrey O. Kephart and David M. Chess – « The Vision of Autonomic Computing », IEEE Computer 36 (2003), no. 1, p. 41–50. 103

[KKK08] Claude Kirchner, Florent Kirchner and Hélène Kirchner – « Strategic Computations and Deductions », Festchrift in honor of Peter Andrews, Studies in Logic and the Foundation of Mathematics, Elsevier, 2008. 17, 25, 26

[KKV95] Claude Kirchner, Hélène Kirchner and Marian Vittek – « Designing Constraint Logic Programming Languages using Computational Systems. », Principles and Practice of Constraint Programming. The Newport Papers. (P. Van Hentenryck and V. Saraswat, eds.), MIT Press, 1995, p. 131–158. 26

[Klo80] Jan Willem Klop – « Combinatory Reduction Systems », Thesis, CWI, Amsterdam, 1980. 129

[KM01] Hélène Kirchner and Pierre-Etienne Moreau – « Promoting rewriting to a programming language : a compiler for non-deterministic rewrite programs in associative-commutative theories. », J. Funct. Program. 11 (2001), no. 2, p. 207–251. 122

[KMT08] Jean Krivine, Robin Milner and Angelo Troina – « Stochastic Bigraphs », The 24th Conference on the Mathematical Foundations of Programming Semantics, 2008. 84, 99

[Laf90] Yves Lafont – « Interaction Nets », POPL, 1990, p. 95–108. 130

[Lin68] Aristid Lindenmayer – « Mathematical models for cellular interaction in development », Journal of Theoretical Biology I and II (1968), no. 18, p. 280–315. 3

[LT07] Cosimo Laneve and Fabien Tarissan – « A simple calculus for proteins and cells. », Electronic Notes in Theoretical Computer Science 171 (2007), no. 2, p. 139–154. 103, 111, 115

[LV02] Javier Larrosa and Gabriel Valiente – « Constraint Satisfaction Algorithms for Graph Pattern Matching », Mathematical Structures in Computer Science 12 (2002), no. 4, p. 403–422. 25, 85

[Mac98] Saunders Mac Lane – Categories for the Working Mathematician, 2nd ed., Graduate Texts in Mathematics, Springer, 1998. 22, 80

[Meh84] Kurt Mehlhorn – Data Structures and Algorithms 2 : Graph Algorithms and NP-Completeness, Monographs in Theoretical Computer Science. An EATCS Series, vol. 2, Springer, 1984. 24

[Mes92] José Meseguer – « Conditioned Rewriting Logic as a United Model of Concurrency. », Theoretical Computer Science 96 (1992), no. 1, p. 73–155. 57

[Mes96] — , « Rewriting Logic as a Semantic Framework for Concurrency : a Progress Report », CONCUR (U. Montanari and V. Sassone, eds.), Lecture Notes in Computer Science, vol. 1119, Springer, 1996, p. 331–372. 119, 121, 129

[Mil99] Robin Milner – Communicating and Mobile Systems : the Pi-Calculus, Cambridge University Press, 1999. 8, 115

[Mil01] — , « Bigraphical Reactive Systems. », CONCUR (K. G. Larsen and M. Nielsen, eds.), Lecture Notes in Computer Science, vol. 2154, Springer, 2001, p. 16–35. 83

[Mil06] — , « Pure bigraphs : Structure and dynamics. », Inf. Comput. 204 (2006), no. 1, p. 60–122. 83, 103

[MOMV05] Narciso Martí-Oliet, José Meseguer and Alberto Verdejo – « Towards a strategy language for Maude », Proceedings Fifth International Workshop on Rewriting Logic and its Applications, WRLA 2004, Barcelona, Spain, March 27 – April 4, 2004 (N. Martí-Oliet, ed.), Electronic Notes in Theoretical Computer Science, vol. 117, Elsevier, 2005, p. 417–441. 26

[MP43] W. S. McCullough and W. Pitts – « A logical calculus for the ideas immanent in nervous activity », Bulletin of Mathemathical Biophysics 5 (1943), p. 115–133. 3

[MRM+08] Pedro T. Monteiro, Delphine Ropers, Radu Mateescu, Ana T. Freitas and Hidde de Jong – « Temporal logic patterns for querying dynamic models of cellular interaction networks », Bioinformatics 24 (2008), no. 16, p. 227–233. 10, 132

[Neu66] John Von Neumann – Theory of Self-Reproducing Automata, University of Illinois Press, Champaign, Illinois, 1966. 3

[New42] M.H.A. Newman – « On theories with combinatorial definition of "equivalence" », Annals of Mathematics 43 (1942), no. 2, p. 223–243. 16

[Pau02] Gheorghe Paun – Membrane Computing. An Introduction, Springer, 2002. 4, 57

[Pau06] — , « Twenty Six Research Topics About Spiking Neural P Systems. », Available at the P Systems Web Page : http ://psystems.disco.unimib.it., 2006. 117

[PC03] Sabine Peres and Jean-Paul Comet – « Contribution of Computational Tree Logic to Biological Regulatory Networks : Example from Pseudomonas Aeruginosa », CMSB (C. Priami, ed.), Lecture Notes in Computer Science, vol. 2602, Springer, 2003, p. 47–56. 132

[PE93] Marinus J. Plasmeijer and Marko C. J. D. van Eekelen – Functional Programming and Parallel Graph Rewriting, Addison-Wesley, 1993. 25

[Plu99] Detlef Plump – « Term Graph Rewriting », Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2 : Applications, Languages, and Tools (H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg, eds.), World Scientific, 1999, p. 3–61. 130

[Plu05] — , « Confluence of Graph Transformation Revisited », Processes, Terms and Cycles : Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday (A. Middeldorp, V. van Oostrom, F. van Raamsdonk and R. C. de Vrijer, eds.), Lecture Notes in Computer Science, vol. 3838, Springer, 2005, p. 280–308. 80, 83

[PRC08] Gheorghe Paun and Francisco José Romero-Campero – « Membrane Computing as a Modeling Framework. Cellular Systems Case Studies », SFM (M. Bernardo, P. Degano and G. Zavattaro, eds.), Lecture Notes in Computer Science, vol. 5016, Springer, 2008, p. 168–214. 4

[PS81] Gerald E. Peterson and Mark E. Stickel – « Complete Sets of Reductions for Some Equational Theories. », J. ACM 28 (1981), no. 2, p. 233–264. 22

[QLF+06] Zhengwei Qi, Minglu Li, Cheng Fu, Dongyu Shi and Jinyuan You – « Membrane Calculus : a formal method for Grid transactions », Concurrency and Computation : Practice and Experience 18 (2006), no. 14, p. 1799–1809. 103

[Rad07] Yann Radenac – « Programmation "chimique" d'ordre supérieur », Thèse de doctorat, Université de Rennes 1, April 2007. 4, 9, 29, 59, 100

[Roz97] G. Rozenberg (ed.) – Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1 : Foundations, World Scientific, 1997. 23, 65

[RPS+04] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli and Ehud Y. Shapiro – « BioAmbients : an abstraction for biological compartments. », Theoretical Computer Science 325 (2004), no. 1, p. 141–167. 103

[Sch97] Andy Schürr – « Programmed Graph Replacement Systems. », Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1 : Foundations (G. Rozenberg, ed.), World Scientific, 1997, p. 479–546. 24, 78

[SM05] Antoine Spicher and Olivier Michel – « Using Rewriting Techniques in the Simulation of Dynamical Systems : Application to the Modeling of Sperm Crawling », International Conference on Computational Science (1) (V. S. Sunderam, G. D. van Albada, P. M. A. Sloot and J. Dongarra, eds.), Lecture Notes in Computer Science, vol. 3514, Springer, 2005, p. 820–827. 61

[SMC+08] Antoine Spicher, Olivier Michel, Mikolaj Cieslak, Jean-Louis Giavitto and Przemyslaw Prusinkiewicz – « Stochastic P systems and the simulation of biochemical processes with dynamic compartments », Biosystems 91 (2008), no. 3, p. 458–472. 57

[Spi06] Antoine Spicher – « Transformation de collections topologiques de dimension arbitraire. Application à la modélisation de systèmes dynamiques », Thesis, Université d'Évry, 2006. 4, 57, 61

[ST99] Ron Shamira and Dekel Tsura – « Faster Subtree Isomorphism », Journal of Algorithms 33 (1999), no. 2, p. 267–280. 85

[SWZ97] Andy Schürr, Andreas J. Winter and Albert Zündorf – « The PROGRES Approach : Language and Environment. », Handbook of Graph Grammars and Computing by Graph Transformations, Volume 2 : Applications, Languages, and Tools (H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg, eds.), World Scientific, 1997, p. 479–546. 25, 74, 78

[Ull76] J. R. Ullman – « An Algorithm for Subgraph Isomorphism », Journal of the ACM 23 (1976), no. 1, p. 31–42. 25, 85

[Val02] Gabriel Valiente – Algorithms on Trees and Graphs, Springer, 2002. 25, 85

[Vis01] Eelco Visser – « Stratego : A Language for Program Transformation based on Rewriting Strategies. System Description of Stratego 0.5. », Rewriting

Techniques and Applications (RTA'01) (A. Middeldorp, ed.), Lecture Notes in Computer Science, vol. 2051, Springer-Verlag, May 2001, p. 357–361. 26

[Zün96] Albert Zündorf – « Graph Pattern Matching in PROGRES », TAGT (J. E. Cuny, H. Ehrig, G. Engels and G. Rozenberg, eds.), Lecture Notes in Computer Science, vol. 1073, Springer, 1996, p. 454–468. 25, 85

Table des figures

0.1 Deux graphes moléculaires reliés pas une réaction de complexation 9

3.1 A decomposition of an abstraction on graphs 39
3.2 Représentation sous forme de boîte d'un monde consistant en les abstractions A_1, A_2, et A_3, et les objets structurés O_1 et O_2. 41
3.3 Syntaxe de base de $\rho_{\langle\Sigma\rangle}$-calcul 42

4.1 Sémantique de base de $\rho_{\langle\Sigma\rangle}$-calcul 43
4.2 Sémantique de $\rho_{\langle\Sigma\rangle}$-calcul avec application explicite 46

5.1 Illustration de la nécessité d'une évaluation de type appel-par-nom dans le calcul 50
5.2 Syntaxe du $\rho_{\langle\Sigma\rangle}$-calcul avec stratégies 53
5.3 Sémantique du $\rho_{\langle\Sigma\rangle}$-calcul avec stratégies 53

10.1 Deux vues d'un graphe avec ports. 67
10.2 Un exemple de graphe avec ports 71
10.3 Règles de réécriture de graphe avec ports 76
10.4 Étapes de réécriture 76
10.5 Une application de la règle de réécriture de graphe avec ports de la Figure 10.3 (a) sur un graphe avec ports G résultant en le graphe avec ports G' 78
10.6 Une application de la règle de réécriture de graphe avec ports donnée en Figure 10.3 (a) sur G résultant en deux graphes avec ports G' et G'' avec un graphe avec ports intermédiaire I 80

11.1 Une abstraction avec câblages explicites qui spécifient le découpage d'un nœud identifié par le couple de variables $i : X$ connectée à un nœud $j : Y$ et introduit une règle découpant tout nœud ayant le même nom que i . . . 90

12.1 A mail system configuration 105
12.2 Basic rules for the mail delivery system 105
12.3 Rules for self-configuration and self-healing in the mail delivery system . . 107
12.4 Rules for self-protection in the mail delivery system 108
12.5 Rules for self-optimization in the mail delivery system 109
12.6 Initial and intermediate molecular graphs in the EGFR model 112
12.7 The reaction patterns in the EGFR signaling pathway fragment 113

12.8 G" and G"' are obtained from G' by rewriting using the rule r2 on different subgraphs 113
12.9 The molecular graph H for the EGFR signaling pathway fragment in the equilibrium state 115

13.1 The operation set $\mathcal{F}$ 121

14.1 Port graph expressions 134
14.2 Structural formulas for port graphs 134
14.3 Formulas in CTL for port graphs 137

A.1 Test 173

Résumé

L'objectif de cette thèse est d'explorer des descriptions formelles pour la structure et le fonctionnement des systèmes biologiques, ainsi que des outils formels pour raisonner au sujet de leur comportement. Cette thèse s'inscrit dans les travaux étudiant les modèles informatiques sûrs où les calculs sont exprimés par l'intermédiaire de la réécriture, et où nous pouvons compter sur la vérification formelle pour exprimer et valider les propriétés des modèles.

Dans cette thèse nous développons un calcul de réécriture d'ordre supérieur pour décrire des molécules, des règles de réaction, et la génération des réseaux biochimiques. Le calcul est basé sur la métaphore chimique en décrivant les calculs en termes de solutions chimiques dans lesquelles les molécules représentant des données agissent l'une sur l'autre librement selon des règles de réaction. Ainsi nous avons obtenu un Calcul Biochimique Abstrait étendant le modèle chimique d'ordre supérieur en considérant des molécules structurées. Le calcul est équipé d'une spécification naturelle de la concurrence et des mécanismes de contrôle grâce à l'expression des stratégies de réécriture sous forme de molécules.

La description des complexes moléculaires ou des réactifs chimiques appartient à une classe spécifique de graphes. Nous définissons la structure des graphes avec ports et nous montrons que les principes du calcul biochimique instanciés pour les graphes avec ports sont assez expressifs pour modéliser des systèmes autonomes et des réseaux biochimiques. En plus, les techniques de la réécriture stratégique ouvrent la voie au raisonnement basé sur les calculs et à la vérification des propriétés des systèmes modélisés.

Mots clefs : calcul de réécriture, réécriture de graphes, stratégies de réécriture, réseaux biochimiques, systèmes autonomes

Abstract

The objective of this thesis is to explore formal descriptions for the structure and functioning of biological systems, as well as formal tools for reasoning about their behavior. This work takes place in the overall prospective to study safe computational models where computations are expressed via rewriting, and where we can rely on formal verification to express and validate suitable properties.

In this thesis we develop a higher-order calculus rewriting for describing molecules, reaction patterns, and biochemical network generation. The calculus is based on the chemical metaphor by describing the computations in terms of chemical solutions in which molecules representing data freely interact according to reaction rules. This way we obtained an Abstract Biochemical Calculus as an extension of the higher-order chemical model by considering structured molecules. The calculus is provided with a natural specification of concurrency and of controlling mechanisms by expressing rewrite strategies as molecules.

The description of molecular complexes or chemical reactants belong to specific classes of graphs. We define the structure of port graphs and we show how the principles of the biochemical calculus instantiated for port graphs are expressive enough for modeling autonomous systems and biochemical networks. In addition, strategic rewriting techniques open the way to reason about the computations and to verify properties of the modeled systems.

Keywords: rewriting calculus, graph rewriting, strategic rewriting, biochemical networks, autonomous systems

INSTITUT NATIONAL
POLYTECHNIQUE
DE LORRAINE

AUTORISATION DE SOUTENANCE DE THESE
DU DOCTORAT DE L'INSTITUT NATIONAL
POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :
Monsieur Jean-Pierre BANATRE, Professeur, Université de Rennes 1, IRISA, Rennes

Monsieur Jean-Louis GIAVITTO, Directeur de Recherche, IBISC, Université d'Evry Val d'Essonne, Evry

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Madame ANDREI Oana-Maria

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 54501
VANDŒUVRE CEDEX

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

"Un calcul de réécriture de graphes : applications à la biologie et aux systèmes autonomes"

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : **« Informatique »**

Fait à Vandoeuvre, le 22 octobre 2008
Le Président de l'I.N.P.L.,
F. LAURENT

TEL. 33/03.83.59.59.59
FAX. 33/03.83.59.59.55

Fig. A.1: Test