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

Oana Andrei

Nancy-Université
INPL

Département de formation doctorale en informatique

Institut National Polytechnique de Lorraine

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

Contents

Introduction

In this thesis we develop a higher-order calculus based on port graph rewriting for describing molecules, reaction patterns, and biochemical network generation. This calculus is an extension of the chemical model by considering structured objects. Then we obtain a natural specification of concurrency and of controlling mechanisms by expressing rewrite strategies as objects of the calculus. We introduce the structure of port graphs and we show how the principles of the biochemical calculus instantiated for port graphs are expressive enough for modeling systems with self-organizing and emergent properties. In addition, strategic rewriting techniques open the way to reason about the computations and to verify properties of the modeled systems.

Motivation: from Chemical to Biochemical Computations

Since the early ages of computer science researchers were interested in nature-inspired computational models which led, for instance, to neural networks [MP43], cellular automata [Neu66], and Lindenmayer systems [Lin68]. By the time the development in the theoretical computer science accelerated, the simplicity of the basic principles of chemistry inspired researchers to abstract a computational paradigm for programming, the *chemical programming model* or the *chemical metaphor*, in terms of molecules, solutions of molecules and reactions. In the following we review this computational paradigm, and afterwards we present a way of moving to a biological dimension of the model by considering structured molecules. The result is an abstract biochemical calculus which can be instantiated for various structures and extended with verification features.

The Chemical Metaphor

The chemical computation metaphor emerged as a computation paradigm over the last three decades. This metaphor describes computation in terms of a chemical solution in which molecules representing data freely interact according to reaction rules. Chemical solutions are represented by multisets and the computation proceeds by rewritings, which consume and produce new elements according to some rules. Several reactions occur in parallel if they do not compete for the same data. Hence multisets represent the fundamental structure of the chemical computation models. The chemical computational model was proposed by [BM86] using the Gamma formalism. The goal of this work was to capture the intuition of computation as a global evolution of a collection of atomic values interacting freely. The generality of the rules ensures a great expressive power and, in a direct manner, computational universality. More generally, the structured

multisets defined in [FM98] can be seen as a syntactic facility allowing the organization of explicit data, and providing a notation leading to higher-level programs manipulating more complex data structures.

The CHemical Abstract Machine (CHAM) formalism [BB92] extends the Gamma formalism by introducing the notion of sub-solution enclosed in a membrane, together with a classification of the rules as heating rules (for rearranging a solution such that reaction can take place), cooling rules (for removing useless molecules after a reaction took place), or ordinary reaction rules. This formalism was designed as a model of concurrency and as a specific style for defining the operational semantics of concurrent systems.

In AlChemy [FB96], the molecules are normalized λ-terms [Bar84] and a reaction between two molecules corresponds to a β-reduction. The underlying motivation of this system was to develop a formal understanding of self-maintaining organizations inspired by biological systems.

The γ-calculus [BFR04, Rad07] was designed as a basic higher-order calculus developed on the essential features of the chemical paradigm. It generalizes the chemical model by considering the reactions as molecules as well. The Higher-Order Chemical Language (HOCL) [BFR06c, BFR06a, BFR07] extends the γ-calculus with programming elements. These formalisms were proved to be well-suited for modeling autonomous systems and for grid programming.

Membrane systems or P systems [Pau02] are another example of chemical model. They represent an abstract model of parallel and distributed computing inspired by cell compartments and molecular membranes. A cell is divided into various compartments, each compartment with a different task, with all of them working simultaneously to accomplish a more general task for the whole system. The membranes of a P system determine regions where multisets of objects and evolution rules can be placed. The objects evolve according to the rules associated with each region, and the regions cooperate in order to maintain the proper behavior of the whole system. P systems provide a nice abstraction for parallel systems, and a suitable framework for distributed and parallel algorithms. Membrane computing is directly inspired by the cell biology and uses new and useful ideas: localization, hierarchical structures, distribution, and communication. P systems provide an elegant and powerful computation model, able to solve computationally hard problems in a feasible time and useful to model various biological phenomena [PRC08].

MGS is another formalism based on the chemical model [GM01, Gia03, Spi06]. It was designed to represent and manipulate local transformations of entities structured by abstract topologies [GM01]. A set of entities organized by an abstract topology is called a topological collection. The collection types range in MGS from sets and multisets to more structured types. MGS has the ability to nest topologies in order to describe biological systems. Using transformation on multisets, MGS is a formalism unifying biologically inspired computational models like Gamma, P systems, or Lindenmayer systems.

Multiset rewriting lies at the core of these formalisms. It is a special case of rewriting where the function symbols are both associative and commutative. Several frameworks provide efficient environments for applying multiset rewriting rules, possibly following

some evaluation strategies. All the formalisms mentioned above are particular artificial chemistry instances based on the rewriting mechanism. An artificial chemistry is "a man-made system which is similar to a real chemical system" [DZB01]. Formally, an artificial chemistry is defined by a set of all possible molecules, a set of collision (or reaction) rules representing interactions among the molecules, and an algorithm describing how rules are applied on a fixed set of molecules.

Towards a Biochemical Calculus

A natural extension of the chemical metaphor is to add a biological flavor by providing the molecules with a particular structure and with association (complexation) and dissociation (decomplexation) capabilities. In living cells, molecules like nucleic acids, proteins, lipids, carbohydrates can combine based on their structural properties to form more complex entities. Biochemistry as a science focuses heavily on the role, function, and structure of such molecules. In a computer representation, the data structures that best describe these molecules range from lists through trees and graphs to more complex containers [Car05a].

Moving from chemistry to biochemistry by using an adequate structure for molecules capable of expressing connections between them was shown in [CZ08] to be very computationally interesting. It was proved that by adding basic association and dissociation capabilities of entities (for complexation and decomplexation respectively) to a minimal process algebra-based formalism for modeling chemistry increases the computational power such that a Turing complete computational model is obtained. This result encouraged us to believe that adding association and dissociation capabilities for molecules represents an essential feature for passing from a minimal chemical model to a biochemical one. In addition, it justifies our aim of defining a biochemical calculus by extending the minimal chemical model proposed by the γ-calculus with a structure for molecules that permits the expression of connections between molecules and operations concerning such connections.

A Biochemical Calculus based on Port Graph Rewriting

An Abstract Biochemical Calculus

The passage from a chemical model to a biochemical one and the gain in expressivity it may provide motivated us in the work presented in this thesis. We propose a calculus which extends the γ-calculus through a more powerful abstraction capability that considers for matching not a sole variable but a whole structured molecule. We assume that the structure considered for molecules, in general denoted by Σ, also permits them to connect. This approach is similar to the definition of the ρ-calculus [CK01] as an extension of the λ-calculus and first-order term rewriting. The result is a rewriting calculus with higher-order capabilities based on the chemical metaphor with structured molecules having connective capabilities and reaction rules over such molecules; we called it the $\rho_{\langle\Sigma\rangle}$-calculus. Based on the connectivity features of the Σ-structured molecules, we con-

sider the $\rho_{\langle\Sigma\rangle}$-calculus to be a biochemical extension of the γ-calculus, hence the name Abstract Biochemical Calculus.

The first-class citizens of the $\rho_{\langle\Sigma\rangle}$-calculus are *structured objects* as molecules, *abstractions* as rewrite rules over molecules or other abstractions, and *abstraction applications*. The structured objects and the abstractions are defined at the same level as molecules. Following the same principles as in the chemical model, a juxtaposition of molecules in a multiset represents also a molecule. We abstract the environment where molecules are floating using an operator that groups them in a *world*. An interaction between an abstraction and a molecule may take place in multiple ways due to all possible matching solutions between the abstraction and the molecule. As a consequence a world can have several evolution possibilities and we collect them all in a structure of alternative worlds called *multiverse*.

The high expressive power of the $\rho_{\langle\Sigma\rangle}$-calculus allows us to model some control on composing or choosing the application order of rules based on the notions of strategy and strategic rewriting. We encode strategies as particular abstractions and include them in the calculus at the same level as the other molecules. In addition, strategies permit us to exploit failure information.

Port Graphs as Structures for Biological Molecules

In [AIK06] we explored graph models for simulating a chemical reactor in TOM based on the work on the GasEl project [BCC+03, BIK06, Iba04]. This project was developed using rule-based systems and strategies for the problem of automated generation of kinetics mechanisms following the artificial chemistry approach. Both for a chemical reactor in [AIK06] and for modeling protein interactions in [AK07b], molecules are represented as graphs where the nodes correspond to atoms and to proteins respectively, and the reactions rules create or break bonds between the nodes. On the basis of these works, we highlight a graph structure where the nodes have points, called *ports*, for attaching the edges, thus providing an explicit partitioning of nodes connectivity. In this thesis we identify a general class of directed graphs allowing multiple edges and loops, where a node label is a triple of node identifier, node name and set of ports, while an edge label is the ordered pair of source port and target port. We call such graphs *port graphs* (or multigraphs with ports) and we define a suitable (strategic) rewriting relation on them [AK07a]. We also provide an axiomatization of port graphs and port graph rewriting using a suitable first-order term algebra and a corresponding term rewriting relation.

The concept of port for graphs is not a novelty. It can be seen as a refinement of the connectivity information for nodes. In particular, an inspiring starting point for our work on port graphs was the graphical formalism 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. In the same vein, another inspiring formalism for us was the κ-calculus [DL04]; this is a language of formal proteins which models complexes as graphs-with-sites and their interactions as a particular graph-rewriting operation. It uses an algebraic notation in the style of the

π-calculus [Mil99] and bonds are represented in molecular complexes by shared names.

Proteins are abstracted as boxes with interaction sites on the surface having particular states. Hence by adding a refinement on the ports and calling them *sites* with at most one edge attached to each port, port graph rewriting becomes suitable for modeling the interactions of molecular complexes. Each site has a state indicating the connection availability. We call this variation of port graphs used for modeling molecular complexes *molecular graphs* [AK07b]. In Figure 0.1 we illustrate in the middle a reaction pattern that applied on the left molecular graph creates an edge (called bond in the biochemical framework) as we can see in the molecular graph on the right. This example is extracted from a larger example developed in Section 6.2.1 which models a fragment of the epidermal growth factor receptor (EGFR) signaling pathway. The protagonists of the example are four signal proteins denoted by **S** with **S.S** their dimerized form, two receptor proteins **R** and one adapter protein **A**. Sites are represented differently according to their state: filled circles for bound sites and empty circles for free ones.

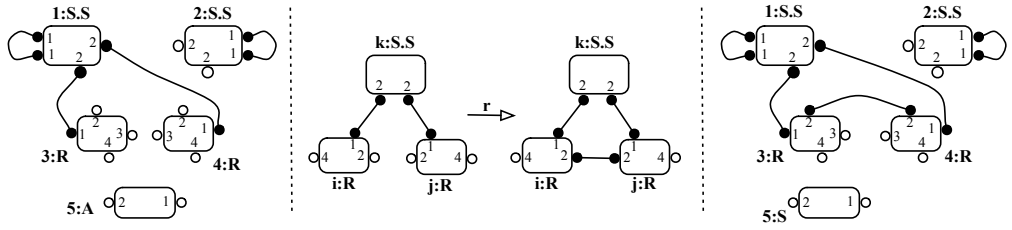


Figure 0.1: Two molecular graphs related by a complexation reaction

As already seen in the example above, modeling molecules by using the structure of port graphs endows them with connection capabilities. This motivates us in instantiating the abstract structure Σ in the $\rho_{\langle\Sigma\rangle}$-calculus with port graphs. In consequence, we obtain a biochemical calculus based on strategic port graph rewriting, the *ρ_{pg}-calculus*. Port graphs represent a unifying structure for representing all kinds of abstract molecules in the ρ_{pg}-calculus. In addition, the operations behind the application mechanism, matching and replacement, usually defined at the metalevel of a rewriting calculus, are expressible using appropriate nodes and port graph transformations. By restricting the port graphs to molecular graphs, we obtain a calculus for modeling biochemical networks [AK08a].

Since the γ-calculus and the HOCL were shown to be well-suited formalisms for modeling autonomous systems [Rad07], we also investigate the suitability of the ρ_{pg}-calculus calculus for such an application [AK08c, AK08b]. In particular the use of strategy as objects (molecules) in the calculus helps a system self-managing and coordinating the behaviors of its components. This study is also relevant for modeling biological systems because of their highly complex and autonomous behavior. We use the ρ_{pg}-calculus for modeling a fragment of the EGFR signaling pathway as well. Also in the context of modeling autonomic systems we analyze the possibility of embedding verification features in the calculus based on its higher-order capabilities.

Beyond Simulation: Embedding the Biochemical Calculus with Runtime Verification

In the context of modeling autonomous systems, runtime verification is useful for recovering from problematic situations, i.e., for the self-healing property. Typical requirements one may want a system to satisfy concern the occurrence, consequence or invariance of particular structural or behavioral properties. Such types of requirements are also interesting for verifying biochemical models [CRCFS04, MRM+08].

Thanks to the possibility of encoding strategies as objects of the calculus and to the multiverse construct which considers all possible ways of interaction between an abstraction and a molecule, we endow the ρ_{pg}-calculus with an automated method for validating the behavior of the system with respect to some initial design requirements or properties. We express the requirements as formulas in 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 based on port graph expressions which we encode by means of some adequate rewrite strategies. Then we verify that the modeled system satisfies an atomic proposition using the evaluation mechanism of the rewrite strategies. We put the temporal formulas at the same level as the system description in the ρ_{pg}-calculus and we obtain a runtime verification technique which allows the running system to detect its own failures. In addition, the modeled system can be provided with recovery strategies for tackling the failure of initial requirements.

In conclusion, we propose a higher-order biochemical formalism based on strategic rewriting on specific structures which is designed not only for simulating the evolution of a system in time, but also for verifying the systems structure and evolution with respect to given requirements.

Outline of the Thesis

The thesis is organized as follows:

Chapter 1 We review basic notions and concepts on rewriting and strategies that we use in the thesis.

Chapter 2 We propose an Abstract Biochemical Calculus called the $\rho_{\langle\Sigma\rangle}$-calculus, with Σ describing the structure of molecules. We introduce its syntax and semantics stepwise, starting from the basic intuition, then making the application of an abstraction to a molecule explicit. We then define strategies as abstractions in the calculus.

Chapter 3 We define the structure of port graphs, a matching algorithm for port graphs, port graph rewrite rules and a rewriting relation on port graphs. We also study the confluence property for port graph rewriting.

Chapter 4 Based on the structure of port graphs, we instantiate the $\rho_{\langle\Sigma\rangle}$-calculus to obtain a biochemical calculus based on strategic port graph rewriting. We illustrate the expressivity power of the port graph structure by defining the matching and

the replacement mechanisms in the calculus via evaluation rules on port graphs which are detailed in Appendix A.

Chapter 5 We give an operational semantics for the port graph rewriting based on algebraic terms over a suitable order-sorted signature. This term encoding of port graphs and port graph rewriting permits us to instantiate the ρ-calculus to obtain a rewriting calculus for terms encoding port graphs.

Chapter 6 We illustrate the suitability of the ρ_{pg}-calculus for modeling autonomous systems thanks to the strategies encoded as molecules in the calculus. We also instantiate the ρ_{pg}-calculus with the particular molecular graph structure of proteins for modeling a fragment of the epidermal growth factor receptor (EGFR) signaling pathway and give the main ideas of the corresponding implementation in TOM described in Appendix C.

Chapter 7 We extend the syntax and the semantics of the calculus with a class of temporal formulas for verifying the satisfiability of the formulas. We obtain in this way a biochemical calculus with runtime verification capabilities. We illustrate the advantages of the runtime verification on some biological examples with an emphasis on the self-healing property of biological systems.

We end the thesis with some final conclusions and perspectives.

In Figure 0.2 we provide a diagrammatic view of the relations between the concepts we introduced in each chapter.

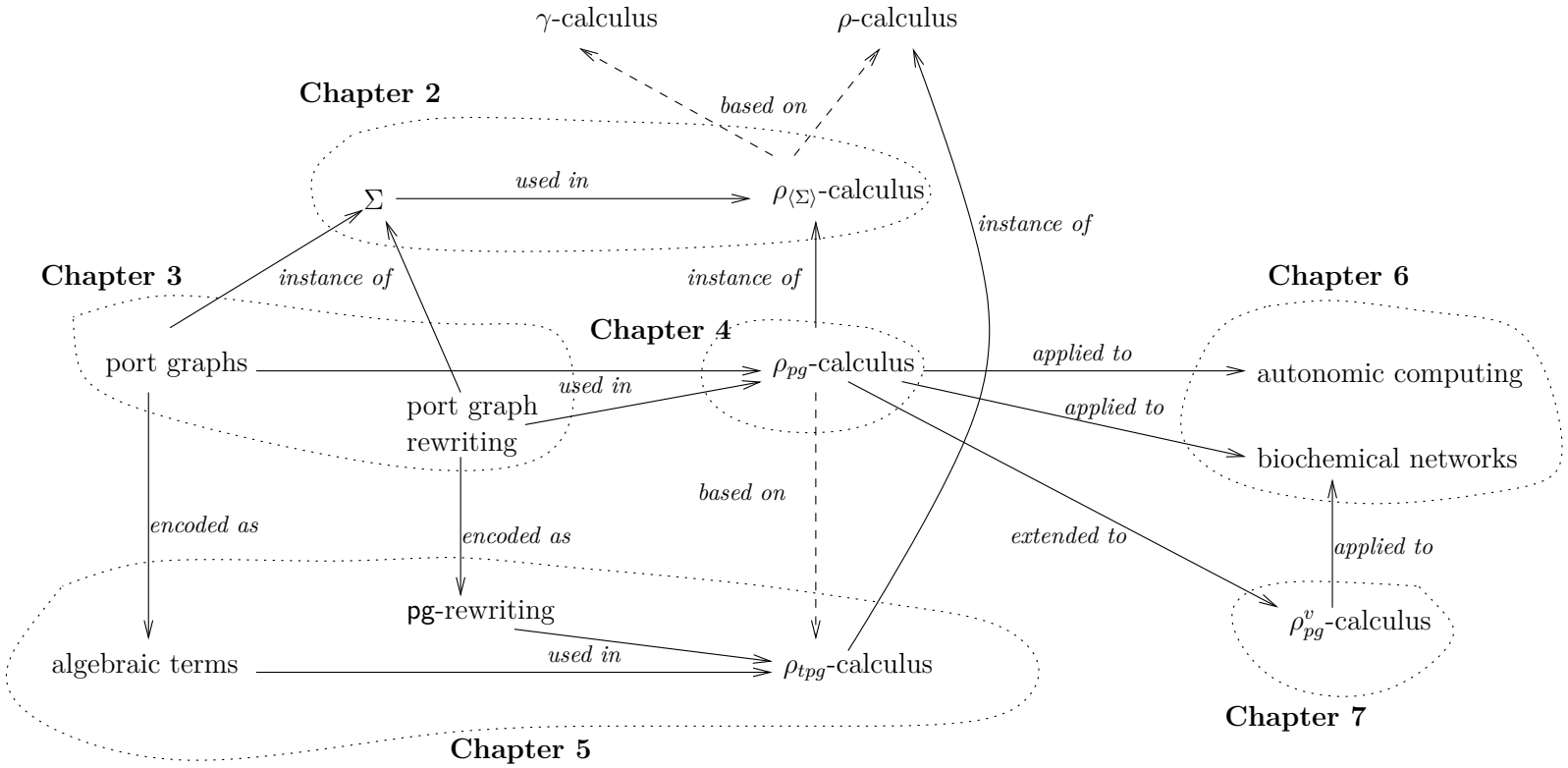


Figure 0.2: The relations between the concepts and the chapters in the thesis

1 Preliminary Notions

We present in this chapter the necessary background concerning term rewriting, graph rewriting and strategic rewriting.

1.1 Binary relations and their properties

In the following we review basic definitions and notations, as well as usual properties of binary relations [BN98].

Definition 1 (Binary relations)**.** *Given two binary relations $R \subseteq A \times B$ and $S \subseteq B \times C$, their* composition *is defined by*

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

Let $\to$ be a binary relation on a set A. We denote by:

- $\to^0$ *the identity on A,*
- $\to^n$ *the n-fold composition of $\to$, $\to^n = \to \circ \to^{n-1}$, for every $n > 0$,*
- $\to^=$ *the reflexive closure of $\to$, $\to^= = \to \cup \to^0$,*
- $\leftarrow$ *is the inverse of $\to$, $\leftarrow = \{(y, x) \mid x \to y\}$,*
- $\leftrightarrow$ *the symmetric closure of $\to$, $\leftrightarrow = \to \cup \leftarrow$*
- $\to^+$ *the transitive closure of $\to$, $\to^+ = \cup_{n>0} \to^n$,*
- $\to^*$ *the reflexive transitive closure of $\to$, $\to^* = \to^0 \cup \to^+$,*
- $\leftrightarrow^*$ *the reflexive transitive symmetric closure of $\to$.*

Definition 2 (Reducibility)**.** *Let $\to$ be a relation over a set A. An element x in A is* reducible *if there exists an element y in A such that $x \to y$; x is* irreducible *(or in* normal form*) if it is not reducible. A* normal form *of x is any irreducible element y such that $x \to^* y$. Two elements x and y in A are joinable if there exists z in A such that $x \to^* z$ and $y \to^* z$ and we denote it by $x \downarrow y$.*

Definition 3 (Properties of binary relations)**.** *Let $\to$ be a relation over a set A. The relation $\to$ is called:*

- locally confluent *if $x \to y_1$ and $x \to y_2$ implies $y_1 \downarrow y_2$;*

- confluent *if* $x \to^* y_1$ *and* $x \to^* y_2$ *implies* $y_1 \downarrow y_2$*;*
- strongly normalizing *(or* terminating*) if there is no infinite sequence* $x_0 \to x_1 \to \ldots$*;*
- normalizing *if every element in A has a normal form;*
- convergent *if it is confluent and terminating.*

Proving the confluence of a relation is in general difficult. But if the relation is terminating, is sufficient to show that the relation is locally confluent.

Theorem 1 (Newman's Lemma [New42])**.** *A strongly terminating relation is confluent if it is locally confluent.*

1.2 Abstract Reduction Systems

Usually an abstract reduction system is described by a set and a binary relation over that set. For the purpose of this thesis, in particular for reasoning later on the notion of strategies, we adopt the more general definitions from [KKK08] based on the notion of graph. These definitions allow one to describe the possible different ways an object is reached from another one.

Definition 4 (Abstract reduction system)**.** *An* abstract reduction system *(ARS) is a labelled oriented graph* $(\mathcal{O}, \mathcal{S})$*. The nodes in* $\mathcal{O}$ *are called* objects*, the oriented edges in* $\mathcal{S}$ *are called* steps*.*

Definition 5 (Derivation)**.** *For a given ARS* $\mathcal{A}$*:*

1. *A* reduction step *is a labelled edge* ϕ *together with its source a and target b. This is written* $a \to_{\mathcal{A}}^{\phi} b$*, or simply* $a \to^{\phi} b$ *when unambiguous.*
2. *An* $\mathcal{A}$-derivation *or* $\mathcal{A}$-reduction sequence *is a path* π *in the graph* $\mathcal{A}$*.*
3. *When it is finite,* π *can be written* $a_0 \to^{\phi_0} a_1 \to^{\phi_1} a_2 \ldots \to^{\phi_{n-1}} a_n$ *and we say that* a_0 *reduces to* a_n *by the derivation* $\pi = \phi_0\phi_1 \ldots \phi_{n-1}$*; this is also denoted by* $a_0 \to^{\pi} a_n$*. The* source *of* π *is the singleton* $\{a_0\}$ *denoted by* $dom(\pi)$*. The* target *of* π *is the singleton* $\{a_n\}$ *and it is denoted by* $[\pi](a_0)$*.*
4. *A derivation is* empty *when its source and target are the same. The empty derivation issued from a is denoted by* id_a*.*
5. *The* concatenation *of two derivations* $\pi_1; \pi_2$ *is defined when* π_1 *is finite and* $dom(\pi_2) = [\pi_1](dom(\pi_1))$ *as follows:*

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

Note that an $\mathcal{A}$-derivation is the concatenation of its reduction steps. The concatenation of π_1 and π_2 when it exists, is a new $\mathcal{A}$-derivation.

The following definitions generalize classical properties of a relation to an ARS.

Definition 6 (Termination)**.** *For a given ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ we say that:*

- *$\mathcal{A}$ is* terminating *(or* strongly normalizing*) if all its derivations are of finite length;*
- *an object a in $\mathcal{O}$ is* normalized *when the empty derivation is the only one with source a (e.g., a is the source of no edge);*
- *a derivation is* normalizing *when its target is normalized;*
- *an ARS is* weakly terminating *if every object a is the source of a normalizing derivation.*

Definition 7 (Confluence)**.** *An ARS $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ is* confluent *if for all objects a, b, c in $\mathcal{O}$, and all $\mathcal{A}$-derivations π_1 and π_2, when $a \twoheadrightarrow^{\pi_1} b$ and $a \twoheadrightarrow^{\pi_2} c$, there exist d in $\mathcal{O}$ and two $\mathcal{A}$-derivations π_3, π_4 such that $c \twoheadrightarrow^{\pi_3} d$ and $b \twoheadrightarrow^{\pi_4} d$.*

1.3 First-order Term Rewriting

This section contains the basic notions on first-order term algebra and term rewriting [BN98, GM92].

1.3.1 Term Algebra

A *many-sorted signature* is a pair $(\mathcal{S}, \mathcal{F})$ where $\mathcal{S}$ is a set of sorts and $\mathcal{F}$ a set of sorted function symbols, $\mathcal{F} = \{\mathcal{F}_{S_1 \dots S_n, S} \mid S_1, \dots S_n, S \in \mathcal{S}\}$. For $f \in \mathcal{F}_{S_1 \dots S_n, S}$ we use the notation $f : S_1 \ \dots \ S_n \to S$. An *order-sorted signature* is a triple $(\mathcal{S}, \leq, \mathcal{F})$ such that $(\mathcal{S}, \mathcal{F})$ is a many-sorted signature and $(\mathcal{S}, \leq)$ is a partially ordered set, and the function symbols satisfy a monotonicity condition: if $f \in \mathcal{F}_{S_1 \dots S_n, S} \cap \mathcal{F}_{S'_1 \dots S'_n, S'}$ and $S_i \leq S'_i$ for all i, $1 \leq i \leq n$, then $S \leq S'$. In the following, for presenting term rewriting we consider only many-sorted signatures; a complete introduction on order-sorted algebra can be found in [GM92].

When $f \in \mathcal{F}_{S_1 \dots S_n, S}$, we say that f has the *rank* $\langle S_1 \dots S_n, S \rangle$, *arity* $S_1 \dots S_n$, and *sort* S. If $n = 0$, then f is called a *constant*. If f has the arity $S \dots S$ of a variable size, then f is *variadic*. In general, when $\mathcal{S}$ is a singleton, the arity of a function symbol is reduced to a number.

Let $(\mathcal{S}, \mathcal{F})$ be a many-sorted signature and $\mathcal{X} = \{\mathcal{X}_S\}_{S \in \mathcal{S}}$ be an $\mathcal{S}$-sorted family of disjoint sets of variables.

Definition 8. *The set of terms of sort S over the signature $(\mathcal{S}, \mathcal{F})$ and the set of variables $\mathcal{X}$, denoted $\mathcal{T}(\mathcal{F}, \mathcal{X})_S$, is the smallest set containing $\mathcal{X}_S$ such that $f(t_1, \dots, t_n)$ is in $\mathcal{T}(\mathcal{F}, \mathcal{X})_S$ whenever $f : S_1 \dots S_n \to S$ and $t_i \in \mathcal{T}(\mathcal{F}, \mathcal{X})_{S_i}$ for $1 \leq i \leq n$, $n \geq 0$. Then $\mathcal{T}(\mathcal{F}, \mathcal{X}) = \bigcup \mathcal{T}(\mathcal{F}, \mathcal{X})_{S \in \mathcal{S}}$ is the* term algebra *generated by the signature $(\mathcal{S}, \mathcal{F})$ and the set of variables $\mathcal{X}$.*

The top symbol of a term is denoted $Head(t)$. The set of variables occurring in a term t is denoted by $\mathcal{V}ar(t)$. If $\mathcal{V}ar(t)$ is empty, t is called a *ground term*. $\mathcal{T}(\mathcal{F})$ is the set of all ground terms. We may omit sort names when they are clear from the context. A term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is said to be *linear* if each variable in t occurs at most once.

Let $\mathbb{N}$ be the set of natural numbers, $\mathbb{N}_+$ the set of non-zero naturals. The set of finite sequences of non-zero natural numbers $\mathbb{N}_+^*$ is defined as $p = \epsilon \mid n \mid p.p$, where ϵ represents the empty sequence and $n \in \mathbb{N}_+$. For all $p, q \in \mathbb{N}_+^*$, p is a prefix of q if there is $r \in \mathbb{N}_+^*$ such that $q = p.r$.

The set of *positions* $\mathcal{P}os(t)$ of the term t is recursively defined as follows:

- $\epsilon \in \mathcal{P}os(t)$ is the head position of t.
- For all $p \in \mathcal{P}os(t)$ and all $i \in \mathbb{N}_+^*$, $p.i \in \mathcal{P}os(t)$ if and only if $1 \leq i \leq |arity(f)|$ where $f \in \mathcal{F}$ is the symbol at the position p of t.

We call *subterm* of t at the position $p \in \mathcal{P}os(t)$ the term denoted $t_{|p}$ which satisfies the following condition:

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

We denote $t[s]_p$ the term t where the subterm at the position p has been replaced by the term s.

Example 1. The set of Peano integers can be described by a signature consisting of a single sort $\mathcal{S} = \{Nat\}$ and a set of function symbols:

$$\mathcal{F} = \{s : Nat \to Nat, 0 : \to Nat, \ plus : Nat \ Nat \to Nat\}$$

for succesor, zero, and addition operations. The set of positions of the term $plus(s(0), s(s(0)))$ is $\mathcal{P}os(t) = \{\epsilon, \ 1 \ , \ 2, \ 1.1, \ 2.1, \ 2.1.1\}$ which corresponds respectively to the subterms $plus(s(0), s(s(0)))$, $s(0)$, $s(s(0))$, 0, $s(0)$ and 0.

A *substitution* σ is a mapping from each variable in a finite subset $\{x_1, \ldots, x_k\}$ of $\mathcal{X}$ to a term of the same sort in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, written $\sigma = \{x_1 \mapsto t_1, \ldots, x_k \mapsto t_k\}$. We define the domain of σ as $dom(\sigma) = \{x_1, \ldots, x_k\}$. The application of a substitution σ to a term t, denoted by $\sigma(t)$ simultaneously replaces all occurrences of variables by their respective σ-images. The composition of two substitutions σ and μ is denoted $\sigma\mu$ and $(\sigma\mu)(t) = \sigma(\mu(t))$ for any term t. We say that σ instantiates x if $x \in dom(\sigma)$.

A substitution σ is *more general* than a substitution σ' if there is a substitution δ such that $\sigma' = \delta\sigma$. In this case we write $\sigma \lesssim \sigma'$. We also say that σ' is an instance of σ.

Two terms are *unifiable* if there is a substitution σ such that $\sigma(s) = \sigma(t)$. Then σ is a *most general unifier (mgu)* for s and t if for any other unifier σ' of s and t, $\sigma \lesssim \sigma'$.

Example 2. On the example on Peano integers above we consider a set of variables $\{x, y\}$ and a substitution $\sigma = \{x \mapsto 0, y \mapsto s(0)\}$. Then for $t = plus(s(x), s(y))$ we have $\sigma(t) = plus(s(0), s(s(0)))$.

Definition 9 (Matching)**.** *We say that a term t* matches *a term t', or t' is an* instance *of t, if there is a substitution σ such that $t' = \sigma(t)$.*

We usually refer to t as the pattern and to t' as the subject of the matching. This type of matching is known as *syntactical matching*. Syntactical matching is always decidable. It is linear on the size of the pattern, if this last one is a linear term. Otherwise, matching is linear on the size of the subject.

1.3.2 Equational Theories

An *equality* or *axiom* over a term algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is a pair of terms $\langle l, r\rangle$, denoted by $l = r$, where l and r are terms of the same sort. Given a set of axioms E, we denote by $\longleftrightarrow_E$ the symmetric binary relation over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ defined by $s \longleftrightarrow_E t$ if there is an axiom $l = r$ in E, a position p in s and a substitution σ such that $s|_p = \sigma(l)$ and $t = s[\sigma(r)]_p$. The reflexive and transitive closure of $\longleftrightarrow_E$, denoted by $\overset{*}{\longleftrightarrow}_E$, is the *equational theory generated by* E, or briefly, the *equational theory* E.

Some theories we mention in this thesis are defined below for a binary operator f:

(A)	Associativity	$f(f(x,y),z) = f(x,f(y,z))$
(C)	Commutativity	$f(x,y) = f(y,z)$
(I)	Idempotency	$f(x,x) = x$
(U_e)	Unit	$f(x,e) = f(e,x) = x$

We can combine these theories to obtain for instance associative with unit element (AU), associative-commutative (AC), or associative-commutative with unit element (ACU) theories. In addition, an equational theory E is called a *permutative theory* if for every equation $s \longleftrightarrow_E t$, the number of occurrences of every symbol in s is the same as in t.

Deciding whether two arbitrary terms are equal in an equational theory is known as the *word problem* in this theory.

The notion of matching can be generalized to take into account the fact that terms can be equal *modulo* a given equational theory. We say that a term t *matches modulo* E a term s if there exists a substitution σ such that $s \overset{*}{\longleftrightarrow}_E \sigma(t)$.

In contrast to the syntactical matching problem, matching modulo an equational theory is undecidable in general [BS01]. When they can be decided, the available algorithms may have a considerable complexity. Well-known examples are matching modulo associativity and commutativity.

1.3.3 Term Rewriting

Let $(\mathcal{S}, \mathcal{F})$ and $\mathcal{X}$ denote as usual a many-sorted signature and a variable set as before.

Definition 10 (Rewrite rule)**.** *A* rewrite rule *for the term algebra $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is an oriented pair of terms, denoted $l \to r$, where l and r are terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$. We call l and r respectively right-hand side and left-hand side of the rule.*

A term rewrite system *is a set R of rewrite rules for $\mathcal{T}(\mathcal{F}, \mathcal{X})$.*

Sometimes we add labels to rules to identify them. A labeled rewrite rule has the form $id : l \to r$.

Some restrictions are usually imposed on a rewrite rule $l \to r$:

- $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$ (the set of variables from the right-hand side is a subset of the set of variables of the left-hand side),
- $l \notin \mathcal{X}$ (the left-hand side is not a variable),
- l and r are of the same sort.

Definition 11 (Rewrite Relation)**.** *Let R be a rewrite system over $\mathcal{T}(\mathcal{F}, \mathcal{X})$. The rewrite relation associated to R over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is denoted $\rightarrow_R$ and is defined as follows: $t \rightarrow_R s$ if there exists a position p in t, a rewrite rule $l \rightarrow r$ in R and a substitution σ such that $t|_p = \sigma(l)$ and $s = t[\sigma(r)]_p$. The subterm $t|_p$ is an instance of the left-hand side l and it is called a* redex.

Example 3. The operator *plus* for Peano integers can be defined by the following term rewrite system:

$$R = \begin{cases} r_1 : & plus(0, y) & \rightarrow & y \\ r_2 : & plus(s(x), y) & \rightarrow & s(plus(x, y)) \end{cases}$$

The term $t = plus(s(0), s(s(0)))$ is normalized by the following derivation:

$$plus(s(0), s(s(0))) \rightarrow s(plus(0, s(s(0)))) \rightarrow s(s(s(0)))$$

The properties of a term rewrite system R are those of the relation $\rightarrow_R$. All these properties, in particular termination and confluence are undecidable in general. This is not surprising because term rewriting is at least as expressive as Turing machines. Indeed, Turing machines can be expressed as a single rewrite rule [Dau92].

However, there are methods for deciding these properties for specific classes of term rewrite systems. For example, termination of a term rewrite system can be proved through the use of an appropriate simplification ordering thanks to the theorem below. A *rewrite order* is a compatible order over the set of terms. A *simplification order* is a rewrite order which contains the strict subterm relation.

Theorem 2. *[Der82] Let $\mathcal{F}$ be a signature with a finite set of symbols. A term rewrite system R over $\mathcal{T}(\mathcal{F}, \mathcal{X})$ terminates if there is a simplification order $\succ$ such that $l \succ r$ for each rule $l \rightarrow r \in R$.*

Confluence can be decided for terminating term rewrite systems by applying the Newman's lemma which assures that local confluence implies the confluence for these systems. Local confluence can be decided by testing the joinability of *critical pairs* [BN98].

Definition 12 (Critical Pair)**.** *Let $l \rightarrow r$ and $g \rightarrow d$ be two rules with disjoint sets of variables. We call a critical pair in the rule $g \rightarrow d$ over $l \rightarrow r$ at the non variable position $p \in \mathcal{P}os(l)$, the pair $(\sigma(r), \sigma(l)[\sigma(d)]_p)$ such that σ is a most general unifier of g and $l|_p$.*

If every critical pair is joinable, the term rewrite system is locally confluent. Since the number of critical pairs in a finite term rewrite system is also finite, local confluence is decidable.

Conditional rewrite systems arise naturally in some of the specifications adopted in this thesis.

Definition 13 (Conditional Rewriting)**.** *A conditional term rewrite system is a set of conditional rewrite rules R over a set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Each rewrite rule is of the form $l \to r$ if $s_1 \to t_1, \ldots, s_k \to t_k$ with $l, r, s_1, \ldots, s_k, t_1, \ldots t_k \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.*

- *For all rules in R term rewrite system $\mathcal{V}ar(r) \cup \mathcal{V}ar(c) \subseteq \mathcal{V}ar(l)$, where c is an abbreviation for the conditional part of the rule, $s_1 \to t_1, \ldots, s_k \to t_k$.*
- *Each t_j in c is a ground normal form with respect R_u, which contains all rules in R without their conditional part.*

Definition 14. *Given a conditional rewrite system R, a term t rewrites to a term t', which is denoted as usual $t \to_R t'$ if there exists a conditional rewrite rule $l \to r$ if c, a position ω in t, and a substitution σ satisfying $t_{|\omega} = \sigma(l)$, and $\sigma(s_1) \to_{R_u} t_1, \ldots, \sigma(s_k) \to_{R_u} t_k$.*

We now introduce the notion of rewriting modulo a set of equations. When the axioms of an equational theory can be oriented into a canonical term rewrite system, the rewrite rules can be used for solving the word problem in such theory. However, there are equalities that cannot be oriented without loosing the termination property. A typical example is the commutativity axiom. In this case, equational reasoning needs a different rewrite relation which works on term equivalence classes modulo these non-orientable equalities.

Definition 15 (Rewriting Modulo Equivalence Classes)**.** *Given a term rewrite system R and a set of axioms $\mathcal{E}$, the term t rewrites into the term s by R modulo E, denoted $t \longrightarrow_{R/\mathcal{E}} s$, if there is a rule $l \to r \in R$, a term u, a position p in u and a substitution σ, such that $t \overset{*}{\longleftrightarrow}_{\mathcal{E}} u[\sigma(l)]_p$ and $s \overset{*}{\longleftrightarrow}_{\mathcal{E}} u[\sigma(r)]_p$.*

The relation $\longrightarrow_{R/\mathcal{E}}$ is not satisfactory with respect to efficiency because in order to rewrite a term, it is necessary to search in the whole equivalence class modulo E. Such a search is even harder in the case of infinite equivalence classes. In order to solve this problem, a weaker relation has been proposed by [PS81], and generalized by [JK86], in which matching is replaced by matching modulo an equational theory. This relation is called *rewriting modulo an equational theory* and is denoted $\to_{R,\mathcal{E}}$.

In practice, the most used equational theory is associativity and commutativity. The relation $\to_{R,\mathcal{E}}$ is called in this case rewriting modulo associativity and commutativity (AC). The efficiency of matching modulo AC is essential for the performance of rewriting modulo AC. However, matching modulo AC is know to the a NP-Hard problem [BKN87] and it can have an exponential number of solutions.

1.4 Elements of Category Theory

We review a few elements from the category theory [Mac98] needed in this thesis. We recall the definitions of category, functor, pushout, and strict symmetric strict monoidal category.

Definition 16 (Category)**.** *A category* **C** *is given by:*

- *A class of objects denoted by* $Obj(\mathbf{C})$.
- *A class of morphisms (or arrows) denoted by* $Arr(\mathbf{C})$*, where each morphism* f *has a unique source object* A *and target object* B*, with* A *and* B *objects of* $\mathbf{C}$*. We denote by* $\mathbf{C}(A, B)$ *the class of all morphisms from the object* A *to the object* B.
- *A composition law* $\circ : \mathbf{C}(A, B) \times \mathbf{C}(B, C) \to \mathbf{C}(A, C)$ *which is associative, that is*

$$\text{if } f \in \mathbf{C}(A, B), g \in \mathbf{C}(B, C), h \in \mathbf{C}(C, D) \text{ then } h \circ (g \circ f) = (h \circ g) \circ f.$$

- *An identity morphism* $\mathsf{id}_A \in \mathbf{C}(A, A)$ *for all objects* A *which is a neutral element for* $\circ$*, that is*

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

A functor is a morphism of categories.

Definition 17 (Functor)**.** *A* functor F *from a category* $\mathbf{C}$ *to a category* $\mathbf{D}$*, written* $F : \mathbf{C} \to \mathbf{D}$*, consists of two functions:*

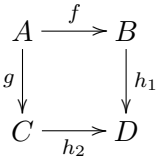
- *the object function which assigns to each object* A *in* $\mathbf{C}$ *an object* $F(A)$ *in* $\mathbf{D}$*, and*
- *the arrow function which assigns to each arrow* $f : A \to B$ *of* $\mathbf{C}$ *an arrow* $F(f) : F(A) \to F(B)$ *in* $\mathbf{D}$*,*

such that

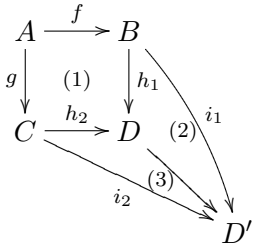
$$F(id_A) = id_{F(A)}, \qquad F(g \circ f) = F(g) \circ F(f)$$

Definition 18 (Pushout)**.** *Given in* $\mathbf{C}$ *a pair of arrows* $f : A \to B$ *and* $g : A \to C$*, a* pushout *of* f *and* g *consists of an object* D *and two arrows* $h_1 : C \to D$ *and* $h_1 : B \to D$ *for which the following two conditions are satisfied:*

(commutativity) *The diagram below commutes:*



(universality) *For every object* D' *and arrows* $i_1 : B \to D'$ *and* $i_2 : C \to D'$ *such that* $i_1 \circ f = i_2 \circ g$*, there is a unique port graph morphism* $D \to D'$ *the diagrams (2) and (3) below commute respectively.*



Definition 19 (Strict symmetric strict monoidal category)**.** *A* strict symmetric strict monoidal category *(sssmc)* $(B, +, e)$ *is a category* B *equipped with a bifunctor* $+ : B \times B \to B$ *called the tensor product or disjoint sum, and an object* e *called the unit object or identity object which satisfy:*

1. *the class of objects* $Ob(B)$ *is a commutative monoid with respect to* $+$ *with the neutral element* e*, (i.e.,* $+$ *is associative, commutative, with neutral element),*
2. *for every* $f \in B(a, b)$ *and* $g \in B(a', b')$ *the sum* $f + g$ *is in* $B(a + a', b + b')$ *and satisfies the following axioms:*

   (C1) $(f + g) + h = f + (g + h)$,
   (C2) $\mathsf{id}_e + f = f = f + \mathsf{id}_e$,
   (C3) $\mathsf{id}_a + \mathsf{id}_b = \mathsf{id}_{a+b}$,
   (C4) $(f + g); (u + v) = f; u + g; v$,
   (C5) $f + g = g + f$ *for any* $f, g \in Arr(B)$.

1.5 Labeled Graphs

Definition 20 (Labeled graph)**.** *A* label alphabet $\mathcal{L} = (\mathcal{L}_V, \mathcal{L}_E)$ *is a pair of sets of node labels and edge labels. A (finite)* graph *over* $\mathcal{L}$ *is a tuple* $G = (V, E, s^G, t^G, l^G)$ *where:*

- V *is a set* $\{v_1, \ldots, v_k\}$ *of elements called* nodes *(or* vertices*),*
- E *is a set* $\{e_1, \ldots, e_m\}$ *of elements of the Cartesian product* $V \times V$ *called* edges*,*
- $s^G, t^G : E \to V$ *are the source ant target functions respectively, and*
- $l^G = (l^G_V, l^G_E)$ *is the* labeling function *for nodes* ($l^G_V : V \to \mathcal{L}_V$) *and edges* ($l^G_E : E \to \mathcal{L}_E$)*.*

If G is a graph, we usually denote by V_G its node set and by E_G its edge set.

An edge of the form (v, v) is called a *loop*. For an edge (u, v), u and v are called *end nodes* with u the *source* and v the *target*; moreover we say that u and v are *adjacent* or *neighbouring* nodes, with v neighbour of u. An edge is *incident* to a node if the node is one of its end nodes. An edge is *multiple* if there is another edge with the same source and target; otherwise it is *simple*. A *multigraph* is a graph allowing multiple edges and loops. A *path* in a sequence of nodes $\{v_1, \ldots, v_n\}$ such that (v_1, v_2), ..., (v_{n-1}, v_n) are edges of the graph.

An *adjacency list* for a node is given by a list of pairs consisting of a neighbour and the corresponding edge label. If a node has no neighbour then its adjacency list is empty.

A *subgraph* of a graph G is a graph whose node and edge sets are subsets of those of G. A subgraph H of a graph G is said to be *induced* if, for any pair of vertices v and u of H, (v, u) is an edge of H if and only if (v, u) is an edge of G. In other words, H is an induced subgraph of G if it has all the edges that appear in G over the same vertex set.

A *graph morphism* $f : G \to H$ is a pair of functions $f_V : V_G \to V_H$ and $f_E : E_G \to E_H$ which preserve sources, targets, and labels while preserving adjacency, i.e., which satisfies $f_V \circ t^G = t^H \circ f_E$, $f_V \circ s^G = s^H \circ f_E$, $l_V^H \circ f_V = l_V^G$, $l_E^H \circ f_E = l_E^G$.

A *partial graph morphism* $f : G \to H$ is a total graph morphism from some subgraph $dom(f)$ of G to H, with $dom(f)$ called the domain of f.

The composition of two (partial) graph morphisms is defined by the composition of the components, and the identities as pairs of component identities.

The category having labeled graphs as objects and graph morphisms as arrows is called **Graph**. By restricting the arrows to partial morphisms, a new category is obtained called $\mathbf{Graph^P}$.

1.6 Graph Transformation

We briefly describe in this section the intuition behind graph transformation and the algebraic approach, also called "double-pushout approach" [Roz97, EEKR97, EKMR97].

A *graph transformation rule* $L \rightsquigarrow R$ consists of two graphs L and R called the left- and right-hand side respectively, and a partial graph morphism between. The partial graph morphism provides a correspondence between elements of the left-hand side and elements of the right-hand side. Graphically this correspondence is provided by some unique identifiers associated to nodes.

Definition 21 (Graph transformation rule)**.** *In the DPO approach a* production $p : L \leftarrow K \rightarrow R$ *is a pair of graph homomorphisms* $l : K \to L$ *and* $r : K \to R$ *where* L, K, R *are finite graphs and are called the* left-hand side*, the* interface *and the* right-hand side *respectively. A* graph transformation systems *is a finite set of graph transformation rules.*

A graph direct derivation can be formalised in the categorical setting as a single pushout *SPO* [EHK+97] or double pushout *DPO* [CMR+97]. We present here the double pushout approach, which historically was the first one to be proposed.

Definition 22 (Direct derivation)**.** *A* direct derivation *from* G *to* H *using a production* $p : L \leftarrow K \rightarrow R$ *exists if and only if the diagram below can be constructed,*

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

where both squares are required to be pushouts in the category **Graph**. *In this case* D *is called the* context graph.

Given a production, the graph K can be seen as an "interface", in the sense that it is not affected by the rewrite step itself, but it is necessary for specifying how the right-hand side R is glued with the graph D. Intuitively, the context graph D is obtained from the given graph G by deleting all elements of G which have a pre-image in L but

not in K. The second pushout diagram models the insertion into H of all elements of R that do not have a pre-image in K.

In general, as presented in [Sch97], the application of a graph transformation rule $L \rightsquigarrow R$ to a graph G, called *host graph*, produces a new graph G' according to the following steps:

1. Find a matching morphism m for L in G (hence $m(L)$ is a subgraph of G).
2. Remove the subgraph $m(L)$ from G resulting in the context graph G^-.
3. Add $m(R)$ to the context graph G^-.
4. Reconnect $m(R)$ and G^-.

The differences between various approaches for graph replacement arise mainly in the last step, depending on the mechanism chosen for establishing connections between new and old nodes. Two particular problems are handled at this stage ([CMR+97, Sch97]). The first one refers to whether or not noninjective matching is allowed. For example, if two different nodes L are matched to one node in the host graph, and one of the two nodes is deleted and the other preserved, will the node in the host graph be deleted or preserved? The second problem concerns the dangling edges in the host graph which are unmatched edges with one endpoint deleted by the transformation rule. These two problematic situations are referred to as the identification and the dangling problem respectively.

The problem of finding a match for a given rule is an instance of the subgraph homomorphism problem which is known to be NP complete [Meh84]. In fact there many definitions for graphs and graph transformation depending on the motivations and the context of the application. There are several algorithms available for general graph pattern matching, let us mention here a few works [Ull76, LV02, Zün96, Val02]. Among the existing tools for graph transformation we mention AGG [ERT97], PROGRES [SWZ97], and Clean [PE93]. More information on the application domains and tools for graph transformation can be found in [EEKR97].

1.7 Strategic Rewriting

The notion of strategy used in this thesis is fundamental in rewriting. We give here a general presentation of the main ideas following the approach based on abstract reduction systems [KKK08].

A term rewrite system $\mathcal{R}$ or a graph transformation system $\mathcal{G}$ generates an abstract reduction system whose nodes are terms or graphs respectively, and whose oriented edges are rewriting steps or direct derivations. Then a derivation in $\mathcal{R}$ or in $\mathcal{G}$ is a path in the underlying graph of the associated abstract reduction system. The notions of strategy and strategic rewriting have been introduced in order to control such derivations.

Definition 23 (Abstract strategy)**.** *For a given ARS $\mathcal{A}$:*

1. *An* abstract strategy ζ *is a subset of the set of all derivations (finite or not) of* $\mathcal{A}$.

2. *Applying the strategy* ζ *on an object* a*, denoted by* $[\zeta](a)$*, consists of the set of all objects that can be reached from* a *using a derivation in* ζ*:*
   $[\zeta](a) = \{b \mid \exists \pi \in \zeta \text{ such that } a \twoheadrightarrow^{\pi} b\} = \{[\pi](a) \mid \pi \in \zeta\}$.
   When no derivation in ζ *has for source* a*, we say that the strategy application on* a *fails.*

3. *Applying the strategy* ζ *on a set of objects consists in applying* ζ *to each element* a *of the set. The result is the union of* $[\zeta](a)$ *for all* a.

4. *The* domain *of a strategy is the set of objects that are source of a derivation in* ζ*:*

$$dom(\zeta) = \bigcup_{\delta \in \zeta} dom(\delta)$$

5. *The strategy that contains all empty derivations is* $Id = \{id_a \mid a \in \mathcal{O}\}$.

Remark that, following from the previous definition, a strategy is not defined on all objects of the ARS, hence it is a partial function. A strategy that contains only infinite derivations from a source $\{a\}$ applies to the object a and returns the empty set. The empty set of derivations is a strategy called $Fail$; its application always fails.

The formalisation of abstract reduction systems and abstract strategies allows then to define properties like termination (all relevant derivations are of finite length) and confluence (all relevant derivations lead to the same object).

Definition 24 (Termination under strategy)**.** *For a given abstract reduction system abstract reduction system* $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ *and a strategy* ζ*:*

- $\mathcal{A}$ *is* terminating *under* ζ *if all derivation in* ζ *are of finite length;*
- *An object* a *in* $\mathcal{O}$ *is* ζ-normalized *when the empty derivation is the only one in* ζ *with source* a*;*
- *A derivation is* ζ-normalizing *when its target is* ζ*-normalized;*
- *An ARS is weakly* ζ-terminating *if every object* a *is the source of a* ζ*-normalizing derivation.*

Definition 25 (Weak confluence under strategy)**.** *An ARS* $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ *is* weakly confluent under a strategy ζ *if for all objects* a, b, c *in* $\mathcal{O}$ *and all derivations in* $\mathcal{A}$*-derivations* π_1 *and* π_2 *in* ζ*, when* $a \rightarrow^{\pi_1} b$ *and* $a \rightarrow^{\pi_2} c$ *there exists* d *in* $\mathcal{O}$ *and two derivations* π_3', π_4' *in* ζ *such that* $\pi_3' : a \rightarrow c \rightarrow d$ *and* $\pi_4' : a \rightarrow b \rightarrow c$

Definition 26 (Strong confluence under strategy)**.** *An ARS* $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ *is* strongly confluent under a strategy ζ *if for all objects* a, b, c *in* $\mathcal{O}$ *and all* $\mathcal{A}$*-derivations* π_1 *and* π_2 *in* ζ*, when* $a \rightarrow^{\pi_1} b$ *and* $a \rightarrow^{\pi_2} c$ *there exists* d *in* $\mathcal{O}$ *and two derivations* π_3 *and* π_4' *in* ζ *such that* $c \rightarrow^{\pi_3} d$ *and* $b \rightarrow^{\pi_4} d$ *and* $\pi_4; \pi_1, \pi_3; \pi_2 \in \zeta$.

It is now possible to give the definition of strategic rewriting:

Definition 27 (Strategic rewriting)**.** *Given an abstract reduction system $\mathcal{A} = (\mathcal{O}, \mathcal{S})$ and a strategy ζ, a strategic rewriting derivation (or rewriting derivation under strategy ζ) is an element of ζ.*

If the abstract reduction system $\mathcal{A}$ is generated by a term rewrite system R, a strategic rewriting step under ζ is a rewriting step $t \rightarrow_{\omega}^{R} t'$ that occurs in a derivation of ζ.

If $\mathcal{A}$ is generated by a graph transformation system, a strategic graph rewriting step ζ is a direct derivation $G \rightarrow H$.

A strategy can be described by enumerating all its elements or more suitably by a *strategy language*. Various approaches have been followed, yielding slightly different strategy languages such as ELAN [KKV95, BKKR01b], Stratego [Vis01], TOM[1] [BBK+07a, BBK+07c] or more recently Maude [MOMV05]. All these languages provide flexible and expressive strategy languages where high-level strategies are defined by combining low level primitives. We describe below the main elements of the TOM strategy language that are of interest in this thesis.

Following [KKK08], we can distinguish two classes of constructs in the strategy language: the first class allows construction of derivations from the basic elements, namely the rewrite rules. The second class corresponds to constructs that express the control, especially left-biased choice (or first). Moreover, the capability of expressing recursion in the language brings even more expressive power.

Elementary constructor strategies. An elementary strategy is either Identity which corresponds to the set Id of all empty derivations, Fail which denotes the empty set of derivations $Fail$, or a set of rewrite rules $\mathcal{R}$ which represents one-step derivations with rules in $\mathcal{R}$. $Sequence(\zeta_1, \zeta_2)$, also denoted by $\zeta_2; \zeta_1$, is the concatenation of ζ_1 and ζ_2 whenever it exists: for a given object a in an ARS $\mathcal{A}$, $[\zeta_1; \zeta_2](a) = [\zeta_2]([\zeta_1](a))$.

Control constructor strategies. A few constructions are needed to build derivations, branching and to take into account the structure of the objects.

first $First(\zeta_1, \zeta_2)$ applies the first strategy if it does not fail, otherwise it applies the second strategy; it fails if both strategies fail. Remark this is a deterministic choice, more precisely, the left-biased choice.
$[First(\zeta_1, \zeta_2)](a) = [\zeta_1](a)$ if $[\zeta_1](a)$ does not fail else $[\zeta_2](a)$.

not $Not(\zeta)$ fails if the strategy ζ does not fail, and it does nothing if ζ fails:
$[Not(\zeta)](a) = a$ if $[\zeta](a)$ fails else it fails.

if then else $IfThenElse(\zeta_1, \zeta_2, \zeta_3)$ applies the first strategy: if it does not fail, it applies the second strategy, else it applies the third strategy; it fails if both strategies ζ_2 and ζ_3 fail.
$[IfThenElse(\zeta_1, \zeta_2, \zeta_3)](a) = [\zeta_2](a)$ if $[\zeta_1](a)$ does not fail else $[\zeta_3](a)$.

[1] `http://tom.loria.fr`

fixpoint The μ recursion operator (comparable to *rec* in OCaml) is introduced to allow the recursive definition of strategies. $\mu x.\zeta$ applies the derivation in ζ with the variable x instantiated to $\mu x.\zeta$, i.e., $\mu x.\zeta = \zeta[x \leftarrow \mu x.\zeta]$

The strategies *Sequence* and $First$ extend naturally to be applicable to a list of strategies.

All these strategies are then composed to build other useful strategies. A composed strategy is for instance $Try(\zeta) = First(\zeta, Id)$ which applies ζ if it can, and applies the identity strategy Id otherwise. Similarly, the *Repeat* combinator is used in combination with the fixpoint operator to iterate the application of a strategy: $Repeat(\zeta) = \mu x.Try(\zeta; x)$.

If the objects in the ARS have a hierarchical structure (or term-like), then strategies are applied only on the top position and strategies such as bottom-up, top-down or leftmost-innermost are higher-order features that describe how rewrite rules should be applied. The basic control constructors for such strategies on terms are the following:

all subterms $All(\zeta)$ denotes the set of all derivations in ζ. On a term t, $All(\zeta)$ applies the strategy ζ on all immediate subterms:
$[All(\zeta)](f(t_1, ..., t_n)) = f(t'_1, ..., t'_n)$ if $[\zeta](t_i) = t'_i$ for all i, $1 \leq i \leq n$, and fails if there exists i such that $[\zeta](t_i)$ fails.

one subterm $One(\zeta)$ gives a way to select one derivation in ζ that does not fail if it exists. On a term t, $One(\zeta)$ applies the strategy ζ on the first immediate subterm of t where ζ does not fail:
$[One(\zeta)](f(t_1, ..., t_n)) = f(t_1, ..., t'_i, ..., t_n)$ if $[\zeta](t_i) = t'_i$ and for all j, $1 \leq j < i$, $[\zeta](t_j) = \emptyset$, and fails if for all i, $[\zeta](t_i)$ fails.

The *All* and *One* combinators are used in combination with the fixpoint operator to define tree traversals. For example, we have $TopDown(\zeta) = \mu x.Sequence(\zeta, All(x))$: the strategy ζ is first applied on top of the considered term, then the strategy $TopDown(\zeta)$ is recursively called on all immediate subterms of the term.

In addition to the strategy $First$, ELAN has two more strategies for describing the choice of strategies to apply from a set of strategies:

don't know $dk(\zeta_1, \ldots, \zeta_n)$ denotes the set of all derivations in ζ_1, ..., ζ_n. It fails if all strategies fail.

don't care $dc(\zeta_1, \ldots, \zeta_n)$ denotes the set of all derivations of a non-failing strategy ζ_i. It fails if all strategies fail.

Other high-level strategies implement term traversal and normalization on terms and are well-known in the term rewrite system literature:

$$\begin{aligned}
TopDown(\zeta) &= \zeta; All(TopDown(\zeta)) \\
BottomUp(\zeta) &= All(BottomUp(\zeta)); \zeta) \\
OnceTopDown(\zeta) &= Choice(\zeta, One(OnceTopDown(\zeta))) \\
OnceBottomUp(\zeta) &= Choice(One(OnceBottomUp(\zeta)), \zeta) \\
Innermost(\zeta) &= Repeat(OnceBottomUp(\zeta)) \\
Outermost(\zeta) &= Repeat(OnceTopDown(\zeta))
\end{aligned}$$

Example 4. Some examples of strategy application are:

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

2 An Abstract Biochemical Calculus

2.1 Introduction

In this chapter we introduce an abstract model of biochemical computation. The model is based on the principles of the chemical model as modeled by the γ-calculus and the principles of the rewriting calculus, and it combines rewriting on multisets of abstract structures with higher-order features from λ-calculus.

The chemical metaphor was proposed as a computational paradigm in the Γ language in [BM86]. It describes computations in terms of a chemical solution in which molecules representing data interact according to reaction rules. Chemical solutions are represented by multisets and the reaction rules by rewrite rules on such multisets. Then the computation proceeds by applications of rewrite rules which consume and produce new elements according to the conditions and transformations specified by the reactions rules. This model was used as a basis for defining the CHemical Abstract Machine (CHAM) [BB92].

In order to understand the particular features of the Abstract Biochemical Calculus we propose in this chapter, first we present the chemical model as modeled by the γ-calculus and HOCL, then the rewriting calculus. In a third step we talk about the passage to a biochemical model.

2.1.1 The γ-calculus and HOCL

The γ-calculus [BFR04, Rad07] was designed as a basic higher-order calculus developed on the essential features of the chemical paradigm. It generalizes the chemical model by considering the reactions as molecules as well. The fundamental structure is the multiset. The syntax of the calculus defines a molecule as either a variable $x \in \mathcal{X}$, an abstraction $\gamma\langle p\rangle.M$, a solution $\langle M\rangle$, or a multiset of solutions. A γ-abstraction is a function with only one argument, a variable from a set $\mathcal{X}$. We call these arguments *patterns* in order to emphasize the motivation of the definition of our calculus later on.

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

Figure 2.1: The syntax of the γ-calculus

The associative and commutative properties of the multiset construct helps simulating the Brownian movement of molecules in a solution. When an abstraction comes into

contact with an inert solution, a reaction takes place according to the following rewrite rule, called the γ-*reduction*:

$$(\gamma\langle x\rangle.M), \langle N\rangle \rightarrow_\gamma M[x := N] \text{ if } \langle N\rangle \text{ is inert}$$

where a solution is *inert* if it contains only abstractions and variables or only solutions and variables. The semantics of the γ-calculus contains, in addition to the above rule, two more rules for expressing reactions in a context. The *locality* and *solution* rules state that any rewriting operation on a given molecule also operates on a larger multiset and solution respectively.

The higher-order nature of the γ-calculus allows the encoding of some programming elements in the calculus as complex expressions as one can do in the λ-calculus, like for instance boolean operators, integers, identifiers for molecules, pairs of molecules, or recursivity. However these constructs are not always practical and expressive enough for modeling more complex programs. One aim of the chemical programming model as proposed by the authors of the γ-calculus is to model complex and large autonomous systems. In order to cope with this problem, the Higher-order Chemical Language (HOCL) [BFR06c, BFR06a, BFR07] was introduced. It extends the γ-calculus with the programming elements enumerated above, with conditional reactions and atomic capture (for reactions with n arguments), and, most importantly, with the following richer language of patterns $\mathcal{P}$:

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

For any patterns $P, P_1, P_2 \in \mathcal{P}$ and molecules $M, M_1, M_2 \in \mathcal{M}$, ω matches any molecule including the empty molecule, $\mathit{ident} = P$ matches any molecule named *ident* that matches P, the pattern P_1, P_2 matches any molecule M_1, M_2 such that P_i matches M_i, for $i = 1, 2$, and $\langle P\rangle$ matches any solution $\langle M\rangle$ such that P matches M. Then an abstraction takes the form $\gamma\lfloor C\rfloor P.M$ with C a condition; a condition is a molecule that should be evaluated to a constant **true** (representing the γ-abstraction $\gamma\langle x\rangle\lfloor x\rfloor.x$) for the reaction to take place. An abstraction is consumed by a γ-reduction; hence it is called a *one-shot reaction rule*. Then, based on the recursion mechanism encoded by a constant **let rec**, *n-shot reaction rules* are defined as reaction rules which are not consumed by a γ-reduction.

Example 5. Let us consider some examples of molecules in the γ-calculus together with some possible reductions. We use integers and the operation of addition on integers _+_ which can be encoded by suitable molecules following the same lines as in the λ-calculus.

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

2.1.2 The ρ-Calculus

The first version of the rewriting calculus (or the ρ-calculus) was introduced by H. Cirstea and C. Kirchner [CK01] to give a semantics of the rewrite based language ELAN [CK98]. In [CKL02] a simplified version of the rewriting calculus was proposed. The ρ-calculus extends first-order term rewriting and the λ-calculus. From the λ-calculus, the ρ-calculus inherits its higher-order capabilities and the explicit treatment of functions and their applications. It was introduced to make all the basic ingredients of rewriting explicit objects, in particular the notions of rewrite rule (or abstraction) "$\_ \twoheadrightarrow \_$", rule application "$\_\ \_$", and structure of results "$\_ \wr \_$". In the ρ-calculus, the usual λ-abstraction $\lambda x.t$ is replaced by a rule abstraction $p \twoheadrightarrow t$, where p and t are two arbitrary terms, with p called a pattern, and the free variables of p are bound in t. The ρ-calculus generalizes the λ-calculus by abstracting over a pattern instead of a simple variable. This kind of generalization is a key feature in our approach.

The syntax is defined in Fig. 2.2 where $\mathcal{X}$ is the set of variables and $\mathcal{K}$ is the set of function symbols. The operator "$\_ \wr \_$" groups terms together into *structures*, and, depending on the chosen theory for this operator, it provides lists, sets or multisets to represent multiple results.

Terms	$\mathcal{T} ::= \mathcal{X} \mid \mathcal{K} \mid \mathcal{P} \twoheadrightarrow \mathcal{T} \mid \mathcal{T}\ \mathcal{T} \mid \mathcal{T} \wr \mathcal{T}$
Patterns	$\mathcal{P} \subseteq \mathcal{T}$

Figure 2.2: The syntax of ρ-calculus

The small-step reduction semantics of the ρ-calculus is defined by the two evaluation rules in Fig. 2.3. If σ is a substitution solution of the matching problem $p \ll t_3$, then the application of the rewrite rule to t_3 evaluates to $\sigma(t_2)$. The set of patterns $\mathcal{P}$ is not a priori fixed ($\mathcal{P}$ is a parameter of the calculus), and the matching power of the ρ-calculus can be regulated using arbitrary theories. Therefore the semantics of the calculus and its properties depend essentially on these parameters.

(ρ)	$(p \twoheadrightarrow t_2)t_3$	$\rightarrow_\rho$	$\sigma_1(t_2) \wr \ldots \wr \sigma_n(t_2) \wr \ldots$ with $\sigma_i \in \mathcal{S}ol(p \ll t_3)$
(δ)	$(t_1 \wr t_2)t_3$	$\rightarrow_\delta$	$t_1t_3 \wr t_2t_3$

Figure 2.3: The semantics of the ρ-calculus

An important feature of the ρ-calculus is its capability of encoding *rewrite strategies* as shown in [CKLW03]. The basic strategies are the rewrite rules. An immediate application of the use of rewrite strategies in the ρ-calculus is the encoding of conditional rewriting [CK01]. The ρ-calculus has been proved confluent for linear algebraic patterns [CF07].

Example 6. Let us consider some ρ-terms over a set of constants a, b, variables x, y, the addition operation $\_ + \_$, and their reductions [CK01]:

1. $(y \twoheadrightarrow ((x \twoheadrightarrow x + y)\ b))\ ((x \twoheadrightarrow x)\ a) \rightarrow_\rho (y \twoheadrightarrow (b + y))\ ((x \twoheadrightarrow x)\ a) \rightarrow_\rho$
   $(y \twoheadrightarrow (b + y))\ a \rightarrow_\rho b + a$
   The ρ-term $(y \twoheadrightarrow ((x \twoheadrightarrow x+y)\ b))\ ((x \twoheadrightarrow x)\ a)$ corresponds to the second γ-molecule from Example 5.

2. $(((x \twoheadrightarrow x + 1) \twoheadrightarrow (1 \twoheadrightarrow x))\ (a \twoheadrightarrow a + 1))\ 1 \rightarrow_\rho (1 \twoheadrightarrow a)\ 1 \rightarrow_\rho a$
   The ρ-term $(((x \twoheadrightarrow x+1) \twoheadrightarrow (1 \twoheadrightarrow x))\ (a \twoheadrightarrow a+1))\ 1$ does not have a corresponding standard rewrite rule of λ-term.

2.1.3 Towards an Abstract Biochemical Calculus

One difference between chemical and biochemical models comes from the usual representation of molecules in a biochemical model as stateful entities which can join to each other in a process called complexation or association [CZ08]. In consequence, in a biochemical model a molecule can be viewed as an abstract object with an internal structure describing, among other features, all possible connections with other molecules.

We extend the chemical model by considering an abstract structure Σ for the molecules and for the computation rules. The structure Σ should permit the modeling of connections between objects as well as the actions of creating and removing such connections. The result is a calculus based on rewriting structured molecules which we call the $\rho_{\langle\Sigma\rangle}$-calculus. The first-class citizens of the $\rho_{\langle\Sigma\rangle}$-calculus are Σ-structured objects, rewrite rules over structured objects, and rule application. This calculus generalizes the λ-calculus, the γ-calculus and the HOCL through a more powerful abstraction power that considers for matching not only a variable or a pattern from a restricted pattern language, but a more generic object built over an algebraic structure and a set of variables. The $\rho_{\langle\Sigma\rangle}$-calculus also encompasses the rewriting calculus [CK01] and the term graph rewriting calculus [BBCK05] by considering the tree-like structure of terms and the graph-like structure of termgraphs respectively as special structures.

The $\rho_{\langle\Sigma\rangle}$-calculus is designed as a formalism for modeling complex systems with dynamical topology where the entities in a state have a particular structure and they interact in a concurrent manner according to a behavior described by rewrite rules. Thanks to the intrinsic parallel nature of rewriting on disjoint redexes and decentralized rule application, we model a kind of *Brownian motion*, a basic principle in the chemical model consisting in *"the free distribution and unspecified reaction order among molecules"* [BRF04], if we consider structured objects as molecules.

By considering the $\rho_{\langle\Sigma\rangle}$-calculus as a modeling framework for a system we gain in expressivity by choosing convenient descriptions of the states, and we dispose of a great flexibility in modeling the system dynamics. For instance, rules can be consumed when applied and new rules can be created by the application of other rules. Then, instead of having a non-deterministic (and possibly non-terminating) behavior of the application of abstractions, one may want to introduce some control to compose or to choose the rules to apply. The notion of abstraction proves to be powerful enough to express such

control, thanks to the notions of *strategy* and *strategic rewriting*. In addition, strategies allow exploiting failure information.

2.1.4 Structure of the Chapter

In this chapter we present the syntax and the semantics of an abstract biochemical calculus. In Section 2.2 we define the syntax of the basic entities of the calculus. We continue in Section 2.3 with the definition of the small-step reduction semantics of the calculus based on three stages: heating, application, and cooling. In Section 2.4 we define strategies as objects of the calculus and prove the correctness of their definitions. Then, we refine the calculus such that we recover from a failing application with the initial interacting strategy and structured object. As an application for strategies, we define persistent strategies which are not consumed by interactions. In Section 2.5 we define the big-step reduction semantics of the calculus such that the failing interactions are invisible in the evolution of the system. We end up with some discussion on other possible strategies to define in the calculus in Section 2.6 and comparisons with the γ-calculus and HOCL in Section 2.7, as well as some conclusions and ideas for future work in Section 2.8.

2.2 Syntax

In the following we describe stepwise all the elements defining the syntax of the calculus as well as the substitution application and the matching.

2.2.1 Structured Objects

We model the states of a system by *structured objects* described by the objects of a particular fixed category Σ. Following the biochemical intuition, the objects are in a Brownian motion like floating in a biochemical soup thus forming together also a structured object. We simulate the Brownian motion by considering a juxtaposition operation on two structured objects which builds also a structured object. Formally, we consider a strict symmetric strict monoidal category (an sssmc) $SSSMC_\Sigma = (\Sigma, \_\ \_, \varepsilon)$ with $\_\ \_ : \Sigma \times \Sigma \to \Sigma$ a bifunctor for juxtaposing two structured objects, and ε the empty object with the condition that the arrows of Σ satisfy the axioms of an sssmc. Let $\mathbb{T}$ denote the set of axioms defining the sssmc and let $\mathcal{O}$ denote $Obj(SSSMC_\Sigma)$. The arrows of $SSSMC_\Sigma$ are the arrows of Σ which satisfy the axioms in $\mathbb{T}$.

Since the bifunctor $\_\ \_$ corresponds to an associative and commutative operator on structured objects, we can see it as a variadic operator and we usually represent it in infix form with no parenthesis.

Example 7. By instantiating the category Σ, we obtain particular structured objects as follows:

Multisets of constants. Consider Σ a small category with $Ob(\Sigma)$ a set of constants. Then, take an arrow, different from the identity, for each pair of distinct objects,

and an identity arrow for each object. Then, for instance, if $a, b, c, d \in Ob(\Sigma)$, then aba, c, $bdbbb$, $cabbacccb$ are examples of multisets. Remark that this structure does not have intrinsic features for expressing connections between objects; hence it is more suitable for a chemical model than a biochemical one.

Multisets of terms. If instead of constants as above we consider algebraic terms over a given first-order signature $\mathcal{F}$ and a set of variables, we obtain multisets of terms. Let $a, b, c, f, g, h \in \mathcal{F}$, with a, b, c constants, f and g unary, and h binary. Then $f(c)\ f(c)\ g(h(a, c))\ h(a, g(f(b)))$ and $g(a)$ are two multisets of terms.

Graphs. We consider the category of graphs **Graph** whose objects are graphs and whose arrows are graph morphisms. Then the bifunctor _ _ corresponds to a disjoint sum of graphs, and ε to the empty graph.

Multisets of term graphs. Based on the algebra describing a graph, we obtain term graphs if we consider in addition an operation on nodes which associate them a fixed outgoing incidence degree (arity), no constraints on the incoming incidence degree, an order on the outgoing incident edges, and a unique node as root. As for graphs, a morphism must preserve all these operations. If the number of incoming incidence degree for each node is maximum 1, then we obtain terms.

We consider a particular of subclass of objects in $\mathcal{O}$ as variables, and we denote it by $\mathcal{X}_{\mathcal{O}}$. The exact definition of variables in a structured object depends on each particular instance of Σ. We denote by $\mathcal{V}ar(O)$ the class of variables occurring in the structure of an object O.

By considering an arbitrary category Σ for describing structured objects, our approach can be considered as a lighter version of the high-level replacement systems [EGPP99] which are formulated for an arbitrary category with a distinguished class of morphisms. The high-level replacement systems were introduced as a generalization for the algebraic approach to graph grammars to different kinds of high-level structures, such as hypergraphs, algebraic specifications, place/transition Petri nets, hierarchical graphs used for statecharts, or jungles used for parallel logic programming [EGPP99, Sch99].

2.2.2 Abstractions

We define a *first-order abstraction* as an ordered pair of two structured objects in $\mathcal{O}$, with the first one called the *left-hand side* and the second one called the *right-hand side*, equipped with a morphism between the two sides. The morphism specifies a transformation of the object in the left-hand side into the object in the right-hand side. We denote by $\mathcal{A}_o$ the set of first-order abstractions:

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

For an abstraction A, we usually denote by L_A and R_A the left- and right-hand side respectively. We represent an abstraction (O_1, f, O_2) as the left-hand side followed by a new operator $\Rightarrow$ not occurring in structured objects of Σ, and the right-hand side, i.e.,

$O_1 \overset{f}{\Rightarrow} O_2$. Unless necessary, we usually do not specify the morphism between the two sides of an abstraction. Hence we can also define $\mathcal{A}_0$ as follows:

$$\mathcal{A}_0 ::= \mathcal{O} \Rightarrow \mathcal{O}$$

Example 8. The first-order abstractions defined on the sssmc constructed for the category of graphs with partial morphisms $\mathbf{Graph^P}$ correspond to the concept of graph transformations formalized using the single-pushout (SPO) approach.

We define a morphism on first-order abstractions $g : (L_A \overset{f}{\Rightarrow} R_A) \to (L_{A'} \overset{f'}{\Rightarrow} R_{A'})$ as a pair of morphisms on structured objects $\langle g_L : L_A \to L_{A'}, g_R : R_A \to R_{A'} \rangle$ such that $g_R \circ f = f' \circ g_L$. Then we can define a category $\mathbf{Abs_0}(\mathcal{O})$ of first-order abstractions over $\mathcal{O}$ with $\mathcal{A}_o$ the class of objects, and morphisms on first-order abstractions as arrows.

The class of abstraction of second-order is defined as:

$$\mathcal{A}_2 = \{(A, g, A') \mid A, A' \in \mathcal{A}_0, g \in \mathbf{Abs_0}(\mathcal{O})(A, A')\}$$

or $\mathcal{A}_o \Rightarrow \mathcal{A}_o$ for short.

2.2.3 Abstract Molecules

The main idea of this calculus is to have all entities at the same level, either objects representing states of the system, or abstractions representing different behaviors of the system, or their interactions. We call such an entity an *abstract molecule*. We define gradually the class of abstract molecules in order to include abstractions whose left-hand sides are structured objects and right-hand side are abstract molecules, as well as abstraction whose both sides are first-order abstraction.

Let $\mathbf{M_0}$ be a category with $Ob(\mathbf{M_0}) = \mathcal{O} \cup \mathcal{A}_0 \cup \mathcal{A}_2$, and $Arr(\mathbf{M_0}) = Arr(SSMC_\Sigma) \cup Arr(\mathbf{Abs_0}(\mathcal{O}))$. Then based on $\mathbf{M_0}$ and using the same bifunctor $\_\ \_$ as before, and ε the empty object, we can construct a first sssmc for abstract molecules $SSMC_{\mathbf{M_0}}$. Let $\mathcal{M}_0$ denote the class of objects of $SSMC_{\mathbf{M_0}}$.

We can now define abstractions with structured objects in the left-hand side and abstract molecules in the right-hand side, where the morphism between the two sides is a morphism between structured objects:

$$\mathcal{A}_1 = \{(O, f, M) \mid O \in \mathcal{O}, M \in Ob(SSMC_{\mathbf{M_0}}), f \in \mathbf{M_0}(O, M)\}$$

For short, we can represent the class of such abstractions as $\mathcal{O} \Rightarrow \mathcal{M}_0$.

Based on these two new classes of abstractions, we consider a new category $\mathbf{M}$ with $Ob(\mathbf{M}) = Ob(\mathbf{M_0}) \cup \mathcal{A}_1$, and $Arr(\mathbf{M_0}) = Arr(SSMC_\Sigma) \cup Arr(\mathbf{Abs_0}(\mathcal{O}))$. Then, similarly to the construction of $SSMC_{\mathbf{M_0}}$, we consider the sssmc associated to $\mathbf{M}$ whose objects are exactly the abstract molecules. Let $\mathcal{M}$ denote the class of the objects of $SSMC_{\mathbf{M}}$.

We remark that, following the procedure above, we can iteratively construct abstractions and molecules of higher-order.

Having the abstract molecules constructed using categorical concepts, below we give their schematic definition which we will use in the rest of the chapter where we no longer make reference to the categorical concepts. Let $\mathcal{X}$ denote the union of $\mathcal{X}_{\mathcal{O}}$ with a class of variables for any kind of abstract molecule. Then the syntax of the object of the calculus is the following:

$$\begin{array}{lcl} \mathcal{A}_0 & ::= & \mathcal{O} \Rightarrow \mathcal{O} \\ \mathcal{A}_2 & ::= & \mathcal{A}_0 \Rightarrow \mathcal{A}_0 \\ \mathcal{M}_0 & ::= & \mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{A}_2 \mid \mathcal{M}_0\ \mathcal{M}_0 \\ \mathcal{A}_1 & ::= & \mathcal{O} \Rightarrow \mathcal{M}_0 \\ \mathcal{M} & ::= & \mathcal{X} \mid \mathcal{M}_0 \mid \mathcal{A}_1 \mid \mathcal{M}\ \mathcal{M} \end{array}$$

An abstraction in $\mathcal{A}_1$ or $\mathcal{A}_2$ has a distinguished *main arrow* operator $\Rightarrow$, the one defining the morphism between the two sides of the abstraction.

As we already said, $\mathbb{T}$ denotes the set of axioms defining the sssmc properties of the structured objects and the morphisms between them. In consequence $\mathbb{T}$ generates a structural congruence relation between the objects in $\mathcal{O}$. Since abstractions and abstract molecules are constructed based on structured objects, it is natural that we extend the structural congruence relation over abstract molecules.

Definition 28 (Structural congruence on abstract molecules)**.** *The* structural congruence relation on abstract molecules *is the smallest congruence relation closed with respect to* $\_\ \_$ *(juxtaposition) on sets of abstract molecules that extends the structural congruence* $=_{\mathbb{T}}$ *on structured objects over* Σ*, hence satisfying the following axioms:*

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

for all $M_1, M_2, M_3 \in \mathcal{M}$*.*

The juxtaposition operator as axiomatized by the theory $\mathbb{T}$ constructs multisets. Equivalently, we may consider that the juxtaposition corresponds to a variadic operator, hence parenthesis are no longer needed. If one wants a different type of collection, new axioms must be added and/or removed from $\mathbb{T}$. For instance if we remove the strict symmetric aspect of the monoidal category (i.e., the commutativity), we obtain lists; whereas if we add an axiom for idempotence, we obtain sets.

2.2.4 Subobjects, Submolecules, Substitutions, Matching

Any structured object O which is neither a variable nor a constant operator from Σ can be decomposed into two structured objects O_1 and O_2 glued together based on a relative position of one with respect to the other. A position is described by a neighborhood information. We denote such a decomposition by $O_1\lfloor O_2 \rfloor_{\mathcal{B}}$ and we call O_1 a *context*, O_2 a *subobject* or *submolecule*, and $\mathcal{B}$ a *neighborhood information*.

An abstraction has also a particular structure given by the structure of its sides and the arrow, hence it can be decomposed into a context, *submolecule*, and neighborhood

information. Let us consider for instance the abstraction $L \Rightarrow R$. If we decompose each side as $L^-\lfloor L_1 \rfloor_{\mathcal{B}'}$ and $R^-\lfloor R_1 \rfloor_{\mathcal{B}''}$, then the abstraction $L \Rightarrow R$ has a decomposition $(L^- \Rightarrow R^-)\lfloor L_1 \Rightarrow R_1 \rfloor_{\mathcal{B}' \Rightarrow \mathcal{B}''}$.

Example 9 (Graphs)**.** Let us consider Σ the category of graphs. We illustrate in Figure 2.4 a decomposition of a first-order abstraction on graphs $L \overset{f}{\Rightarrow} R$ into the submolecule $L_1 \overset{f'}{\Rightarrow} R_1$, the context $L^- \overset{f''}{\Rightarrow} R^-$, and the neighborhood information $\mathcal{B}' \overset{f'''}{\Rightarrow} \mathcal{B}''$ where $\mathcal{B}'$ consists of the edges $(1,4)$ and $(3,4)$, $\mathcal{B}''$ consists of the edge $(4,3)$, and f''' maps the edge $(3,4)$ to the edge $(4,3)$. We remark that f is decomposed into the three morphism f' on the submolecules, f'' on the contexts, and f''' on the neighborhood information.

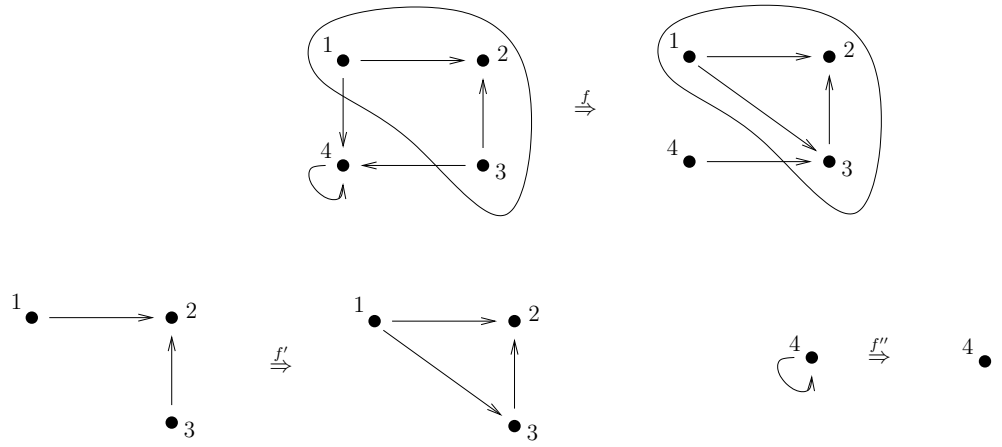


Figure 2.4: A decomposition of a first-order abstraction on graphs

Due to the non-hierarchical structure of graphs, in a decomposition of a graph molecule we can switch the context with the submolecule and obtain the same initial graph molecule as we can see in the example above. In general for a non-hierarchical structure, we have that $O_1\lfloor O_2 \rfloor_{\mathcal{B}}$ is equivalent to $O_2\lfloor O_1 \rfloor_{\overline{\mathcal{B}}}$, with $\overline{\mathcal{B}}$ the "inverse" $\mathcal{B}$. For graphs, we have $\overline{\mathcal{B}} = \mathcal{B}$.

Example 10 (Algebraic terms or term graphs)**.** If Σ describes algebraic terms or term graphs, the objects have a hierarchical structure and we can no longer switch the context with the subobject and get a valid object: the position of a subobject relatively to the context is defined relatively to the root of the term. In addition, since the algebraic operators for terms are not necessarily variadic and the arguments of an operator are ordered, a placeholder is needed for defining a context term. For instance the term $h(a, g(f(b)))$ can be decomposed into the context term $h(a, g(\_))$ with $\_$ the place holder, and $f(b)$ the subobject (or subterm) at the position 2.1 since $f(b)$ is the first argument of g, and $g(\_)$ is the second argument of h. The neighborhood information $\mathcal{B}$ is given by the position of the placeholder in the context term with respect to the root. Consequently, this decomposition of the term $h(a, g(f(b)))$ is denoted by $h(a, g(\_))\lfloor f(b) \rfloor_{2.1}$.

The only type of binder in the $\rho_{\langle\Sigma\rangle}$-calculus is the abstraction. Hence the notions of free and bound variables are similar to those in the ρ-calculus, i.e., in $O \Rightarrow M$ the free variables of O are bound in M.

Definition 29 (Free and bound variables)**.** *Given an abstract molecule M, its set of free variables $FV(M)$ and its set of bound variables $BV(M)$ are defined as follows:*

- *if $M = X$ then $FV(M) = \{X\}$ and $BV(M) = \emptyset$;*
- *if $M = O$ then $FV(M) = \mathcal{V}ar(O)$ and $BV(M) = \emptyset$;*
- *if $M = M_1 \Rightarrow M_2$ then $FV(M) = FV(M_2) \setminus FV(M_1)$ and $BV(M) = BV(M_2) \cup BV(M_1) \cup FV(M_1)$;*
- *if $M = M_1\ M_2$ then $FV(M) = FV(M_1) \cup FV(M_2)$ and $BV(M) = BV(M_1) \cup BV(M_2)$.*

As in any calculus involving binders, we work modulo the α-convention and modulo the hygiene-convention of Barendregt [Bar84], i.e. free and bound variables have different names.

Definition 30 (Substitution)**.** *A* substitution *σ is a mapping from the set of variables $\mathcal{X}$ to the set of abstract molecules $\mathcal{M}$, $\sigma : \mathcal{X} \mapsto \mathcal{M}$. We write it as $\{X_1 \mapsto M_1, \ldots, X_n \mapsto M_n\}$ if there are only finitely many variables not mapped to themselves. It uniquely extends to a homomorphism from $\mathcal{M}$ to $\mathcal{M}$.*

The application of a finite substitution $\sigma = \{X_1 \mapsto M_1, \ldots, X_n \mapsto M_n\}$ on an abstract molecule M, $\sigma(M)$, is obtained by substituting M_i for X_i, $1 \leq i \leq n$, using appropriate variable renaming for avoiding the capture of free variables [CK01].

Given an abstract molecule, we are interested in transforming some of its submolecules which match particular characteristics. Such characteristics describe a *pattern*. The process of identifying its location in the given molecule and mapping it to a subobject is called *submatching*. If the transformation concerns the entire molecule, i.e., the pattern must be identified to the molecule via a substitution, then only a mapping between the elements of the pattern and those of the submolecule has to be provided, and the process is called *matching*.

Definition 31 (Submatching modulo $\mathbb{T}$)**.** *A* submatching equation modulo $\mathbb{T}$ *is an ordered pair of abstract molecules $M \prec\!\!<_{\mathbb{T}} M'$. A* solution *of the submatching equation is a triple $(\sigma, M'', \mathcal{B})$ with σ a substitution, M'' an abstract molecule, and $\mathcal{B}$ the neighborhood information such that $M''\lfloor\sigma(M)\rfloor_{\mathcal{B}} =_{\mathbb{T}} M'$. We call a submatching problem $M \prec\!\!<_{\mathbb{T}} M'$ a* matching problem *if $\sigma(M) =_{\mathbb{T}} M'$ or, equivalently, the context M'' is the empty molecule, and we denote it rather by $M \ll M'$. We usually denote by ς a submatching solution, and by $\mathcal{S}ol(M \prec\!\!<_{\mathbb{T}} M')$* the set of all solutions of the submatching equation $M \prec\!\!<_{\mathbb{T}} M'$.

With every instantiation of the structure Σ of the objects and $\mathbb{T}$, a submatching algorithm modulo $\mathbb{T}$ should also be provided.

Usually, a submatching algorithm will consider as molecules of the same order for a submatching equation, i.e., of the form $O_1 \prec\!\!\!\prec (O_2\ M)$ or $(O_1 \Rightarrow O_2) \prec\!\!\!\prec (O_3 \Rightarrow O_4)$. We introduce this restriction for submatching in order to provide a meaningful transformation for abstractions later.

2.2.5 Worlds

We define the *world* construct by placing abstract molecules in an *environment* where they can interact freely. In other words, a world is an encapsulation of an abstract molecule existing in a state of the modeled system. The environment is "aware" of all abstract molecules it contains. We represent this knowledge by considering a permutative variadic operator [ ] constructing a world which takes as arguments all abstract molecules in the environment. We denote by $[M]$ a world containing the abstract molecule $M \in \mathcal{M}$. If M is a juxtaposition of m variables, n structured objects from $\mathcal{O}$ and p abstractions, then the world $[M]$ is a construct based on $(m+n+p)$ components. The class of worlds $\mathcal{V}$ is defined as follows:

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

where $\mathcal{Y}$ is a set of variables.

An intuitive graphical representation for the environment as a world considers a rectangular box acting as a container, somehow providing an intuition of an environment. We illustrate in Figure 2.5 a world consisting of two structured objects and three abstractions.

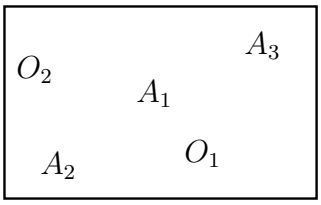


Figure 2.5: Box-based representation of a world consisting of the abstractions A_1, A_2, and A_3, and the structured objects O_1 and O_2.

2.2.6 Structures of Worlds or Multiverses

There may be different ways for a system to evolve from a world (a state) via a successful interaction between two molecules. In order to collect all possible evolution results we introduce a permutative variadic operator { } for building a *structure of worlds* by juxtaposing them. A structure of worlds is also called a *multiverse* (or meta-universe, since a world can also be named a universe). The possible worlds within a multiverse are called *alternative worlds*. Let $\mathcal{W}$ denote the class of multiverses. Then a multiverse consists either of a variable in a set $\mathcal{Z}$, a juxtaposition of worlds, or a juxtaposition of

multiverses.

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

The underlying theory of the operator { } is given by the following structural congruence:

Definition 32 (Structural congruence on worlds)**.** *The* structural congruence relation on multiverses *is the smallest congruence relation satisfying the propriety of* { } *of being variadic and the following flattening axioms:*

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

Overview of the Basic Syntax of the Calculus

In Figure 2.6 we summarize the syntax of the calculus as defined until now which includes structured objects, abstractions, abstract molecules, worlds and multiverses.

(Structured objects)	$\mathcal{O}$		
(First-order abstractions)	$\mathcal{A}_0$	$::=$	$\mathcal{O} \Rightarrow \mathcal{O}$
(Second-order abstractions)	$\mathcal{A}_2$	$::=$	$\mathcal{A}_0 \Rightarrow \mathcal{A}_0$
	$\mathcal{M}_0$	$::=$	$\mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{A}_2 \mid \mathcal{M}_0\ \mathcal{M}_0$
	$\mathcal{A}_1$	$::=$	$\mathcal{O} \Rightarrow \mathcal{M}_0$
(Abstract molecules)	$\mathcal{M}$	$::=$	$\mathcal{X} \mid \mathcal{M}_0 \mid \mathcal{A}_1 \mid \mathcal{M}\ \mathcal{M}$
(Worlds)	$\mathcal{V}$	$::=$	$\mathcal{Y} \mid [\mathcal{M}]$
(Multiverse)	$\mathcal{W}$	$::=$	$\mathcal{Z} \mid \{\mathcal{V} \ldots \mathcal{V}\} \mid \{\mathcal{W} \ldots \mathcal{W}\}$

Figure 2.6: The basic syntax of the $\rho_{\langle\Sigma\rangle}$-calculus

2.3 Small-Step Semantics

2.3.1 Basic Semantics

The semantics of the $\rho_{\langle\Sigma\rangle}$-calculus corresponds to a system whose initial state is a world of the form $[A_1 \ldots A_n\ O_1 \ldots O_m]$, with $n, m \geq 0$. Every abstraction can interact non-deterministically with an abstract molecule in the same world. Such interaction may produce a meaningful result if the matching equation between the left-hand side of the abstraction and the abstract molecule is valid; otherwise the unsuccessful interaction is just expressing the Brownian motion modeled by the permutative juxtaposition operator.

Let A be an abstraction described by $L_A \Rightarrow \mathcal{R}_A$ with ξ the morphism embedded in the arrow operator, and let M be an abstract molecule such that there exists a submatching solution $\varsigma \in \mathcal{S}ol(L_A \prec\!\!\!\prec M)$ with $\varsigma = (\sigma, M', \mathcal{B})$; the existence of a submatching solution ensures the success of the interaction. The interaction between the abstraction A and the abstract molecule M produces to a transformation of M according to A by replacing in M the submolecule $\sigma(L_A)$ by $\sigma(R_A)$ using the neighborhood information $\mathcal{B}$ updated according to ξ. Therefore if M has a decomposition $M'\lfloor\sigma(L_A)\rfloor_{\mathcal{B}}$, then the result of transforming M according to A is the abstract molecule that can be decomposed as $M'\lfloor\sigma(R_A)\rfloor_{\xi(\mathcal{B})}$ also denoted by $\varsigma(M)$.

An interaction between an abstraction and an abstract molecule is described by one of the rules bellow:

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

Figure 2.7: The basic semantics of the $\rho_{\langle\Sigma\rangle}$-calculus

A successful application of an abstraction A on the abstract molecule M, both entities in the same world, reduces to a structure of alternate worlds (or substates) $\{[M_1] \ldots [M_n]\}$ such that $\mathcal{S}ol(L_A \prec\!\!\!\prec M) = \{\varsigma_1, \ldots, \varsigma_n\}$ and $\varsigma_i(R_A) = M_i$. A matching failure means that the interaction of the two entities is not possible; hence the initial abstraction and the structured object are returned unchanged. This approach is similar to the one in AlChemy [FB96], where if two molecules (represented as typed λ-expressions) randomly collide in a reaction and their types are incompatible, then the reaction is considered elastic, i.e., it does not take place.

2.3.2 Making the Application Explicit

The evaluation rules given in Figure 2.7 provide the main idea of the application of an abstraction as a one-step rewriting. We did not specify how the matching and replacement operations, i.e., the effective application of the abstraction, are performed, since these operations, usually defined at the metalevel of the calculus, depend on the structure Σ of objects, a parameter of the calculus.

We refine the semantics of the interactions in order to understand which are the actors and to better control the application process. Let us come back to the situation where the application is not successful (the matching fails): the interaction is elastic and the initial actors are returned. We introduce a second level of internal organization and processing where appropriate tests can be performed. We define an application operator which permits isolating an abstract molecule M and an abstraction A randomly chosen for interaction. Hence, besides allowing different tests to be performed during the application, the application operation also isolates or blocks the access of other abstractions to the two interacting entities. An *application world* is constructed using

the application operator @ which takes as arguments an abstraction and an abstract molecule M. The corresponding box-based representation uses a rectangular box with rounded corners for the application where the abstraction is placed in a rectangular dotted box (see Figure 2.8 for an example).

We extend the syntax of worlds in order to include application worlds as components:

$$\begin{aligned} \mathcal{V}_{app} &::= \mathcal{A}_0@\mathcal{M} \mid \mathcal{A}_1@\mathcal{M} \mid \mathcal{A}_2@\mathcal{M} \\ \mathcal{V} &::= \mathcal{Y} \mid [\mathcal{M}] \mid [\mathcal{M}\ \mathcal{V}_{app}] \end{aligned}$$

Using a similar mechanism as in the CHAM, an interaction takes place in a system after a preparation procedure which restructures the system, a *heating* process. The heating isolates an abstraction A and an abstract molecule M in an application world. This is expressed by the following evaluation rule:

$$(\textbf{Heating}) \qquad h \ : \ [N\ A\ M] \longrightarrow [N\ A@M] \tag{2.1}$$

In the heating rule above we explicit the context $[N\ \_]$ under which the interaction takes place in order to prevent such a reduction to occur under an operator different from the [ ].

Another equivalent representation of this heating rule is given in Figure 2.8 using boxes. Once an abstraction A and an abstract molecule M are non-deterministically chosen, they are placed together in an application box preparing them for effective application.

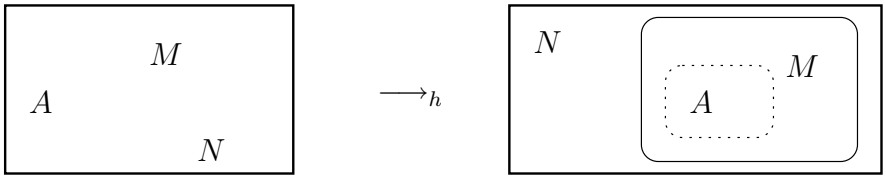


Figure 2.8: Box-based representation of the application of the heating rule 2.1 on a world consisting of an abstraction A, and two abstract molecules M and N

Then the steps computing the application of the isolated abstraction A on the abstract molecule M, including the matching and the substitution application (replacement) operations, are expressed by the following application evaluation rules:

$$(\textbf{Application}) \qquad a \ : \ A@M \longrightarrow \{[\varsigma_1(R_A)] \ldots [\varsigma_n(R_A)]\} \quad \textbf{if} \ \ \mathcal{S}ol(L_A \prec\!\!\!\prec M) = \{\varsigma_1, \ldots, \varsigma_n\} \tag{2.2}$$

$$(\textbf{ApplicationFail}) \qquad af \ : \ A@M \longrightarrow A\ M \ \ \textbf{if} \ \ \mathcal{S}ol(L_A \prec\!\!\!\prec M) = \emptyset \tag{2.3}$$

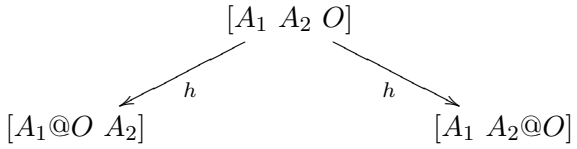
When the application of an abstraction on a structured object successfully takes place in the current state of the system, a *cooling* rule, the counterpart of the heating rule, is in

charge of rebuilding the state of the system by plugging the result of the rewriting in the environment. The context consisting of molecules not participating at the interaction is replicated for each result by the following evaluation rule:

$$(\textbf{Cooling}) \qquad c \ : \ [N \ \{[M_1] \dots [M_n]\}] \longrightarrow \{[N \ M_1] \dots [N \ M_n]\} \tag{2.4}$$

2.3.3 On the Local Confluence

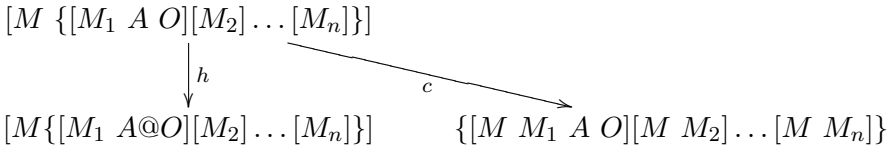
The chemical model is highly non-deterministic: on one hand, due to the intrinsic non-deterministic interaction strategy for choosing the abstract molecules to heat up, and, on the other hand, due to the choice of the submatching solution. By considering a structure of all possible results of an application we have eliminated the non-determinism provided by multiple solutions for submatching. But still, the remaining degree of non-determinism makes the calculus non locally confluent. Let us see an example:



The world consisting of two abstractions A_1 and A_2 and a structured object O is heated up giving rise to two possibilities corresponding to each of the two abstractions to be applied on O. In the particular case when the critical pair raised by the two abstractions is joinable, the reduction converges, but, in general, it does not converge. Therefore the rewriting relation induced by the reduction rules giving the basic semantics of the abstract biochemical calculus is not locally confluent. In such conditions, usually the confluence can be enforced by using appropriate evaluation strategies. In Section 2.4 we provide means of adding strategies to the calculus in order to control the interactions between abstractions and structured objects.

2.3.4 First Cool Down, then Heat Up

Let us analyze in the following the conditions for heating and cooling a world and the importance of their succession in time. The heating and cooling reduction rules give rise to the following critical pair:



If we consider that $A@O \longrightarrow_a \{[N_1] \dots [N_k]\}$, the critical pair above is joinable since on one hand:

$$[M \ \{[M_1 \ A@O][M_2] \dots [M_n]\}] \longrightarrow_a$$

$$
\begin{array}{l}
[M\ \{[M_1\ \{[N_1]\dots[N_k]\}][M_2]\dots[M_n]\}] \longrightarrow_c \\
[M\ \{\{[M_1\ N_1]\dots[M_1\ N_k]\}[M_2]\dots[M_n]\}] \equiv \\
[M\ \{[M_1\ N_1]\dots[M_1\ N_k][M_2]\dots[M_n]\}] \longrightarrow_c \\
\{[M\ M_1\ N_1]\dots[M\ M_1\ N_k][M\ M_2]\dots[M\ M_n]\}
\end{array}
$$

and on the other hand:

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

If $A@O \longrightarrow_{af} A\ O$ we still obtain joinability. This result assures us that *the order of reduction using heating or cooling is not important*. In order to follow a biochemical intuition as well as for the sake of simplicity, we decide then to give a higher priority of application to the cooling evaluation rule. Therefore a world is heated up only after cooling down the multiverse it belongs to.

Overview of the Semantics of the Calculus with Explicit Application

In Figure 2.9 we group all evaluation rules of the calculus where the application between an abstraction and an abstract molecule is made explicit.

(Heating)	h	:	$[N\ A\ M] \longrightarrow [N\ A@M]$
(Application)	a	:	$A@M \longrightarrow \{[\varsigma_1(R_A)]\dots[\varsigma_n(R_A)]\}$ **if** $\mathcal{S}ol(L_A \prec\!\!\!\prec M) = \{\varsigma_1, \dots, \varsigma_n\}$
(Application Fail)	af	:	$A@M \longrightarrow A\ M$ **if** $\mathcal{S}ol(L_A \prec\!\!\!\prec M) = \emptyset$
(Cooling)	c	:	$[N\ \{[M_1]\dots[M_n]\}] \longrightarrow \{[N\ M_1]\dots[N\ M_n]\}$

Figure 2.9: The semantics of the $\rho_{\langle\Sigma\rangle}$-calculus with explicit application

2.4 Adding Strategies to the Calculus

The intrinsic parallelism of the rewriting on disjoint redexes using the abstractions available in a world can be changed by considering a control mechanism or an order of application over the abstractions. Rewrite strategies provide such control in a rule-based framework. In this section, we define strategies as objects of the calculus, using the basic constructs, as one can do in the λ-calculus or the γ-calculus. For such definition we use

a similar approach to the one used in [CKLW03] where rewrite strategies are encoded by rewrite rules. The definitions of the strategy objects are given together with the proofs of their correctness with respect to the semantics of the abstract strategies they encode. Then, thanks to strategies, new extensions for the calculus are possible, for instance to catch a failure in the application or to define persistent strategies.

2.4.1 Strategies as Abstractions

Let us consider for the rewrite strategies given in Sect. 1.7 the following objects (aliases): id for Id, $\mathtt{fail}$ for $Fail$, $\mathtt{first}$ for $First$, $\mathtt{seq}$ for $\_;\_$, $\mathtt{not}$ for Not, $\mathtt{ifThenElse}$ for *IfThenElse*, $\mathtt{try}$ for Try, $\mathtt{repeat}$ for *Repeat*. The application operator for strategies $[\_](\_)$ from Sect. 1.7 corresponds to the construct @.

In the following, when we want to emphasize the use of strategies as abstractions, we replace the symbol for abstraction by the symbol for strategy S in any evaluation rule.

We first extend the syntax of the abstract molecules with the failure object, stk. This failure object stk is the result of a failing application of an abstraction to an abstract molecule:

$$(\textbf{ApplicationFail}) \qquad af \ : \ A@M \longrightarrow \{[\mathsf{stk}]\} \ \ \textbf{if} \ \ \mathcal{S}ol(L_A \prec\!\!\!\prec M) = \emptyset \tag{2.5}$$

We call an *extended abstraction* or *strategy* an abstraction whose right-hand side may contain the application construct.

Let S, S_1, S_2 denote strategies. We encode the strategies Id, $Fail$, $\_;\_$, and $First$ as the following aliases for extended abstractions respectively:

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

The failure of the application of an abstract strategy as defined in Section 1.7 is represented by the empty set. However, we define the failure of the application of a strategy in the $\rho_{\langle\Sigma\rangle}$-calculus not by an empty set but by the singleton containing the failure object stk, i.e., a structure with a world $\{[\mathsf{stk}]\}$. Explicit failure provides more expressivity to the strategy objects in the $\rho_{\langle\Sigma\rangle}$-calculus. By considering a strategy with the failure object as the left-hand side, we can catch and tackle appropriately the failure of the application of another strategy.

In Figures 2.10 and 2.11 we give the representation using boxes of the strategies $\mathtt{seq}$ and $\mathtt{first}$.

The objects corresponding to the two composed abstract strategies Try and *Repeat* are then easily defined based on the strategies $\mathtt{first}$, $\mathtt{try}$, and $\mathtt{seq}$, as already seen in

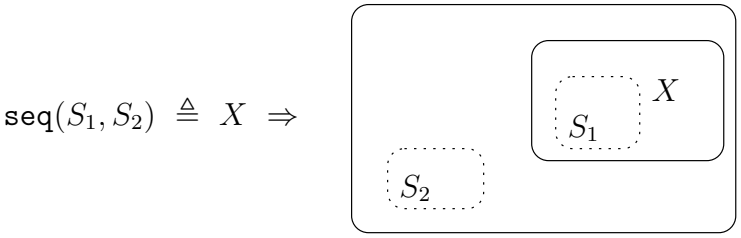


Figure 2.10: Box-based representation of the strategy `seq`

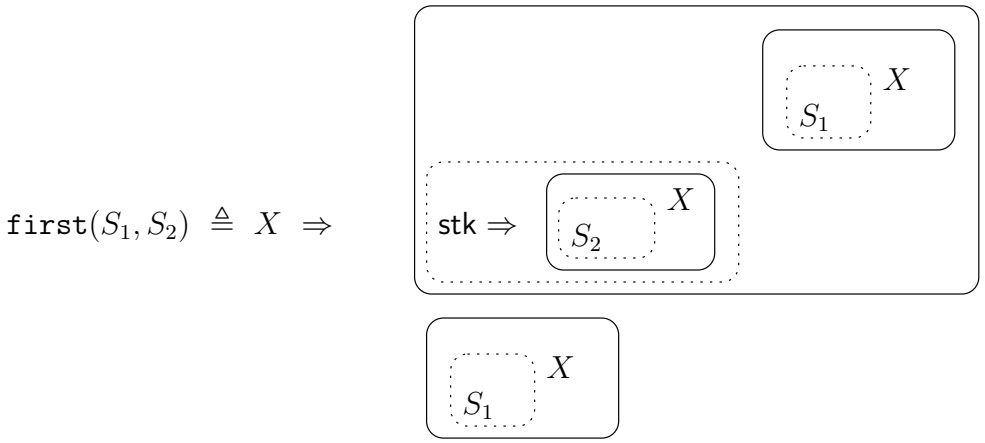


Figure 2.11: Box-based representation of the strategy `first`

the general case of abstract strategies in Section 1.7:

$$\begin{aligned} \texttt{try}(S) &\triangleq \texttt{first}(S, \mathsf{id}) \\ \texttt{repeat}(S) &\triangleq \texttt{try}(\texttt{seq}(S, \texttt{repeat}(S))) \end{aligned}$$

Since we consider all possible results of an application and we can apply a strategy on an application world as it is the case for the definition of `seq`, we need an evaluation rule that distributes the application of a strategy to each world in a structure:

$$(\textbf{AppCooling}) \qquad ac \ : \ S@\{[M_1] \dots [M_n]\} \longrightarrow \{[S@M_1] \dots [S@M_n]\} \tag{2.6}$$

The name of the rule comes from the fact that it has the characteristics of a cooling rule by plugging into the context (here, an application world) the elements of a structure of worlds.

A second rule is needed for flattening a world consisting only of a multiverse:

$$(\textbf{Flattening}) \qquad f \ : \ \{[\{[M_1] \dots [M_n]\}]\} \longrightarrow \{[M_1] \dots [M_n]\} \tag{2.7}$$

If a failure occurs during the application process, we handle it explicitly. We remove a world containing the failing object if it is the only world in a multiverse juxtaposed to

another multiverse (rule (2.8)), or if it is juxtaposed to another world (rule (2.9)), hence behaving as a neutral element for the juxtaposition of multiverses and worlds:

$$\mathsf{stk}_1 \ : \ \{[\mathsf{stk}]\}\ W' \longrightarrow W' \tag{2.8}$$

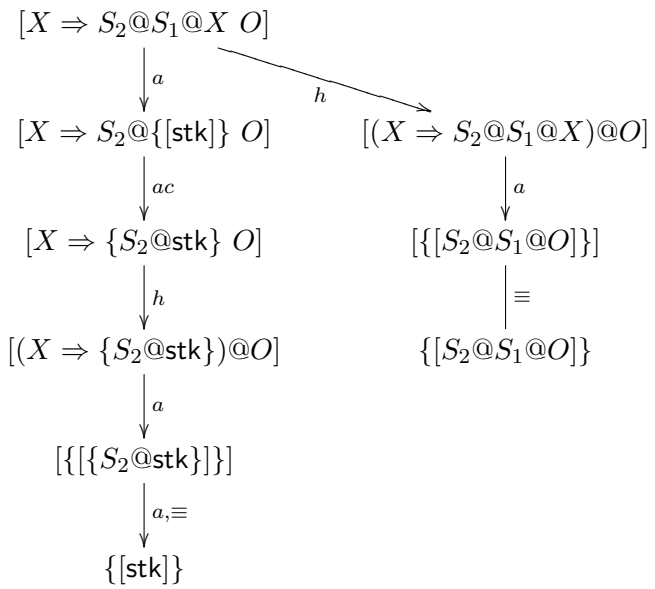
$$\mathsf{stk}_2 \ : \ [\mathsf{stk}]\ V' \longrightarrow V' \tag{2.9}$$

For instance, the first stk evaluation rule can be used to reduce the application of a strategy $\mathtt{first}(S_1, S_2)$ on an object O when at least one of the molecules $S_1@O$ and $(\mathsf{stk} \Rightarrow (S_2@X))@(S_1@X)$ reduces to the failure object stk.

2.4.2 Call-by-Name in the Calculus with Strategies

We consider a call-by-name evaluation strategy for the calculus which prevent any reduction inside an abstraction. In the following we motivate the choice of such an evaluation strategy.

We saw that the definition of a strategy may contain application worlds. Let us see what happens if an evaluation rule (either (2.2), (2.5), or (2.6)) is applied inside an abstraction. Let us consider for instance the world $[X \Rightarrow S_2@S_1@X\ O]$. Then the heating rule and the application rule overlap as follows:



If in particular the left-hand sides of S_1 and S_2 are not variables nor the failure constant stk, and the sequential application of S_1 followed by S_2 on O succeeds, then the right-branch reduces to something different from $\{[\mathsf{stk}]\}$.

The left branching leads to an unwanted result obtained by reducing too early in the right-hand side of the abstraction. Another unwanted branching is eliminated from the beginning thanks to typing the abstract molecules: since $S_1@X$ is not a structured object, the application rule cannot be used for reducing $S_2@S_1@X$. In conclusion, for

an expression $S_2@S_1@M$ the priority of the application operator @ goes from right to left, that is we parenthesize $S_2@S_1@M$ as $S_2@(S_1@M)$.

2.4.3 Correctness of the Encoding of Strategies as Abstractions

We state that our encoding of rewrite strategies as abstractions in the $\rho_{\langle\Sigma\rangle}$-calculus is correct with respect to the semantics of abstract strategies as given in Section 1.7. For ζ a rewrite strategy and S its encoding in the calculus, we say that the encoding is correct if it preserves the solutions. The correctness of the encoding depends on the correctness of the definition of reducing an abstraction and an abstract molecule with respect to the rewriting relation defined on abstract molecules.

Proposition 1 (Correctness of id and $\mathtt{fail}$)**.** The objects id and $\mathtt{fail}$ are correct encodings of the abstract strategies Id and $Fail$ respectively.

Proof. Let M be an abstract molecule. Firstly, on one side $[Id](M)$ reduces to the a singleton consisting of the element M, whereas, on the other side, $\mathsf{id}@M \triangleq (X \Rightarrow X)@M \longrightarrow_a \{[M]\}$; hence, equal results modulo some syntactic notations.

Secondly, on one side, the application of the abstract strategy $Fail$ on M reduces to the empty set. On the other side $\mathtt{fail}@M \triangleq (X \Rightarrow \mathsf{stk})@M \longrightarrow_a \{[\mathsf{stk}]\}$ and $\{[\mathsf{stk}]\}$ has the same properties as the empty set with respect to the set union operation, i.e., neutral element for juxtaposition as defined by the rules (2.8) and (2.9). □

Proposition 2 (Correctness of $\mathtt{seq}$)**.** If the extended abstractions S_1 and S_2 are correct encodings of the abstract strategies ζ_1 and ζ_2 respectively, then $\mathtt{seq}(S_1, S_2)$ is a correct encoding of the abstract strategy $Sequence(\zeta_1, \zeta_2)$.

Proof. Let M be an abstract molecule. From the hypothesis, if the application of the strategy ζ_1 on M is successful, i.e. $[\zeta_1](M) = \{M_1, \ldots, M_n\}$, $n \geq 1$, then we have the reduction $S_1@M \longrightarrow^* \{[M_1] \ldots [M_n]\}$.

Let $[\zeta_2](M_i) = \{M_{i1}, \ldots, M_{ik_i}\}$, for $1 \leq i \leq p \leq n$, and $[\zeta_2](M_j) = \emptyset$, for $p < j \leq n$. It follows that:

$$S_2@M_i \longrightarrow^* \{[M_{i1}] \ldots [M_{ik_i}]\} \text{ for } 1 \leq i \leq p,$$

and

$$S_2@M_j \longrightarrow^* \{[\mathsf{stk}]\} \text{ for } p < j \leq n.$$

The reduction corresponding to the application of the strategy $\mathtt{seq}(S_1, S_2)$ on M is the following:

$$\begin{aligned}
&\mathtt{seq}(S_1, S_2)@M \triangleq (X \Rightarrow S_2@(S_1@X))@M \longrightarrow_a \\
&\{[S_2@(S_1@M)]\} \longrightarrow^* \\
&\{[S_2@\{[M_1] \ldots [M_n]\}]\} \longrightarrow_{ac} \\
&\{[\{[S_2@M_1] \ldots [S_2@M_n]\}]\} \longrightarrow^* \\
&\{[\{\{[M_{11}] \ldots [M_{1k_1}]\} \ldots \{[M_{p1}] \ldots [M_{pk_p}]\}\ \{[\mathsf{stk}]\} \ldots \{[\mathsf{stk}]\}\}]\} \equiv \\
&\{[\{[M_{11}] \ldots [M_{1k_1}] \ldots [M_{p1}] \ldots [M_{pk_p}]\ [\mathsf{stk}] \ldots [\mathsf{stk}]\}]\} \longrightarrow^*_{\mathsf{stk}_1, \mathsf{stk}_2} \\
&\{[\{[M_{11}] \ldots [M_{1k_1}] \ldots [M_{p1}] \ldots [M_{pk_p}]\}]\} \longrightarrow_f
\end{aligned}$$

$$\{[M_{11}]\dots[M_{1k_1}]\dots[M_{p1}]\dots[M_{pk_p}]\}$$

We obtain the same result as for applying the strategy $\zeta_1;\zeta_2$ to M since $[\zeta_1;\zeta_2](M) = \bigcup_{l=1,n} \zeta_2(M_l) = \bigcup_{i=1,p} \zeta_2(M_i)$ and $\zeta_2(M_j) = \emptyset$ for all j, $p < j \leq n$.

If $[\zeta_2](M_i) = \emptyset$ for all i, $1 \leq i \leq n$, then $[\zeta_1;\zeta_2](M) = \emptyset$. Following the same reasoning as above, $S_2@M_i \longrightarrow^* \{[\mathsf{stk}]\}$ for all i, $1 \leq i \leq n$, and therefore $\mathtt{seq}(S_1, S_2)@M$ reduces to the failure $\{[\mathsf{stk}]\}$.

Let now consider the case when the application of the first strategy fails, $[\zeta_1](M) = \emptyset$. By definition, $[\zeta_1;\zeta_2](M) = \emptyset$. From the hypothesis, the failure of the first strategy corresponds to the reduction $S_1@M \longrightarrow^* \{[\mathsf{stk}]\}$. Since S_2 is an encoding of the strategy ζ_2, the left-hand side of S_2 is different from stk, then

$$S_2@\{[\mathsf{stk}]\} \longrightarrow_{ac} \{S_2@\mathsf{stk}\} \longrightarrow^* \{\{[\mathsf{stk}]\}\} \equiv \{[\mathsf{stk}]\}$$

□

Proposition 3 (Correctness of `first`)**.** If the extended abstractions S_1 and S_2 are correct encodings of the abstract strategies ζ_1 and ζ_2 respectively, then $\mathtt{first}(S_1, S_2)$ is a correct encoding of the abstract strategy $\mathit{First}(\zeta_1, \zeta_2)$.

Proof. Let M be an abstract molecule. From the hypothesis, if the application of the strategy ζ_1 on M is successful, i.e. $[\zeta_1](M) = \{M_1, \dots, M_n\}$, $n \geq 1$, then we have

$$S_1@M \longrightarrow^* \{[M_1]\dots[M_n]\}$$

and

$$\begin{array}{l}(\mathsf{stk} \Rightarrow (S_2@M))@(S_1@M) \longrightarrow^* \\ (\mathsf{stk} \Rightarrow (S_2@M))@\{[M_1]\dots[M_n]\} \longrightarrow_{ac} \\ \{[(\mathsf{stk} \Rightarrow (S_2@X))@M_1]\dots[(\mathsf{stk} \Rightarrow (S_2@X))@M_n]\} \longrightarrow^* \\ \{\{[\mathsf{stk}]\}\dots\{[\mathsf{stk}]\}\} \equiv \{[\mathsf{stk}]\dots[\mathsf{stk}]\} \equiv \{[\mathsf{stk}]\}\end{array}$$

On one hand, the application of the strategy $\mathit{First}(\zeta_1, \zeta_2)$ on M produces the result of the application of ζ_1 on O since it does not fail. On the other hand, the reduction corresponding to the application of the strategy $\mathtt{first}(S_1, S_2)$ on M is the following:

$$\begin{array}{l}\mathtt{first}(S_1, S_2)@M \triangleq (X \Rightarrow (S_1@X)\ (\mathsf{stk} \Rightarrow (S_2@X))@(S_1@X))@M \longrightarrow_a \\ \{[(S_1@M)\ (\mathsf{stk} \Rightarrow (S_2@M))@(S_1@M)]\} \longrightarrow^* \\ \{[\{[M_1]\dots[M_n]\}\ \{[\mathsf{stk}]\}]\} \longrightarrow_{\mathsf{stk}_1} \\ \{[\{[M_1]\dots[M_n]\}]\} \longrightarrow_f \\ \{[M_1]\dots[M_n]\}\end{array}$$

Therefore the encoding has the same behavior when the application of the first strategy does not fail on M.

Now, if $[\zeta_1](M) = \emptyset$, hence $S_1@M \longrightarrow^* \{[\mathsf{stk}]\}$. Then

$$\begin{aligned}&(\mathsf{stk} \Rightarrow (S_2@M))@(S_1@M)\\ &\longrightarrow^* (\mathsf{stk} \Rightarrow (S_2@M))@\{[\mathsf{stk}]\} \longrightarrow_{ac}\\ &\{[(\mathsf{stk} \Rightarrow (S_2@M))@\mathsf{stk}]\} \longrightarrow_{a}\\ &\{\{[S_2@M]\}\} \equiv \{[S_2@M]\}\end{aligned}$$

On one hand we have that $[First(\zeta_1, \zeta_2)](M) = [\zeta_2](M)$, and on the other hand the application of its encoding reduces to the same result as follows:

$$\begin{aligned}&\mathtt{first}(S_1, S_2)@M \triangleq (X \Rightarrow (S_1@X)\ (\mathsf{stk} \Rightarrow (S_2@X))@(S_1@X))@M \longrightarrow_a\\ &\{[(S_1@M)\ (\mathsf{stk} \Rightarrow (S_2@M))@(S_1@M)]\} \longrightarrow^*\\ &\{[\{[\mathsf{stk}]\}\ \{[S_2@M]\}]\} \longrightarrow_{\mathsf{stk}_1}\\ &\{[\{[S_2@M]\}]\} \longrightarrow_f\\ &\{[S_2@M]\}\end{aligned}$$

□

Proposition 4 (Correctness of `not`)**.** If the extended abstraction S is a correct encoding of the abstract strategy ζ, then $\mathtt{not}(S)$ is a correct encoding of the abstract strategy $Not(\zeta)$.

Proof. Let M be an abstract molecule. From the hypothesis, if the application of the strategy ζ on M is successful, i.e. $[\zeta](M) = \{M_1, \ldots, M_n\}$, $n \geq 1$, then we have:

$$S@M \longrightarrow^* \{[M_1] \ldots [M_n]\}$$

Then on one side $[Not(\zeta)](M) = \emptyset$, and on the other side:

$$\mathtt{not}(S)@M \longrightarrow^* \{[\mathtt{first}(\mathsf{stk} \Rightarrow M, X' \Rightarrow \mathsf{stk})@(S@M)]\} \longrightarrow^* \{[\mathsf{stk}]\}$$

But if the application of the strategy ζ on M fails, then $[\zeta](M) = \emptyset$, and $[Not(\zeta)](M) = M$; whereas, $S@M \longrightarrow^* \mathsf{stk}$ and $\mathtt{not}(S)@M \longrightarrow^* \{[M]\}$. □

Proposition 5 (Correctness of `ifThenElse`)**.** If the extended abstractions S_1, S_2, S_3 are correct encodings of the abstract strategies ζ_1, ζ_2, ζ_3 respectively, then $\mathtt{ifThenElse}(S_1, S_2, S_3)$ is a correct encoding of the abstract strategy $IfThenElse(\zeta_1, \zeta_2, \zeta_3)$.

Proof. Let M be an abstract molecule. From the hypothesis, if the application of the strategy ζ_1 on M is successful, then $S@M$ reduces to something different from $\{[\mathsf{stk}]\}$. Then on one side $[IfThenElse(\zeta_1, \zeta_2, \zeta_3)](M) = [\zeta_2](M)$, and on the other side:

$$\begin{aligned}&\mathtt{ifThenElse}(S_1, S_2, S_3)@M \longrightarrow^*\\ &\{[\mathtt{first}(\mathsf{stk} \Rightarrow S_3@M, X' \Rightarrow S_2@M)@(S@M)]\} \longrightarrow^* \{[S_2@M]\}\end{aligned}$$

Now, if the application of the strategy ζ_1 on M fails, then $[IfThenElse(\zeta_1, \zeta_2, \zeta_3)](M) = [\zeta_3](M)$; whereas, $S_1@M \longrightarrow^* \mathsf{stk}$ and $\mathtt{ifThenElse}(S_1, S_2, S_3)@M \longrightarrow^* \{[S_3@M]\}$. □

Remark 1. If S is a correct encoding of an abstract strategy ζ, then $\texttt{try}(S)$ and $\texttt{repeat}(S)$ are correct encodings of $\mathit{Try}(\zeta)$ and $\mathit{Repeat}(\zeta)$ respectively.

The following theorem summarizes the previous results on the correctness of encoding the strategies.

Theorem 3. *For every strategy object S obtained by combining the primitive strategy objects* id, `fail`, `seq`, `first`, `not`, `ifThenElse` *and for every abstract molecule M, a successful computation of $S@M$ yields a structure of worlds $\{[M_1] \dots [M_n]\}$ such that each abstract molecule M_i is obtained from M by strategic rewriting under the strategy S.*

2.4.4 Extending the Semantics with Strategies and Failure Recovery

Based on the strategy definitions, we can reformulate the main reduction rule modeling the interaction between a strategy S and an abstract molecule M in a world using a failure catching mechanism:

$$(\textbf{HeatingR}) \quad hr \ : \ [N \ S \ M] \longrightarrow [N \ \texttt{seq}(S, \texttt{try}(\mathsf{stk} \Rightarrow S \ M))@M] \tag{2.10}$$

A reduction using the rule (2.10) proceeds in one of the following ways:

- if $S@M$ reduces to the failure construct stk, then the strategy $\texttt{try}(\mathsf{stk} \Rightarrow S \ M)$ restores the initial molecules subject to reduction;
- if $S@M$ succeeds, then the application of the strategy $\texttt{try}(\mathsf{stk} \Rightarrow S \ M)$ does not change the result.

We call this improved reduction *heating with recovery* and $\mathsf{stk} \Rightarrow S \ M$ a *recovery rule*. The two molecules S and M from the right-hand side of the recovery rule cannot be heated before the rule is applied since they are arguments of the arrow construct and not juxtaposed in a world. Hence there is no risk of recursive reduction using this heating rule.

The behavior presented above for the heating with recovery rule is possible under the assumption that the strategies in a world should not return the failure object stk.

Remark 2. The heating with recovery rule (2.10) is just an example of tackling a matching failure occurred in the application process. Instead of just recovering the interacting abstraction S and abstract molecule M, one can imagine other ways of tackling a failure in the interaction, for instance by modifying M or even S.

2.4.5 Persistent Strategies

At this level of definition of the calculus, a strategy is consumed by a non-failing interaction with an abstract molecule M. One advantage is that, since we work with multisets, a strategy can be given a multiplicity, and each interaction between the strategy and an abstract molecule M consumes one occurrence of the strategy. This permits controlling the maximum number of times an interaction can take place.

Sometimes it may be suitable to have persistence of strategies. In this case, the strategy should not be consumed by the reduction. For this purpose, we define the *persistent* strategy combinator that applies a strategy given as argument and, if successful, replicates itself:

$$
\begin{array}{rcl}
S! & \triangleq & \texttt{seq}(S, \\
 & & \quad \texttt{first}(\mathsf{stk} \Rightarrow \mathsf{stk}, \\
 & & \quad\quad X \Rightarrow (X\ S!) \\
 & & \quad\quad) \\
 & &)
\end{array}
$$

However, for instance $(O \Rightarrow \mathsf{stk})@O'$ reduces to $\{[\mathsf{stk}]\}$ if O matches O'. The abstraction is applied successfully but the result is saying that the application failed. The contradictory situation comes from the fact that we allow the use of the application failure information as an explicit object. This example shows that the great expressive power we gained by making the failure an explicit object can lead to an unexpected behavior if not handled with caution. As a consequence we restrict the use of persistence such that any strategy that produces stk cannot be made persistent. Such strategies are the ones built using abstractions with stk as right-hand side.

The following result proves that the persistent strategy combinator is correctly defined under the previously given condition.

Proposition 6. Let M be an abstract molecule and S a strategy such that any abstraction occurring in S does not have stk as right-hand side. If $S@M \longrightarrow^* \{[\mathsf{stk}]\}$, then $S!\ M \longrightarrow_{hr} \longrightarrow^* \{[S!\ M]\}$, else, if $S@M \longrightarrow^* \{[M_1] \dots [M_n]\}$, $n \geq 1$, i.e., the application does not fail, then $S!\ M \longrightarrow_{hr} \longrightarrow^* \{[S!\ M_1] \dots [S!\ M_n]\}$.

Proof. Let S_2 denote the strategy $\texttt{try}(\mathsf{stk} \Rightarrow (S!\ M))$ and S_3 the strategy $\texttt{first}(\mathsf{stk} \Rightarrow \mathsf{stk}, X \Rightarrow X\ S!)$. Then we have the reduction:

$$
\begin{array}{l}
S!\ M \longrightarrow_{hr} \texttt{seq}(S!, \texttt{try}(\mathsf{stk} \Rightarrow (S!\ M)))@M = \texttt{seq}(S!, S_2)@M = \{[S_2@(S!@M)]\} = \\
\{[S_2@(\texttt{seq}(S, S_3)@M)]\} \longrightarrow^* \{[S_2@(\{@S_3@(S@M)\})]\} \longrightarrow_{ac} \\
\{[\{S_2@(S_3@(S@M)\}]\} \longrightarrow_f \{S_2@(S_3@(S@M)\}
\end{array}
$$

If $S@M \longrightarrow^* \{[\mathsf{stk}]\}$, then:

$$S_3@(S@M) \longrightarrow^* S_3@\{[\mathsf{stk}]\} \longrightarrow_d \{S_3@\mathsf{stk}\} \longrightarrow^* \{\{[\mathsf{stk}]\}\} \equiv \{[\mathsf{stk}]\}$$

and

$$
\begin{array}{l}
S!\ M \longrightarrow_{hr} \longrightarrow^* \{S_2@(S_3@(S@M))\} \longrightarrow^* \{S_2@\{[\mathsf{stk}]\}\} \longrightarrow_{ac} \\
\{\{S_2@\mathsf{stk}\}\} \equiv \{S_2@\mathsf{stk}\} \longrightarrow^* \{\{[S!\ M]\}\} \equiv \{[S!\ M]\}
\end{array}
$$

Else, if $S@M \longrightarrow^* \{[M_1] \dots [M_n]\}$, with $n \geq 1$, then:

$$
\begin{array}{l}
S_3@(S@M) \longrightarrow^* S_3@\{[M_1] \dots [M_n]\} \longrightarrow_d \{S_3@M_1 \dots S_3@M_n\} \longrightarrow^* \\
\{\{[S!\ M_1]\} \dots \{[S!\ M_n]\}\} \equiv \{[S!\ M_1] \dots [S!\ M_n]\}
\end{array}
$$

and

$$
\begin{aligned}
&S!\ M \longrightarrow_{hr} \longrightarrow^* \{[\{S_2@(S_3@(S@M))\}]\} \longrightarrow^* \\
&\{[\{S_2@\{[S!\ M_1] \ldots [S!\ M_n]\}\}]\} \longrightarrow_{ac} \\
&\{[\{\{S_2@(S!\ M_1) \ldots S_2@(S!\ M_n)\}\}]\} \longrightarrow^* \\
&\{[\{\{\{[S!\ M_1]\} \ldots \{[S!\ M_n]\}\}\}]\} \equiv \\
&\{[\{[S!\ M_1] \ldots [S!\ M_n]\}]\} \longrightarrow_f \{[S!\ M_1] \ldots [S!\ M_n]\}
\end{aligned}
$$

□

Remark 3. If we consider that a successful application of a strategy to an abstract molecule does not consume the strategy, then the strategies are persistent by definition and we no longer need to define the persistent combinator for strategies. However, the current approach gives us the freedom of providing some strategies with a persistent behavior, while others exist in a limited number of occurrences, hence they are consumed by applications.

2.4.6 Overview of the Syntax and the Semantics of the Calculus with Strategies

In Figure 2.12 we give the syntax of the calculus with strategies.

(Structured objects)	$\mathcal{O}$		
(First-order abstractions)	$\mathcal{A}_0$	::=	$\mathcal{O} \Rightarrow \mathcal{O}$
(Second-order abstractions)	$\mathcal{A}_2$	::=	$\mathcal{A}_0 \Rightarrow \mathcal{A}_0$
	$\mathcal{M}_0$	::=	$\mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{A}_2 \mid \mathcal{M}_0\ \mathcal{M}_0$
	$\mathcal{A}_1$	::=	$\mathcal{O} \Rightarrow \mathcal{M}_0$
(Abstract molecules)	$\mathcal{M}$	::=	$\mathcal{X} \mid \mathcal{M}_0 \mid \mathcal{S} \mid \mathcal{M}\ \mathcal{M}$
(Strategies	$\mathcal{S}$	::=	$\mathcal{A}_0 \mid \mathcal{A}_1 \mid \mathcal{A}_2$
or extended abstractions)		|	$\mathsf{stk} \Rightarrow \mathsf{stk} \mid \mathsf{stk} \Rightarrow \mathcal{M}$
		|	$\mathcal{X} \Rightarrow \mathsf{stk} \mid \mathcal{X} \Rightarrow \mathcal{M}$
		|	$\mathcal{X} \Rightarrow \mathcal{V}_{app} \mid \mathcal{X} \Rightarrow \mathcal{V}_{app}\ \mathcal{V}_{app}$
(Application worlds)	$\mathcal{V}_{app}$	::=	$\mathcal{S}@\mathcal{M} \mid \mathcal{S}@\mathcal{V}_{app} \mid \mathcal{S}@\mathcal{W}$
(Worlds)	$\mathcal{V}$	::=	$\mathcal{Y} \mid [\mathcal{M}] \mid [\mathcal{M}\ \mathcal{V}_{app}] \mid [\mathcal{M}\ \mathcal{W}]$
(Multiverse)	$\mathcal{W}$	::=	$\mathcal{Z} \mid \{\mathcal{V} \ldots \mathcal{V}\} \mid \{\mathcal{W} \ldots \mathcal{W}\}$

Figure 2.12: The syntax of the $\rho_{\langle\Sigma\rangle}$-calculus with strategies

In Figure 2.13 we review the evaluation rules of the calculus where an abstraction is generalized to a strategy.

(HeatingR)	hr	:	$[N\ S\ M] \longrightarrow [N\ \mathtt{seq}(S, \mathtt{try}(\mathsf{stk} \Rightarrow S\ M))@M]$
(Application)	a	:	$S@M \longrightarrow \{[\varsigma_1(R_S)] \ldots [\varsigma_n(R_S)]\}$ **if** $\mathcal{S}ol(L_S \prec\!\!\!\prec M) = \{\varsigma_1, \ldots, \varsigma_n\}$
(ApplicationFail)	af	:	$S@M \longrightarrow \{[\mathsf{stk}]\}$ **if** $\mathcal{S}ol(L_S \prec\!\!\!\prec M) = \emptyset$
(Cooling)	c	:	$[M\ \{[M_1] \ldots [M_n]\}] \longrightarrow \{[M\ M_1] \ldots [M\ M_n]\}$
(AppCooling)	ac	:	$S@\{[M_1] \ldots [M_n]\} \longrightarrow \{[S@M_1] \ldots [S@M_n]\}$
(Flattening)	f	:	$\{[\{[M_1] \ldots [M_n]\}]\} \longrightarrow \{[M_1] \ldots [M_n]\}$

Figure 2.13: The semantics of the $\rho_{\langle\Sigma\rangle}$-calculus with strategies

2.5 Synchronous Big-Step Semantics

Definition 33 (Evolution step)**.** *An* evolution step *for a world* $[N\ S\ M]$ *is a reduction corresponding to the sequential composition of three stages corresponding to reduction using evaluation rules as follows:*

1. *the heating rule (2.10):*

$$[N\ S\ M] \longrightarrow [N\ \mathtt{seq}(S, \mathtt{try}(\mathsf{stk} \Rightarrow S\ M))@M]$$

2. *the union of the application rules (2.2) and (2.5), the distributing rule (2.6), the flattening rule (2.7), and the* stk *rules (2.8) and (2.9), and modulo the structural congruence relation on worlds given in Definition 32:*

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

3. *the cooling rule (2.4):*

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

An evolution step is silent *(or non-observable), denoted by* $\xrightarrow{\tau}$*, if the recovery rule* $\mathsf{stk} \Rightarrow S\ M$ *is applied during the second stage. In other words, the strategy* S *chosen to be applied on* M *fails. An evolution step is* visible *if it is not silent.*

Definition 34 (Inert and stable worlds)**.** *We say that a world* V *is* inert *if either:*

- $\exists\ O_1, \ldots, O_m \in \mathcal{O}$ *such that* $V \equiv [O_1 \ldots O_m]$ *or*

- $\exists\ S_1, \ldots, S_n \in \mathcal{S}$ *of the same order such that* $V \equiv [S_1 \ldots S_n]$.

We say that a world $V = [S_1 \ldots S_n\ O_1 \ldots O_m]$ *is* stable *if it can evolve only by silent steps.*

A multiverse is inert *(resp.* stable*) if every composite world is inert (resp. stable).*

In other words, a world is inert if it consists only of a multiset of structured objects from $\mathcal{O}$ or of a multiset of non-interacting strategies, and it is stable if all abstract molecules are irreducible with respect to the rest of the strategies in that world.

Definition 35 (Big one-step reduction of a world)**.** *A* big one-step reduction *of a world is induced by finitely many silent evolution steps and one visible evolution step. We denote it by* $\Mapsto$ *or* $\overset{S}{\Mapsto}$ *for* $(\xrightarrow{\tau})^* \longrightarrow$ *and* $(\xrightarrow{\tau})^* \longrightarrow_S$ *respectively if the successfully applied strategy is relevant.*

By suppressing the silent steps, we obtain an equivalent model but smaller in size. The choice of the strategy and the abstract molecule for heating should be fair [CGP00] in the sense that if a strategy is chosen infinitely often for application, then the other strategies as well must be chosen infinitely often. If a world contains more than one strategy, we do not want only one strategy to be applied over and over again since, in particular, if it is not applicable, the system will never evolve.

Informally, the *big-one-step reduction relation* between a strategy S and an abstract molecule M states that:

$$[S\ M] \Mapsto \{[M_1] \ldots [M_n]\} \quad \textbf{if} \quad [S](M) = \{M_1, \ldots, M_n\}, n \geq 1$$

We need such big step reduction level in the $\rho_{\langle\Sigma\rangle}$-calculus instantiated for port graphs in order to verify in Chapter 7 the satisfiability of a temporal formula only after cooling and before heating a world.

The big-one-step reduction of a multiverse corresponds to big-one-step reduction of a maximum number of worlds such that at least one world is reducible in a big-one-step:

Definition 36 (Big-one-step reduction of a multiverse)**.** *We say that a multiverse* $\{V_1 \ldots V_n\}$ *reduces synchronously in a big-one-step into the multiverse* W*, written* $\{V_1 \ldots V_n\} \Mapsto W$*, if there is a partition* $\{i_1, \ldots, i_p\} \cup \{i_{p+1}, \ldots, i_n\}$ *of* $\{1, \ldots, n\}$*, with* $1 \leq p \leq n$*, where:*

- $V_{i_k} \Mapsto W_i$*, for all* k*,* $1 \leq k \leq p$*, and*
- V_{i_l} *is stable, for all* l*,* $p + 1 \leq l \leq n$*,*

such that $W \equiv \{W_{i_1} \ldots W_{i_p} V_{i_{p+1}} \ldots V_{i_n}\}$.

2.6 Possible Strategies for the Calculus

One possible extension of the calculus is to benefit from the intrinsic concurrent nature of rewriting [Mes92]. In this way, all possible interactions can take place in parallel in

a world modeling the Brownian motion – an essential aspect in the chemical model. A parallel strategy should take several abstractions and apply them on an abstract molecule if their left-hand sides match disjoint parts of the molecule. Then for each particular structure Σ, one has to define the disjointness of two submolecule in an abstract molecule.

By adding more control using strategies to group the application of abstractions, we can formalize other types of parallelism. A strategy of interest for regulating the parallelism when modeling real systems and biological ones in particular, concerns a stochastic dimension on choosing the abstractions to apply. A formal description of the stochastic application of rewrite rules (abstraction) can be found in [Spi06].

We get sequential application of abstractions by using for instance the *don't care* strategy from the ELAN language [BKKR01a]: it chooses from a given set of strategies a non-failing strategy for the current state of the system; if it groups all strategies from the state, then the evolution of the system is sequential since only one strategy is applied in each evolution step. In [SMC+08] a stochastic sequential strategy is defined for P systems based on the Gillespie's stochastic simulation algorithm [Gil77] where the rule are assigned probabilities that may change during the simulation. In addition, the concepts of probabilistic rewriting and probabilistic strategies have already been studied in [BK02, BG06].

Other parallel strategies used in rule-based biologically-inspired models of computation are for instance those inspired by the different control mechanisms available for the P systems, the models of the membrane calculus [Pau02]. In [IYD05] the authors review some notions of parallelism for P systems. In terms of the calculus we have presented in this chapter, for k the number of strategies applicable on the structured objects in a world and $n \leq k$, the following parallel reductions can be considered:

n-Max-Parallel specifying that a maximal set of at most n strategies is applied in a step and no larger subset is applicable;

$\leq n$-Parallel specifying that any subset of at most n strategies is applied;

n-Parallel specifying that any subset of exactly n strategies is applied (this is also called *bounded parallelism*).

2.7 Comparison with the γ-Calculus and HOCL

In this section we compare the expressivity of the abstract biochemical calculus with respect to the γ-calculus and HOCL which it generalizes.

The major aspect this generalization concerns is the use of rewrite rules on structured objects in the ρ-calculus style instead of variable abstractions like in the λ-calculus. The HOCL considers a large class of patterns which are suitable renamings of often very complex λ-expressions. However, we gain on expressive power in the $\rho_{\langle\Sigma\rangle}$-calculus by considering a more abstract class of patterns.

In the γ-calculus, the conditions on the reductions due to the encapsulation of solutions impose an innermost application strategy. The $\rho_{\langle\Sigma\rangle}$-calculus does not have an

encapsulation constructor for abstract molecules, but this syntactic lack is prevailed upon adding strategies in the calculus. Still, the reduction in the $\rho_{\langle\Sigma\rangle}$-calculus uses an evaluation strategy, the call-by-name evaluation, which forbids the reduction under an arrow operator in an abstraction. The use of structured patterns enriches the $\rho_{\langle\Sigma\rangle}$-calculus with the capability of encoding strategies as abstractions. This opened the possibility of defining various strategies for controlling the evolution of a state.

In a first stage, HOCL extends the γ-calculus with two high-level constructs: the conditions on reactions and atomic capture (several arguments for a γ-abstraction). The use of strategies allows the definition of abstractions with Boolean conditions since we can define the congruence relation on Booleans. We did not formalize conditional abstractions here, but the approach is similar for the one in the ρ-calculus for encoding conditional rewriting [CK01] where a conditional rewrite rule $l \twoheadrightarrow r$ **if** c is represented by the ρ-term $l \twoheadrightarrow (\mathit{True} \twoheadrightarrow r)c_\rho$ where *True* is a constant and c_ρ the ρ-term describing the reduction of the term c to a Boolean constant, either *True* or *False*.

We also remark that the persistent strategies correspond to n-shot reaction rules in the γ-calculus which are reaction rules not consumed by a reduction.

We did not consider the extension of HOCL with negative of infinite multiplicities for molecules [BFR06b, Rad07], but this is also possible for the $\rho_{\langle\Sigma\rangle}$-calculus using a similar approach.

2.8 Conclusions and Perspectives

In this chapter we have introduced a higher-order calculus based on the classical chemical metaphor. By allowing complex patterns for reaction and a rich structure for describing molecules and possible connections between them, we provide the calculus with an additional biological flavor.

This chapter presents the foundational part of the thesis. In the next chapters we will instantiate the structure Σ with the structure of port graphs and the transformation of the structured molecules with the port graph rewriting relation as defined in Chapter 3. The main features of the resulting calculus are reviewed in Chapter 4 together with the proof that port graphs represent a unifying structure for representing port graph rewrite rules and for describing operationally the application of a port graph rewrite rule on a port graph. Then, in Chapter 7, the biochemical calculus on port graphs is enriched with verification features, an approach that could be used for other structures as well.

Perspectives

To conclude this chapter, we would like to point out some future directions related to the work presented here.

- An interesting research direction is to see how the topological structures considered in MGS [GM02a, Spi06] can fit in the calculus presented here. The topological structures and their transformations were shown to be useful in modeling dynami-

cal systems with a dynamical structure [Gia03] (for membrane systems in [GM02b] for instance and a real biological example in [SM05]).

- In the semantics of the $\rho_{\langle\Sigma\rangle}$-calculus an abstraction is applied locally on the object representing the physical structure of the state of the modeled system since we solve a submatching equation between the left-hand side of an abstraction and an abstract molecule. But what if we want an abstraction to be applied globally on an abstract molecule? For instance, we want a rule to find a pattern M, keep it and delete everything else in the world. If we consider the abstraction $MN \Rightarrow M$, it will provide all possible results of matching N with molecules in the world not matched by M. Of course this will provide the intended result, but we will also obtain other solutions corresponding to partial contexts. We could handle this problem by considering two types of abstraction, *global* and *local* ones, whereas, in the semantics, the evaluation rules describing their applications will consider solutions of a matching equation and a submatching equation respectively. Then the example of abstraction above should be a global one. We should analyze the implications of differentiating the abstractions in this way.

- The current approach for modeling the interaction between an abstraction A and an abstract molecule M is first to heat them up, and only then test if A is applicable on M (i.e., if the left-hand side of A submatches M) and if positively, apply it. Another approach worth considering is to move the test of applicability of an abstraction inside the heating rule; then the application rule will never fail. However, such test of applicability of an abstraction on an abstract molecule no longer works as wanted for strategies since the left-hand side of a strategy may be a variable and the submatching problem always has a solution. Then the problem that should be solved is to determine the application conditions for different strategy constructors.

- In Section 2.6 we have mentioned some possible ways of parallel evolution for systems. We shall investigate how these can be expressed as rewrite strategies, and, if successful, how to encode them as strategies in the $\rho_{\langle\Sigma\rangle}$-calculus.

3 Port graph rewriting

3.1 Introduction

Graphs are data structures (or abstract data types) widely used for describing complex structures, like UML diagrams, microprocessor design, XML documents, communication networks, data and control flow, neural networks, biological systems, etc., in a direct and intuitive way. For complex systems, in addition to providing a static description, transformations of graphs allow the modeling of their dynamic evolution in a uniform manner. Computing by graph transformation is not limited to programming, specification or implementation by graph transformation, it is a fundamental concept also for concurrency and distribution, for visual modeling and model transformation, for graph algorithms and computational models in general [EP05]. Different approaches have been proposed to formalize graph transformation and applications [Roz97, EEKR97, EKMR97].

In this chapter we study a particular class of graphs, where nodes have explicit connection points called *ports* with the edges attached to specific ports of nodes. This graph structure is inspired by the molecular complexes formed in protein-protein interactions in a biochemical network: a protein is characterized by a collection of small patches on its surface, called functional domains or sites, and two proteins can bind at such sites. Then a protein with its collection of sites is modeled by a node with ports, and a bond between two proteins by an edge in a port graph. The objective of this chapter is to formally define port graphs, port graph submatching, port graph rewrite rules and a port graph rewrite relation such that we can instantiate the Abstract Biochemical Calculus introduced in Chapter 2. The result is a calculus on port graphs presented in Chapter 4 with applications in modeling autonomous systems and biochemical networks.

We start by defining port graphs in Section 3.2 and we continue with the definition of a port graph morphism and the subsequent definitions of composition and subgraphs, in Section 3.3. We also introduce a particular port graph morphism defined only on nodes which will be used for encoding the correspondence between elements of the left-hand side and elements of the right-hand side of a port graph rewrite rule. In Section 3.4 we define the submatching and matching equations for port graphs and we provide a sound and complete submatching algorithm. In the next two sections, 3.5 and 3.6, we give the definition of port graph rewrite rules and define a rewrite relation on port graphs. Then, in Section 3.7 we briefly show how we obtain a strategic port graph rewrite relation by instantiating the definitions for abstract reduction systems and abstract strategies from Chapter 1. For the situations where we can or need to specify partial information about the ports in nodes in a port graph rewrite rule, we define weak port graphs in Section 3.8 and adapt the port graph rewrite relation defined earlier. In Section 3.9 we study the confluence of the port graph rewrite relation based on known results on hypergraphs.

We end up with a comparison with the bigraphical reactive systems in Section 3.10.

3.2 Port Graphs

We start by defining the way ports are associated to node names via a signature.

Definition 37 (P-Signature)**.** *A* p-signature *is a pair of sets of names* $\nabla = \langle \nabla_{\mathscr{N}}, \nabla_{\mathscr{P}} \rangle$ *where:*

- $\nabla_{\mathscr{N}}$ *is a set of node names and*
- $\nabla_{\mathscr{P}}$ *is a set of port names,*

such that each node name N *comes with a finite set of ports* $Interface(N) \subseteq \nabla_{\mathscr{P}}$*.*

Note that, by definition, the port names associated to a node name are pairwise distinct.

In what follows the symbols $a, b, c, \ldots$ range over the set $\nabla_{\mathscr{P}}$ of constant port names, $A, B, C, \ldots$ range over the set $\nabla_{\mathscr{N}}$ of constant node names, the symbols $x, y, z, \ldots$ range over the set $\mathcal{X}_{\mathscr{P}}$ of port name variables, and the symbols $X, Y, Z, \ldots$ range over the set of node name variables $\mathcal{X}_{\mathscr{N}}$. All symbols can be indexed. We denote by $\nabla^{\mathcal{X}}$ the p-signature associating to a node name in $\nabla_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{N}}$ an interface (finite set of port names) from $\nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}}$. We extend the definition of *Interface* over variable node names such that $Interface(X) \subseteq \nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}}$ for all $X \in \mathcal{X}_{\mathscr{N}}$.

Definition 38 (Port Graph)**.** *Given a fixed p-signature* $\nabla^{\mathcal{X}}$*, a* labeled port graph over $\nabla^{\mathcal{X}}$ *is a tuple* $G = \langle V_G, E_G, lv_G, le_G, s_G, t_G \rangle$ *where:*

- V_G *is a finite set of nodes;*
- E_G *is a finite set of edges,* $E_G = \{\langle (v_1, p_1), (v_2, p_2) \rangle \mid v_i \in V_G, p_i \in Interface(lv_G(v_i)) \cup \mathcal{X}_{\mathscr{P}}\}\}$*;*
- $lv_G : V_G \to \nabla_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{N}}$ *is the labeling function for nodes,*
- $le_G : E_G \to (\nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}})^2$ *is the labeling function for edges such that* $le_G(\langle (v_1, p_1), (v_2, p_2) \rangle) = (p_1, p_2)$*;*
- $s_G, t_G : E_G \to (V_G \times (\nabla_{\mathscr{P}} \cup \mathcal{X}_{\mathscr{P}}))$ *the source and target functions respectively, such that, for* $e \in E_G$ *with* $e = \langle (v_1, p_1), (v_2, p_2) \rangle$*,* $s_G(e) = (v_1, p_1)$ *and* $t_G(e) = (v_2, p_2)$*.*

We represent the nodes by unique identifiers which are non-empty words over literal symbols $\{i, j, k, \ldots\}$ and integers. For instance, $i.j.1$, 2, 1.3 are three node identifiers. The identifiers must be unique because we allow several nodes to have the same name.

We denote by $Id(G)$ the set of literal symbols occurring in the node identifiers in G and by $\mathcal{V}ar(G)$ we denote the set $Id(G)$ to which we add the set of name variables occurring in G .

In Figure 3.1 we illustrate two views of a port graph: on the left, we use the classical drawing of a labeled multigraph, while on the right, we emphasize the ports. We will use the latter more suggestive representation for port graphs by representing a node as a box with the identifier and the name placed outside the box and the port as small points on the surface of the box.

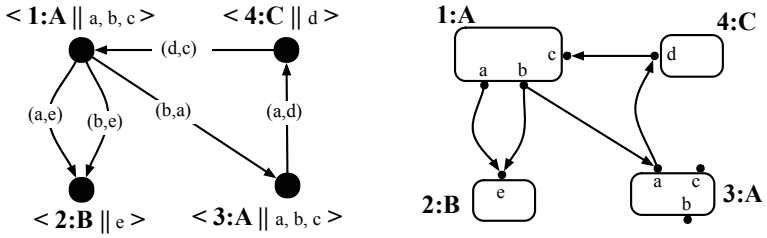


Figure 3.1: Two views of a port graph

A port graph can transformed into multigraph whose nodes are the old ports which have an associated information about the identity and name of the node they belonged to.

We motivate in the following the formalization of port graphs and port graph rewriting. This is closely related to the abstraction level we consider when modeling the structure and behavior of a particular system. The systems we want to model are based on entities interacting, connecting in particular, based on their internal components. If we decentralize the concept of node with ports, the global view of the system is weakened, making less obvious the main actors in the system. Take for instance the case of protein complexes: the reactions are described in terms of two proteins binding at some specific sites; we can express the same operation by saying that two specific sites of some specific proteins are binding, but the emphasis goes mainly to sites, instead of the proteins.

3.3 Port Graph Morphisms and Node-Morphisms

In this section we introduce concepts related to port graphs necessary for defining later the transformations of port graphs. We start by defining port graph morphisms used for relating two port graphs with the particular aim of defining the submatching relation. We also see some particular instances of such morphisms.

A port graph morphism relates the elements of two port graphs by preserving sources and targets of edges, constant node names and associated interfaces up to a variable renaming.

Definition 39 (Port graph morphism)**.** *Given two port graphs G and H over $\nabla^{\mathcal{X}}$ with $\mathcal{V}ar(H) \subseteq \mathcal{V}ar(G)$, a* port-graph morphism *$f : G \to H$ is a pair of function $f = \langle f_V : V_G \to V_H, f_E : E_G \to E_H \rangle$ which satisfies the following requirements:*

- *for each $v \in V_G$, $lv_G(v) =_\psi (f_V \circ lv_H)(v)$ provided that $lv_G(v) \notin \mathcal{X}_{\mathcal{N}}$ and $(f_V \circ lv_h)(v) \notin \mathcal{X}_{\mathcal{N}}$;*

- *for each* $v \in V_G$, *if* $Interface(lv_G(v)) = \{p_1, \ldots, p_i, x_1, \ldots, x_j, y_1, \ldots, y_l\}$ *with* $i, j, l \geq 0$, *then* $Interface((f_V \circ lv_H)(v)) = \{p_1, \ldots, p_{i+j}, z_1, \ldots, z_l\}$ *with* $0 \leq k \leq j$ *such that* $\{y_1, \ldots, y_l\} =_\psi \{z_1, \ldots, z_l\}$
- *for each* $e \in E_G$, *if* $le_G(e) = \langle (v_1, p_1), (v_2, p_2) \rangle$, *then* $(f_E \circ le_H)(e) = \langle (f_V(v_1), p'_1), (f_V(v_2), p'_2) \rangle$ *with* $p'_i \in Interface(f_V(v_i))$, *such that* $p_i =_\psi p'_i$.

where ψ *is a variable renaming.*

Definition 40 (Port subgraph)**.** *A port graph morphism* $f : G \to H$ *is an* inclusion *(also denoted by* $f : G \hookrightarrow H$*) if* $f_V(v) = v$ *and* $f_E(e) = e$ *for all* $v \in V_G$ *and* $e \in E_G$ *up to a variable renaming. Then* G *is a* port subgraph *of* H *which is denoted by* $G \subseteq H$.

We say that a port subgraph G *of* H *is a* full port subgraph *if for any two ports in* G *connected by an edge* e *in* H, *the edge* e *is also in* G.

Definition 41 (Induced port subgraph)**.** *Given a port graph morphism* $f : G \to H$, $f(G)$ *is called the* induced port subgraph of H.

Definition 42 (Composition of port graph morphisms)**.** *Let* $f : G \to H$ *and* $g : H \to I$ *two port graph morphisms. Then their* composition $h : G \to I$ *usually denoted by* $g \circ f$ *(or* $f; g$*) is the pair of function* $h = \langle h_V : V_G \to V_I, h_E : E_G \to E_I \rangle$ *defined by* $h_V = g_V \circ f_V$ *and* $h_E = g_E \circ f_E$.

As in the case of classical graph morphisms, the composition operation on port graph morphisms is associative with the identity morphism as neutral element.

Definition 43 (Particular port graph morphisms)**.** *Let* $f = \langle f_V, f_E \rangle : G \to H$ *be a port graph morphism. We say that* f *is a* monomorphism *(or mono or injective morphism) if* f_V *and* f_E *are both injective. We say that* f *is a* epimorphism *(or epi or surjective morphism) if* f_V *and* f_E *are both surjective.*

If f *is mono and epi, i.e., both* f_V *and* f_E *are bijective, then* f *is an* isomorphism *(or iso) and we write* $G \cong H$*; in addition, we denote by* $[G] = \{H \mid H \cong G\}$ *the isomorphism class of* G.

In the following we introduce a particular graph morphism that maps a node from a graph to a set of nodes of another graph.

Definition 44 (Node-morphism)**.** *Given two port graphs* G *and* H *over* $\nabla^{\mathcal{X}}$ *with* $\mathcal{V}ar(H) \subseteq \mathcal{V}ar(G)$, *a* node-morphism $\xi : G \to H$ *is a function* $\xi : V_G \to \mathcal{P}(V_H)$.

If $\xi_1 : G \to H$ *and* $\xi_2 : H \to I$ *are two node-morphisms, then their* composition *is a node-morphism* $\xi_2 \circ \xi_1 : G \to I$ *defined as follows:*

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

By convention, if $\xi_1(a) = \emptyset$, *then* $(\xi_2 \circ \xi_1)(a) = \emptyset$.

The identity node-morphism $\xi : G \to G$ *associates to a node the singleton containing the same node:* $\xi(v) = \{v\}$, $\forall v \in V_G$.

We extend a node-morphism $\xi : V_G \to \mathcal{P}(V_H)$ *over the edges of* G*, by defining* $\xi : E_G \to \mathcal{P}(E_H)$ *as follows:*
for $e \in E_G$ *with* $e = \langle (v,p),(u,r)\rangle$*,*

- *if* $\xi(v) = \{v_1,\ldots,v_k\}$ *and* $\xi(u) = \{u_1,\ldots,u_n\}$*,* $k,n \geq 1$*, then* $\xi(e) = \{\langle (v_i,p),(u_j,r)\rangle \mid p \in \mathit{Interface}(v_i), r \in \mathit{Interface}(u_j)\}$*;*
- *if* $\xi(v) = \emptyset$ *or* $\xi(u) = \emptyset$ *then* $\xi(e) = \emptyset$*.*

We use the notations $G \xrightarrow{f} H$ and $G \overset{\xi}{\Rightarrow} H$ for a port graph morphism $f : G \to H$ and a node-morphism $\xi : G \to H$ respectively.

Remark 4. A port graph morphism $g : G \to H$ can be seen as a node morphism that associates to each element x from G the singleton $\{g(x)\}$. The composition of a node-morphism with a port graph morphism $G \xrightarrow{g} H \overset{\xi}{\Rightarrow} I$ is a node morphism $G \overset{\xi\circ g}{\Longrightarrow} I$ defined as follows:

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

Conversely, the composition of a port graph morphism with node morphism $G \overset{\xi}{\Rightarrow} H \xrightarrow{g} I$ is also a node morphism $G \overset{g\circ\xi}{\Longrightarrow} I$ defined as follows:

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

3.4 Port Graph Matching and Submatching

3.4.1 General Definition

Let L and G be two port graphs over the same p-signature $\nabla^{\mathcal{X}}$ such that they do not have common node identifiers, node name variables, nor port name variables.

Definition 45 (Submatching)**.** *We say that* L strictly submatches G *if there is a decomposition of* G *as* $g(L) \cup G^- \cup \mathcal{B}$ *where:*

- $g : L \to G$ *is a port graph morphism called the* submatching morphism *such that* $g(L)$ *is a full subgraph of* G*;*
- G^- *is a port graph called the* context*, with* $G^- = G \setminus g(L)$*, given by the set of nodes* $V_{G^-} = V_G \setminus V_{g(L)}$*, and the set of edges* $E_{G^-} = \{\langle (u,p),(v,r)\rangle \in E_G \mid u,v \in V_{G^-}\}$*;*
- $\mathcal{B}$ *is a set of edges (called* bridges*) which have one endpoint in the context graph and the other in the matched subgraph.*

We then write $G = G^-[g(L)]_{\mathcal{B}}$.

We denote the submatching problem *between a pattern port graph* L *and a subject port graph* G *by* $L \preccurlyeq G$. *The solutions of the submatching problem have the form* $(g, G^-, \mathcal{B})$ *and we call them* submatchers. *Then by* $\mathcal{S}ol(L \preccurlyeq G)$ *we denote the* set of all solutions *for the submatching problem.*

In general, we assume that for any submatching problem $L \preccurlyeq G$, the set of variables and node identifiers occurring in each port graph are disjoints, i.e., $\mathcal{V}ar(L) \cap \mathcal{V}ar(G) = \emptyset$ and $Id(L) \cap Id(G) = \emptyset$. If it is not the case we rename appropriately the variables and the node identifiers such that this condition is met. We assume that all ports with variable names in the same port graph are pairwise distinct.

We call the matching and the submatching *strict* when the matched subgraph $g(L)$ must be a full induced port sugbraph of G. A *non-strict submatching* considers a fourth component of the decomposition of G along a port graph L and a port graph morphism g represented by a set of unmatched edges from G with both endpoints in $g(L)$. We usually denote by $\mathcal{U}_e$ the set of unmatched edges. Then $g(L) \cup \mathcal{U}_e$ is a full induced port subgraph of G and we write $G = G^-[g(L)]_{\mathcal{B}}^{\mathcal{U}_e}$.

Formally, the sets of bridges and unmatched edges are defined as follows:

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

We obtain the usual definition of matching between the pattern port L graph and the subject port graph G from the definition of the submatching where the context is empty, or equivalently, where the submatching morphism is in fact an isomorphism.

Definition 46 (Matching)**.** *We say that* L strictly matches G *if there is a port graph isomorphism* $g : L \to G$. *We denote the* matching problem *between a pattern port graph* L *and a subject port graph* G *by* $L \ll G$.

The matching is *non-strict* if the port graph morphism g is a monomorphism and only g_V is required to be surjective. Then G is decomposed as the port graph $g(L)$ and the set of unmatched edges (i.e., edges without a pre-image in L).

3.4.2 A Submatching Algorithm

In order to provide a submatching algorithm on object port graphs, we represent a port graph by the set of labeled nodes and the set of (possibly empty) adjacency equations (or lists). A *labeled node* is defined by the tuple consisting of the node identifier, node name, and interface. An *adjacency equation* is defined by the source node identifier and the set of neighbors, where a *neighbor* is given by the target node identifier and the set of edges (pairs of ports):

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

We use the symbol S to range over sets of any kind, the symbol N to range over sets of labeled nodes, the symbol E to range over sets of adjacency equations, and the symbols s, n, and e to range over elements of a generic set, over nodes, and over adjacency equations respectively. All these symbols can be indexed.

Example 11. The port graph drawn in Figure 3.2, let us denote it by G, has the following encoding:

- the set of labeled nodes:

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

- the set of adjacency equations:

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

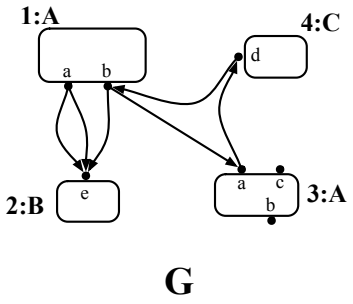


Figure 3.2: An example of port graph

Definition 47 (Pre-solution of the port graph submatching problem)**.** *We say that* (σ, N, E) *is a* pre-solution of the submatching problem $(N_1, E_1) \preccurlyeq (N_2, E_2)$ *if:*

1. $\sigma : Id(N_1) \cup \mathcal{X}_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{P}} \to Id(N_2) \cup \nabla$ *is a substitution defined on the set of node identifiers, node name variables, and port name variables occurring in* (N_1, E_1)*;*
2. $\sigma(N_1) \subseteq N_2$*,* $\sigma(E_1) \subseteq E_2$*, i.e.,* $\sigma((N_1, E_1))$ *is a subgraph of* (N_2, E_2)*;*
3. $E = E_2 \setminus \sigma(E_1)$*,* $N = N_2 \setminus \sigma(N_1)$*.*

If (N_1, E_1) and (N_2, E_2) encode the port graphs G_1 and G_2 respectively, and (σ, N, E) is a pre-solution of the submatching problem $(N_1, E_1) \preccurlyeq (N_2, E_2)$, then we can easily find the corresponding solution for the submatching problem $G_1 \preccurlyeq G_2$. Therefore we

will use also the term of submatching solution for a pre-solution unless it is not clear from the context.

In the following we present the submatching algorithm. We encode the submatching problem $(N_1, E_1) \prec\!\!\!\ll (N_2, E_2)$ given as input to the submatching algorithm as a 5-tuple $N_1 \prec\!\!\!\ll N_2 \wr E_1 \star E_2 \int \emptyset \rhd \emptyset$ on which the rules of the algorithm can be applied. The terms involved in the submatching algorithm, called *matching terms*, have the structure "$\_ \wr \_ \star \_ \int \_ \rhd \_$" with the following meaning for each argument:

(1^{st}) a list (conjunction) of matching equations obtained recursively by decomposing and solving matching equations and by traversing the port graph based on the neighboring information;

(2^{nd} & 3^{rd}) two sets of adjacency equations providing the neighboring information for each side of the matching equations in the first position;

(4^{th}) a set of nodes that will form the context port graph, extracted from matching sets of nodes of different sizes;

(5^{th}) the (partial) substitution constructed from solving the matching equations in the first argument.

We consider that a matching term is in a *normal form* if either the first argument has the value **false**, or the first argument has the value **true** (i.e., there are no more matching equations) and the first list of adjacency equations is empty. Hence the two possible normal forms of a matching term are $\mathbf{false} \wr E_1' \star E_2' \int N' \rhd \sigma'$ and $\mathbf{true} \wr \emptyset \star E_2' \int N' \rhd \sigma$.

We group the rules in three sets according to their type of modification on terms: decomposing ($\mathbf{D_{1-9}}$), solving ($\mathbf{S_{1-3}}$), and cleaning ($\mathbf{C_{1-2}}$). A common feature for all decomposing and solving rules is that each of the rule attempts to solve the first/leftmost matching equation of the list from the first position of the term.

In the following we present in detail each set of rules.

The **decomposing rules** act only on the first matching equation in the list by decomposing or deleting it, or by giving a failure result for the submatching. We denote by M a list of matching equations.

The rule ($\mathbf{D_1}$) decomposes the matching equation between two sets of nodes, ports, neighbors, or edges into a conjunction of two matching equations provided that each set is non-empty and at most one set is a singleton. We impose this condition such that this rule should not apply on matching two elements which can be seen as singletons. For S a set, we denote by $|S|$ the number of its elements.

$$(\mathbf{D_1}) \quad (S_1 \prec\!\!\!\ll S_2) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow \bigvee_{\substack{s_1 \in S_1 \\ s_2 \in S_2}} (s_1 \prec\!\!\!\ll s_2) \wedge (S_1 \setminus \{s_1\}) \prec\!\!\!\ll (S_2 \setminus \{s_2\}) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma$$
$$\text{if } |S_1| \geq 1, |S_2| \geq 1, |S_1| + |S_2| \geq 3$$

Submatching a non-empty set against an empty set leads to failure:

$$(\mathbf{D_2}) \quad (S \prec\!\!\!\ll \emptyset) \wedge M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow \mathbf{false} \wr E_1 \star E_2 \int N \rhd \sigma$$
$$\text{if } |S| \geq 1$$

When submatching two sets of nodes, with the pattern set empty, then the subject set belongs to the context port graph, hence this set is moved to the context nodes component of the match term.

$$(\mathbf{D_3}) \quad (\emptyset \lessdot N_2) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow M \wr E_1 \star E_2 \wr N \cup N_2 \rhd \sigma$$

Whereas when submatching two sets of neighbors or edges with the pattern set empty, the matching equation is simply deleted. As we will see later, the information on this non-empty set is not lost.

$$(\mathbf{D_4}) \quad (\emptyset \lessdot S) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow M \wr E_1 \star E_2 \wr N \rhd \sigma$$

A matching equation between an empty set of ports and a non-empty set of ports leads to failure because two matched nodes must have at least their port sets of equal size:

$$(\mathbf{D_5}) \quad (\emptyset \lessdot P) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow \mathbf{false} \wr E_1 \star E_2 \wr N \rhd \sigma$$

A matching equation between two nodes is decomposed in three matching equations corresponding to the three components: the identifier, the name, and the set of ports.

$$\begin{aligned}(\mathbf{D_6}) \quad & (id_1 : nname_1 : P_1 \lessdot id_2 : nname_2 : P_2) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow \\ & (id_1 \lessdot id_2) \wedge (nname_1 \lessdot nname_2) \wedge (P_1 \lessdot P_2) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma\end{aligned}$$

For building the matching equations in the case of weak port graphs, we extract from the subject node a set of ports of the same size with that of the set of ports in the pattern node. Then the set of ports resulting as a difference is appended along with the node identifier and name to an extra argument of the match term:

$$\begin{aligned}(\mathbf{D_{6bis}}) \quad & (id_1 : nname_1 : P_1 \lessdot id_2 : nname_2 : P_2) \wedge M \wr E_1 \star E_2 \wr N \# N_0 \rhd \sigma \rightarrow \\ & \bigvee_{\substack{P_2 = P_2', P_2'' \\ |P_1| = |P_2'|}} (id_1 \lessdot id_2) \wedge (nname_1 \lessdot nname_2) \wedge (P_1 \lessdot P_2') \wedge M \\ & \qquad \wr E_1 \star E_2 \wr N \# N_0 \cup \{id_2 : nname_2 : P_2''\} \rhd \sigma\end{aligned}$$

The following two rules are instances of the usual decomposition rule for operators with fixed arity. The first one is for matching two neighbors, and the second one for matching two edges.

$$\begin{aligned}(\mathbf{D_7}) \quad & (id_1 \frown S_1 \lessdot id_2 \frown S_2) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow \\ & (id_1 \lessdot id_2) \wedge (S_1 \lessdot S_2) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma\end{aligned}$$

$$\begin{aligned}(\mathbf{D_8}) \quad & (p_1, p_2) \lessdot (p_3, p_4) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow \\ & p_1 \lessdot p_3 \wedge p_2 \lessdot p_4 \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma\end{aligned}$$

Matching different constants yields a failure, otherwise the matching equation is deleted:

$$\begin{aligned}(\mathbf{D_9}) \quad & a \lessdot b \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow \\ & \text{if } (a \neq b) \text{ then } \mathbf{false} \wr E_1 \star E_2 \wr N \rhd \sigma \\ & \text{else } M \wr E_1 \star E_2 \wr N \rhd \sigma\end{aligned}$$

Each of the **solving rules** is applied if the left-hand side of a matching equation is a variable. The current substitution is extended with a mapping of the variable to the constant in the right-hand side of the matching equation, and every occurrence of the variable is replaced by the constant in the whole matching term.

The first solving rule is applied on node identifiers. In addition to the above mentioned behavior, the matching equation is replaced by a new matching equation between the right-hand sides of the adjacency equations of the two identifiers.

$$(\mathbf{S_1}) \quad (x \ll id) \wedge M \wr E_1; x = H_1; E_2 \star E_3; id = H_2; E_4 \wr N \rhd \sigma \rightarrow \\ (H_1\{x \mapsto id\} \ll H_2) \wedge M\{x \mapsto id\} \wr (E_1; x = H_1; E_2)\{x \mapsto id\} \star E_3; v = H_2; E_4 \wr \\ N \rhd \sigma \cup \{x \mapsto id\}$$

The following two rules handle matching between node names and port names:

$$(\mathbf{S_2}) \quad (x \ll nname) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow \\ (M\{x \mapsto nname\} \wr E_1\{x \mapsto nname\} \star E_2 \wr N \rhd \sigma \cup \{x \mapsto nname\}$$

$$(\mathbf{S_3}) \quad (x \ll p) \wedge M \wr E_1 \star E_2 \wr N \rhd \sigma \rightarrow \\ (M\{x \mapsto p\} \wr E_1\{x \mapsto p\} \star E_2 \wr N \rhd \sigma \cup \{x \mapsto p\}$$

The **cleaning rule** $(\mathbf{C_1})$ is applied when all matching equations are solved. It deletes common sets of neighbors from the adjacency equations of the same identifier in order to obtain the (rest of) edges of the context and the bridges.

$$(\mathbf{C_1}) \quad \mathbf{true} \wr E_1; id = H; E_2 \star E_3; id = H', H, H''; E_4 \wr N \rhd \sigma \rightarrow \\ \mathbf{true} \wr E_1; id = \emptyset; E_2 \star E_3; id = H', H''; E_4 \wr N \rhd \sigma$$

We saw that some decomposition rules transform a matching term into a disjunction of matching terms. The disjunction is a commutative and associative operator. In addition, we define the matching terms having the first component equal to **false** as identity elements by removing them using the rule $(\mathbf{C_2})$:

$$(\mathbf{C_2}) \quad (\mathbf{false} \wr E_1 \star E_2 \wr N \rhd \sigma) \vee T \rightarrow T$$

Example 12. We show below a few steps of the execution of the submatching algorithm for the pattern port graph in Figure 3.3 and for the subject port graph G from Example 11.

Firstly we give the encoding of the pattern as a pair of a set of nodes and a set of adjacency equations respectively:

$$N_1 = \{(i : X : \{x_1, x_2\}), (j : Y : \{y\})\}$$

and

$$E_1 = \{i ::= (i^\frown\emptyset), (j^\frown(x_1, y), (x_2, y));\ j ::= (i^\frown\emptyset), (j^\frown\emptyset)\}$$

Let $G = (N_2, E_2)$. Then the submatching problem $(N_1, E_1) \prec\!\!\!\prec (N_2, E_2)$ is reduced as follows:

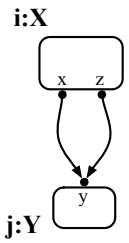


Figure 3.3: An example of a port graph used as pattern in a matching problem

$$N_1 \prec\!\!\!\ll N_2 \wr E_1 \star E_2 \wr \emptyset \rhd \emptyset \rightarrow_{\mathbf{D_1}}$$
$$(i : X : \{x_1, x_2\}) \prec\!\!\!\ll (1 : A : \{a, b\}) \wedge$$
$$(j : Y : \{y\}) \prec\!\!\!\ll (2 : B : \{e\}, 3 : A : \{a, b, c\}, 4 : C : \{d\}) \wr E_1 \star E_2 \wr \emptyset \rhd \emptyset$$
$$\bigvee \ldots$$

The result contains three more matching terms corresponding to matching equations between the node i in the pattern and the nodes 2, 3, and 4 from G. We continue showing the reduction for the first matching equation:

$$(i : X : \{x_1, x_2\}) \prec\!\!\!\ll (1 : A : \{a, b\}) \wedge$$
$$(j : Y : \{y\}) \prec\!\!\!\ll (2 : B : \{e\}, 3 : A : \{a, b, c\}, 4 : C : \{d\}) \wr E_1 \star E_2 \wr \emptyset \rhd \emptyset \rightarrow_{\mathbf{D_6}}$$
$$i \prec\!\!\!\ll 1 \wedge X \prec\!\!\!\ll A \wedge (\{x_1, x_2\} \prec\!\!\!\ll \{a, b\}) \wedge$$
$$(j : Y : \{y\}) \prec\!\!\!\ll (2 : B : \{e\}, 3 : A : \{a, b, c\}, 4 : C : \{d\}) \wr E_1 \star E_2 \wr \emptyset \rhd \emptyset \rightarrow_{\mathbf{S_1}}$$
$$((1 \frown \emptyset), (j \frown (x_1, y), (x_2, y))) \prec\!\!\!\ll ((1 \frown \emptyset), (2 \frown (a, e), (a, e), (b, e)))$$
$$\wedge (X \prec\!\!\!\ll A) \wedge (\{x_1, x_2\} \prec\!\!\!\ll \{a, b\}) \wedge$$
$$(j : Y : \{y\}) \prec\!\!\!\ll (2 : B : \{e\}, 3 : A : \{a, b, c\}, 4 : C : \{d\}) \wr E_1\{i \mapsto 1\} \star E_2 \wr \emptyset \rhd \{i \mapsto 1\}$$

And the reduction continues.

This example illustrates the complexity of the reductions due to solving submatching problems between sets. Clearly, the submatching algorithm presented here is not designed to be very efficient. We just presented a simple and very intuitive algorithm for solving a submatching problem. More efficient algorithms are available in the graph literature, in particular for tools implementing graph transformations like PROGRES (PROgramming with Graph REwriting Systems) [SWZ97] and AGG (Attributed Graph Grammars) [ERT97]. We could use the existing results on efficiently implementing graph transformations to provide a implementation for port graph transformation; this is possible by considering the equivalent definition of a port graph as a graph with an appropriate labeling function including the port names in labels. In consequence the complexity of the usual problems on graphs is the same for port graphs. However, we should perform a thorough analysis on the implementation differences and performances to see if it is also efficient on graphs obtained from port graphs.

Lemma 1. The submatching algorithm terminates.

Proof. Since the cleaning rules are applied on matching terms already reduced using decomposing and solving rules, we separate the termination proof in a first stage for the decomposing and solving rules, and in a second stage for the cleaning rules.

In order to show that the rewriting system given by the decomposing and solving rules terminates, we define a reduction order on matching terms as follows:

$$MT' > MT'' \text{ if } \phi(MT') > \phi(MT'')$$

where:

- $\phi(M \wr E_1 \star E_2 \wr N \rhd \sigma) = (|Var(M)| + |Var(E_1)|, symb(M))$;
- $(n_1, n2) > (m_1, m_2)$ if $n_1 > m_1$ or $n_1 = m_1$ and $n_2 > m_2$;
- $|S|$ computes the size of the set S taken as argument;
- Var return the set of variables occurring in the argument;
- $symb(M)$ computes the number of symbols in M that are not logical connectives:
  - $symb(\{s_1, \ldots, s_n\}) = n - 1 + symb(s_1) + \ldots + symb(s_n)$ for $\{s_1, \ldots, s_n\}$ a generic set
  - $symb(\emptyset) = 0$
  - $symb(T_1 \lessdot T_2) = symb(T_1) + symb(T_2)$
  - $symb(M_1 \wedge \ldots \wedge M_n) = symb(M_1) + \ldots + symb(M_n)$
  - $symb(MT_1 \vee \ldots \vee MT_n) = max\{symb(MT_1), \ldots, symb(M_n)\}$
  - $symb(\mathbf{false}) = 0$
  - $symb(f(t_1, \ldots, t_n)) = 1 + symb(t_1) + \ldots + symb(t_n)$ for f an operator of arity n

We prove the termination of the rewriting system consisting of the decomposing and the solving rules by showing that $l > r$ for every rule $l \to r$ (**if** *cond*) in the matching algorithm. For the decomposing rules we analyze only the second component computed by ϕ since these rules do not eliminate variables. Whereas for the solving rules we analyze only the first component. Let ϕ_1 and ϕ_2 denote the two components of ϕ.

($\mathbf{D_1}$) $\phi_2(l) = symb(S_1) + symb(S_2) + symb(M) = (n-1) + symb(s_1^1) + \ldots + symb(s_1^n) + (m-1) + symb(s_2^1) + \ldots + symb(s_2^m) + symb(M)$
$\phi_2(r) = max\{symb(s_1) + symb(s_2) + symb(S_1 \setminus \{s_1\}) + symb(S_2 \setminus \{s_2\}) \mid s_1 \in S_1,\ s_2 \in S_2\} = (n-2) + symb(s_1^1) + \ldots + symb(s_1^n) + (m-2) + symb(s_2^1) + \ldots + symb(s_2^m) + symb(M)$

($\mathbf{D_2}$) $\phi_2(l) = symb(S) + symb(\emptyset) + symb(M)$
$\phi_2(r) = symb(\mathbf{false}) = 0$

($\mathbf{D_3}$) $\phi_2(l) = symb(\emptyset) + symb(N_2) + symb(M) > 1 + symb(M)$
$\phi_2(r) = symb(M)$

($\mathbf{D_4}$) $\phi_2(l) = symb(\emptyset) + symb(S) + symb(M) > 1 + symb(M)$
$\phi_2(r) = symb(M)$

($\mathbf{D_5}$) $\phi_2(l) = symb(\emptyset) + symb(P) + symb(M) > 0$ $\phi_2(r) = symb(\mathbf{false}) = 0$

($\mathbf{D_6}$) $\phi_2(l) = symb(id_1 : nname_1 : P_1) + symb(id_2 : nname_2 : P_2) + symb(M) = 1 + symb(id_1) + symb(nname_1) + symb(P_1) + 1 + symb(id_2) + symb(nname_2) + symb(P_2) + symb(M)$
$\phi_2(r) = symb(id_1) + symb(nname_1) + symb(P_1) + symb(id_2) + symb(nname_2) + symb(P_2) + symb(M)$

($\mathbf{D_7}$) $\phi_2(l) = symb(id_1^\frown S_1 \lessdot id_2^\frown S_2) + symb(M) = 1 + symb(id_1) + symb(S_1) + 1 + symb(id_2) + symb(S_2) + symb(M)$
$\phi_2(r) = symb(id_1) + symb(S_1) + symb(id_2) + symb(S_2) + symb(M)$

($\mathbf{D_8}$) $\phi_2(l) = symb((p_1, p_2) \lessdot (p_3, p_4)) + symb(M) = 1 + symb(p_1) + symb(p_2) + 1 + symb(p_3) + symb(p_4) + symb(M)$
$\phi_2(r) = symb(p_1) + symb(p_2) + symb(p_3) + symb(p_4) + symb(M)$

($\mathbf{D_9}$) $\phi_2(l) = symb(a \lessdot b) + symb(M) = symb(a) + symb(b) + symb(M)$
$\phi_2(r) = max\{0, symb(M)\}$

($\mathbf{S_{1-3}}$) $\phi_1(l) = \phi_1(r) + 1$.

In all cases above, indeed $\phi(l) > \phi(r)$.

For the cleaning rule ($\mathbf{C_1}$), the number of symbols of the second argument of a matching term is strictly decreasing, while the rule ($\mathbf{C_2}$) decreases the number of components in a disjunction of matching terms

□

Theorem 4. *Any matching term $M \wr E_1 \star E_2 \int N \rhd \sigma$ has a unique normal form with respect to the rules ($\mathbf{D_{1-9}}$), ($\mathbf{S_{1-3}}$), ($\mathbf{C_{1-2}}$) consisting of a disjunction of matching terms in normal form.*

If the normal form of the matching term $N_1 \lessdot N_2 \wr E_1 \star E_2 \int N \rhd \sigma$ corresponding to the submatching problem $(N_1, E_1) \lessdot (N_2, E_2)$ has the form:

1. $\mathbf{false} \wr E_1 \star E_2 \int N \rhd \sigma$, *then the submatching problem does not have solutions;*
2. $\bigvee_{i=1,n} \mathbf{true} \wr \emptyset \star E_i \int N_i \rhd \sigma_i$, $n \geq 1$, *then each tuple (σ_i, N_i, E_i), for $i = 1, n$, is a pre-solution of the submatching problem.*

Proof. We proved the termination of the set of rules in Proposition 1, hence a normal form always exists. The decomposition rules are direct consequences of the decomposition rules for syntactic matching and matching modulo associativity, commutativity and

neutral element. Then the application of each decomposition rule does not introduce unexpected solutions nor looses any solution. By construction the solving rules simulate a traversal of the pattern port graph.

In the following we prove that each of the matching rules preserves the set of solutions of the submatching problem. Let us consider a one-step rewriting of a generic matching term using the submatching algorithm:

$$M \wr E_1 \star E_2 \int N \rhd \sigma \rightarrow \bigvee_i M' \wr E_1^i \star E_2^i \int N' \rhd \sigma'$$

The application of a matching rule preserves the set of solutions for the submatching problem if there exists a substitution σ_o such that:

1. $\sigma' = \sigma_o \cup \sigma$,
2. $E_2 \setminus \sigma_o(E_1) = E_2^i \setminus E_1^i$ for all i, and
3. $N \cup \mathit{rhsNodes}(M) \setminus \sigma_o(\mathit{lhsNodes}(M)) = N' \cup \mathit{rhsNodes}(M') \setminus \mathit{lhsNodes}(M')$

where *lhsNodes* and *rhsNodes* return the sets of nodes from the left-hand side and right-hand sides of all matching equations in conjunction respectively.

We do not consider the rules involving a matching failure since the submatching problem involved has no solution.

The rules $(\mathbf{D_1})$, $(\mathbf{D_4})$, $(\mathbf{D_6})$, $(\mathbf{D_7})$, $(\mathbf{D_8})$, $(\mathbf{D_9})$, $(\mathbf{C_1})$ do not modify the sets of nodes in the matching equations and σ_o is empty. The rule $(\mathbf{D_3})$ is the only one to act on the fourth component of a matching term, by appending to it the set of nodes subject in a matching equation where the pattern is the empty set; hence the condition 3 above is verified. For a rewriting step using the rule $(\mathbf{D_3})$, the substitution σ_o is also empty.

For each of the solving rules $(\mathbf{S_1})$, $(\mathbf{S_2})$, and $(\mathbf{S_3})$, the substitution σ_0 consists of one mapping, $\{x \mapsto id\}$, $\{x \mapsto nname\}$, and $\{x \mapsto p\}$ respectively and the condition 3 on nodes is verified since M' is different from $\sigma_0(M)$ by a matching equation that is not considered in computing *lhsNodes* and *rhsNodes*, i.e., $\mathit{lhsNodes}(M') = \mathit{lhsNodes}(\sigma_0(M))$.

All matching rules except the cleaning rule $(\mathbf{C_1})$ and the rules involving a matching failure verify by construction the condition 2 on the preservation of solutions since $\sigma_0(E_1) = E_1^i$ and $E_2 = E_2^i$, for all i. Then cleaning rule $(\mathbf{C_1})$ removes common parts from E_1 and E_2, where $\sigma_o(E_1) = E_1$, hence the difference specified by the condition 2 remains constant after an one-step rewriting.

In conclusion, the application of the matching rules is terminating and solution preserving, which proves the theorem. □

3.5 Port Graph Rewrite Rules

Definition 48 (Port graph rewrite rule I)**.** *A* port graph rewrite rule *is given by two port graphs L, R over a p-signature $\nabla^{\mathcal{X}}$ and a node-morphism $\xi : V_L \rightarrow \mathcal{P}(V_R)$, denoted by $L \Rightarrow R$ or $L \overset{\xi}{\Rightarrow} R$, such that $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(L)$ and $Id(R) \subseteq Id(L)$. The port graphs L and R are called the left- and right-hand side of the rule respectively, and ξ the correspondence morphism.*

Remark 5. In the context of the definition above we informally present in the following some particular operations the rules may perform on nodes:

1. if there is a node $v \in V_L$ with $\xi(v) = \{v\}$, then v is not modified by the rule;
2. if there is a node $v \in V_L$ with $\xi(v) = \emptyset$ then the node v is *deleted* by the rule;
3. if there is a node $v \in V_L$ with $\xi(v) = \{v_1, v_2\}$, then the rule *duplicates* or *splits* the node;
4. if there is a node $u \in V_R$ such that $\xi^{-1}(\{u\})$ is defined and $\xi^{-1}(\{u\}) = \{v_1, \ldots, v_k\}$ and $\xi(v_i) = \{u\}$, for all $i = 1, .., k$, it means that the rule *merges* the nodes $v_1, \ldots, v_k$ into u;
5. if there is a node $u \in V_R$ and $v_1, \ldots, v_k$ are all the nodes in V_L, with $k \geq 2$, such that $u \in \xi(v_1) \cap \ldots \cap \xi(v_k)$ and there exists i, $1 \leq i \leq k$ such that v_i splits up, then the rule *partially merges* the nodes $v_1, \ldots, v_k$ into u.

We can represent a port graph rewrite rule as a port graph if we define a node encoding the node-morphism ξ giving the correspondence between the nodes of L and the nodes of R:

Definition 49 (Port graph rewrite rule II)**.** *A* port graph rewrite rule $L \Rightarrow R$ *is a port-graph consisting of:*

- *two port-graphs L and R over a p-signature $\nabla^{\mathcal{X}}$ called, as usual, the left- and right-hand side respectively, such that $\mathcal{V}ar(R) \subseteq \mathcal{V}ar(L)$ and $Id(R) \subseteq Id(L)$,*
- *one special "arrow" node, $\Rightarrow$, that has a port for each port in L which is preserved in R, and the* black hole *port, named* bh*,*
- *edges from L to $\Rightarrow$ mapping each port in L uniquely to a port of $\Rightarrow$ if it is not deleted, and to* bh *otherwise,*
- *edges from $\Rightarrow$ to R, mapping each port different* bh *to a port in R.*

such that the correspondence of a port from L to ports from R is expressed by the edges incident to the arrow node.

Expressing a port graph rule as a port graph is very well-suited for graphical representation. However, for studying the properties of the port graph rewriting relation, in this section we use the first definition for port graph rewrite rules given by Definition 48.

Definition 50 (Port graph rewrite system)**.** *A* port-graph rewrite systems *(over a p-signature $\nabla^{\mathcal{X}}$) is a finite set of port-graph rewrite rules (over a p-signature $\nabla^{\mathcal{X}}$).*

Example 13. We illustrate in Figure 3.4 four port-graph rewrite rules: (a) decomposing a node with two ports both connected to the same port of another node into two nodes, each with a port, (b) deleting a node and its ports, (c) deleting two edges, and (d) merging two nodes.

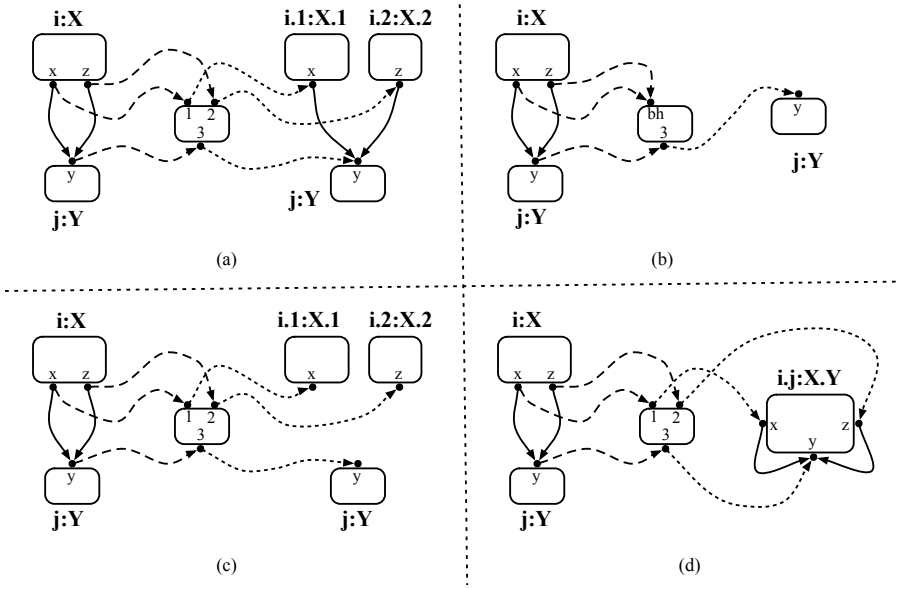


Figure 3.4: Port-graph rewrite rules

3.6 Port Graph Rewriting Relation

Let $L \Rightarrow R$ be a port graph rewrite rule and G a port graph such that there is a submatching morphism g for L in G.

The rewriting step corresponding to applying $L \Rightarrow R$ to G can be decomposed in the following smaller steps as illustrated in Figure 3.5:

1. find the submatching morphism g, the context G^-, and the bridges $\mathcal{B}$ (their orientation is not important in the drawing),
2. identify the left-hand side of the rule with an isomorphic subgraph in G,
3. translate the bridges following the arrow node $\Rightarrow$,
4. remove the arrow node and the isomorphic subgraph $g(L)$ which is now disconnected from the context graph.

The delicate point of applying $L \Rightarrow R$ to G is to properly define the replacement of $g(L)$ by $g(R)$ in G and the way $g(R)$ is reconnected with G.

Usually in graph transformation the matching must ensure a *dangling condition* which guarantees that the endpoints of edges in G^- are preserved (not deleted) by the rule application. A bridge or an unmatched edge is not dangling if the endpoint v from $g_V(L_V)$ is not deleted by the application, i.e., $\xi(v) \neq \emptyset$.

If a bridge of an unmatched edge has an endpoint in $g(L)$ that is changed by the rewrite rule, then it cannot be simply reconnected to $g(R)$ during the replacement. However,

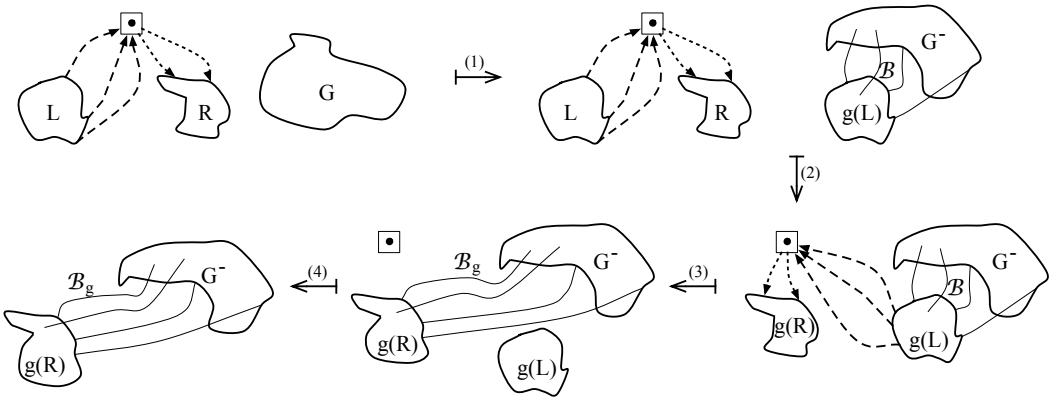


Figure 3.5: Rewriting steps

by extending the node-morphism ξ on edges provides a way of reconnecting; we call this operation *arrow-translation* and it is needed for bridges or unmatched edges. We denote by $\Downarrow e$ the translation of an edge which is nothing else but $\xi(e)$. Thanks to the concept of node-morphism and its extension on edges we allow dangling edges in our framework of rewriting port graphs.

If we consider the second definition of the port graph rewrite rule, the arrow node $\Rightarrow$ together with the incident edges trace the change of the endpoints of bridges or unmatched partial nodes. The arrow-translation operation takes an arrow node $\Rightarrow$ together with all incident edges and an edge $\langle (v,p),(u,p')\rangle$ such that one endpoint of the edge, let this be (u,p'), is the source of an edge (p',r) incident to $\Rightarrow$, and it returns the set of edges $\langle (v,p),(w_i,p_i')\rangle$ for (w_i,p_i') a target of the edge with the source r in the arrow node. In Figure 3.6 we illustrate a graphical representation of the arrow-translation operation.

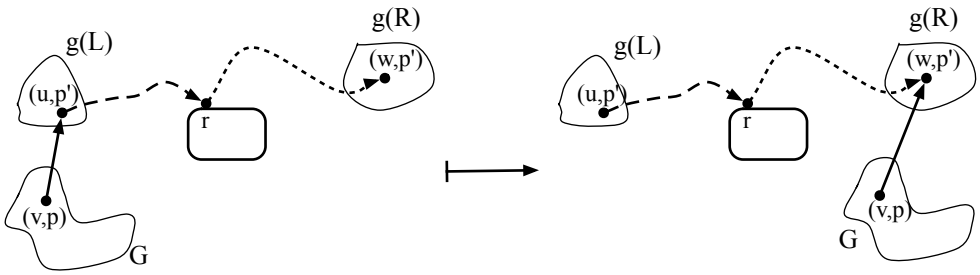


Figure 3.6: An arrow-translation example

Definition 51. *Given a port graph rewrite system $\mathcal{R}$, a port graph G* rewrites *to a port graph G', denoted by $G \Rightarrow_{\mathcal{R}} G'$, if there exist:*

- *a port graph rewrite rule $L \Rightarrow R$ in $\mathcal{R}$ and*

- *a solution* $(g, G^-, \mathcal{B})$ *of the submatching problem* $L \prec\!\!\!\!\cdot\, G$

such that $G = G^-[g(L)]_{\mathcal{B}}$ *and* $G' = G^-[g(R)]_{\Downarrow_g \mathcal{B}}$*, where by* $\Downarrow_g \mathcal{B}$ *we denote the arrow-translation of bridges using the arrow node and its incident edges instantiated by the submatching morphism* g*.*

We note that this definition gives the condition for rewriting a port graph using a rewrite rule, as well as the construction of the resulting port graph.

We can now define the *application of a matcher* $\sigma = (g, G^-, \mathcal{B})$ resulted from a submatching problem $L \prec\!\!\!\!\cdot\, G$ on R as $\sigma(R) = G^-[g(R)]_{\Downarrow_g \mathcal{B}}$.

If we consider in Definition 51 a non-strict port-graph morphism g, then a new set of edges is identified, the set of unmatched edges $\mathcal{U}_e$. Consequently, we have $G = G^-[g(L)]^{\mathcal{U}_e}_{\mathcal{B}}$ and $G = G^-[g(R)]^{\Downarrow_g \mathcal{U}_e}_{\Downarrow_g \mathcal{B}}$.

Example 14. We illustrate in Figure 3.7 a result of the application of the rule given in Figure 3.4 on the port graph G based on the following decomposition:

- the submatching morphism: $g(i) = 1$, $g(j) = 2$, $g(X) = A$, $g(Y) = B$, $g(x) = a$, $g(z) = b$, $g(y) = e$,
- the context graph G^- consists of the nodes identified by 3 and 4 and the edge $\langle (3, a), (4, d) \rangle$, and
- the bridges $\langle (1, b), (3, a) \rangle$ and $\langle (4, d), (1, b) \rangle$.

Then both bridges are modified by the arrow-translation because they both have an endpoint belonging to a node that is split up by the rule application.

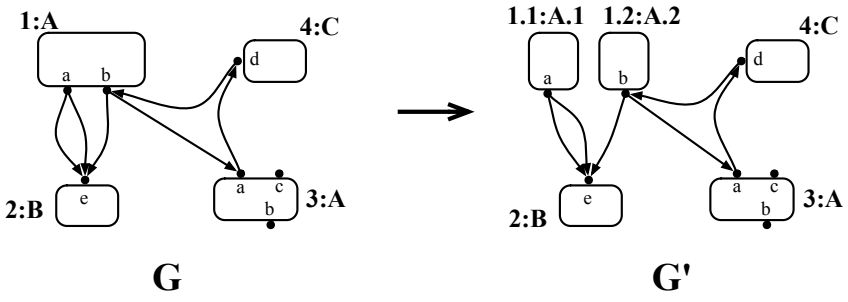


Figure 3.7: An application of the port graph rewrite rule from Figure 3.4 (a) on a port graph G resulting in a port graph G'

3.7 Strategic Port Graph Rewriting

Based on the definition of abstract strategic rewriting from Chapter 1, we can formalize strategic port graph rewriting.

Definition 52 (Strategic port graph rewriting)**.** *Given an abstract reduction system* $\mathcal{A} = (\mathcal{O}_{\mathcal{R}}, \mathcal{S}_{\mathcal{R}})$ *generated by a port graph rewrite system* $\mathcal{R}$*, and a strategy* ζ *of* $\mathcal{A}$*, a* strategic port graph rewriting derivation *(or* port graph rewriting derivation under strategy ζ*) is an element of* ζ*.*

A strategic port graph rewriting step under ζ *is a rewriting step* $G \rightarrow_{\mathcal{R}} G'$ *that occurs in a derivation of* ζ*.*

The idea of imposing some sequencing conditions on the application of graph transformation can be traced back to the programmed graph grammars introduced in [Bun79] which consider imperative control structures. Later, the authors of PROGRES not only considered these imperative deterministic control structures for applying graph transformations, but they defined them in a Prolog-like style using depth-first search and backtracking [Sch97, SWZ97] in order to control the non-deterministic choice of rule application.

3.8 Weak Port Graphs

Sometimes we only have some partial information about the ports of nodes; or, for the sake of simplicity, we consider in a port graph rewrite rule only the ports relevant for the transformation. We introduce a restriction of port graphs where only a subset of the ports in the interface of a node name is considered. We call such subset *RelevantInterface* and the port graphs are called *weak port graphs*. In the following we adjust accordingly the definitions of morphisms and rewriting relation on weak port graphs.

Definition 53 (Weak port graph)**.** *A* weak port graph G *over a p-signature* ∇ *is a port graph with the following restriction: the port set (interface) of each node* $v \in V_G$ *is a subset of* $Interface(lv_G(v))$*. We denote by* $RelevantInterface(v)$ *the port set associated to each node* $v \in V_G$*.*

Definition 54 (Weak port graph morphism)**.** *A* weak port graph morphism *is a port graph morphism* $f : G \rightarrow H$ *with a weak condition on the equality of interfaces of two mapped nodes:*

$$\textit{for each } v \in V_G,\ RelevantInterface(lv_G(v)) \subseteq RelevantInterface((f_V \circ lv_H)(v))$$

Following closely the definition in Section 3.3, the weak counterpart definitions can be easily deduced, in particular, the notions of *weak port graph rewrite rule*, *weak subgraph*, and *weak submatching*.

A weak port-graph morphism implies the existence of a set of unmatched partial node interfaces, correspondingly a set of *unmatched partial nodes* $\mathcal{U}_n$:

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

Let $L \Rightarrow R$ be a weak port graph rewrite rule weakly submatching a port graph G using the following decomposition:

$$G = G^{-}[g(L) \cup \mathcal{U}_n]_{\mathcal{B}}$$

Then the result of applying the rewrite rule based on this submatching has the form:

$$G' = G^-[g(R) \cup V]_{\Downarrow_g \mathcal{B}} \quad \text{where} \quad V \in \Downarrow_g \mathcal{U}_n$$

since the application of the arrow-translation may have give different choices of destination of unmatched ports.

Obviously, for a weak non-strict port-graph morphism, we have:

$$G = G^-[g(L) \cup \mathcal{U}_n]^{\mathcal{U}_e}_{\mathcal{B}} \quad \text{and} \quad G' = G^-[g(R) \cup V]^{\Downarrow_g \mathcal{U}_e}_{\Downarrow_g \mathcal{B}} \quad \text{where} \quad V \in \Downarrow_g \mathcal{U}_n$$

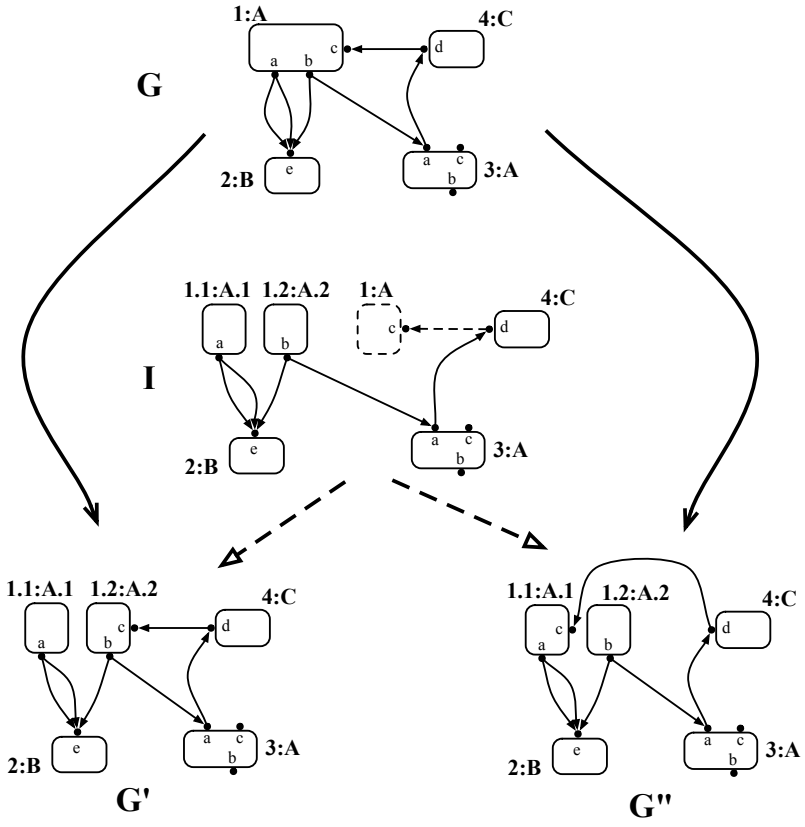


Figure 3.8: An application of the port graph rewrite rule given in Figure 3.4 (a) on G resulting in two port graphs G' and G'' with an intermediate port graph I

Example 15. In Figure 3.8 we illustrate a port graph resulting from rewriting G (also given in Figure 6.1) using the rule given in Figure 3.4 (a). The resulting port graphs G' and G'' are obtained by splitting the node 1, choosing to place the unmatched port c in 1.1 and in 1.2 respectively, and then redirecting the two bridges $\langle (4, d), (1, c) \rangle$ and $\langle (1, b), (3, a) \rangle$ to 1.1 and 1.2 respectively. In the intermediate step we emphasize the

incidence of the node 4 to the unmatched partial node 1 with the the port set $\{c\}$. The node-substitution may map this partial node to either of the two resulting nodes 1.1 and 1.2. Therefore two solutions are possible.

3.9 On the Confluence of Port Graph Rewriting

In this section we analyze the confluence property for port graph rewriting systems based on the approach used in [Plu05] for hypergraph rewriting systems. First we introduce the category of port graphs where we define the concept of direct derivation corresponding to the one-step port graph rewriting. Then we give informally the conditions that port graph rewriting systems must satisfy in order to be locally confluent.

Let **PGraph** be the category having labeled port graphs over a fixed p-signature ∇ as objects and port graph morphisms and node-morphisms as arrows.

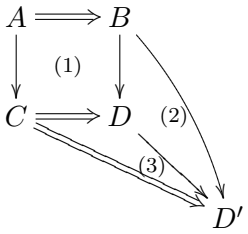
We define a particular pushout in the category **PGraph** based on the classical definition of pushout from [Mac98].

Definition 55 (Alternating pushout)**.** *Given a node-morphism $A \Rightarrow B$ and a port-graph morphism $A \to C$, a port graph D together with a port graph morphism $B \to D$ and a node-morphism $C \Rightarrow D$ is an* alternating pushout *of $A \Rightarrow B$ and $A \to C$ if the following conditions are satisfied:*

Commutativity *$A \Rightarrow B \to D = A \to C \Rightarrow D$ or, diagrammatically, the following diagram commutes:*

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

Universality *For all port graphs D', port graph morphism $B \to D'$ and node-morphism $C \Rightarrow D'$ such that $A \Rightarrow B \to D' = A \to C \Rightarrow D'$, there is a unique port graph morphism $D \to D'$ such that $B \to D \to D' = B \to D'$ and $C \Rightarrow D \to D' = C \Rightarrow D'$ (i.e., the diagrams (2) and (3) below commute respectively).*



Remark 6. An alternating pushout in **PGraph** is a pushout if we view the port graph morphisms as node-morphism mapping a node to the singleton containing its image.

The *construction of an alternating pushout* is based on the set-theoretical construction of pushouts as given in [Ehr79] . Let $\xi : A \Rightarrow B$ be a node-morphism and $g : A \to C$ an injective port graph morphism. Then an alternating pushout

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

can be constructed as follows:

- $V_D = (V_B + V_C)/_{\approx}$ where $\approx$ is the equivalence relation generated by the relation $\sim$ defined by $g(v) \sim \xi(v)$ for all $v \in V_A$. Then, for $u \in V_B + V_C$, the equivalence class of u with respect to $\approx$ is defined as:

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

  Remark that the first case covers as well the situation of a node $u \in V_{g(A)}$ since there exists a node $v \in V_A$ such that $g(v) = u$, and $\xi(v) = \{v_1, \ldots, v_n\} \subseteq V_B$. Hence $[u] = \{[v_1], \ldots, [v_n]\}$.

  In other words, for $u \in V_A$, the equivalence $g(v) \sim \xi(v)$ implies that the representative of the equivalence class of $g(v)$ and $\xi(v)$ is $(g \circ \xi)(v)$ defined in Section 3.3.

- $E_D = (E_B + E_C)/_{\approx}$ where $\approx$ is the equivalence relation generated by the relation $\sim$ defined by $g(e) \sim \xi(e)$ for all $e \in E_A$. Then for $e \in E_B + E_C$:
  - if $e \in E_B$, then $[e] = e$,
  - if $e \in E_C$, $s_C(e) \in V_{g(A)}$, $t_C(e) \notin V_{g(A)}$, and $\xi(g^{-1}(s_C(e))) = \emptyset$, then $[e]$ is not defined;
  - if $e \in E_C$, $t_C(e) \in V_{g(A)}$, $s_C(e) \notin V_{g(A)}$, and $\xi(g^{-1}(t_C(e))) = \emptyset$, then $[e]$ is not defined;
  - if $e \in E_C$, $s_C(e) \in V_{g(A)}$, $t_C(e) \notin V_{g(A)}$, and $\xi(g^{-1}(s_C(e))) = \{v_1, \ldots, v_k\}$, then $[e] = \{e_1, \ldots, e_k\}$ where $s_D(e_i) = g(v_i)$ and $t_D(e_i) = t_C(e)$, for all $i = 1, k$;
  - if $e \in E_C$, $t_C(e) \in V_{g(A)}$, $s_C(e) \notin V_{g(A)}$, and $\xi(g^{-1}(t_C(e))) = \{v_1, \ldots, v_k\}$, then $[e] = \{e_1, \ldots, e_k\}$ where $t_D(e_i) = g(v_i)$ and $s_D(e_i) = s_C(e)$, for all $i = 1, k$.

- $\xi^* : C \Rightarrow D$ and $g^* : B \to D$ are the node and port graph morphism respectively sending each element to its equivalence class, that is, $\xi^*(x) = [x]$ and $g^*(x) = [x]$ separately for nodes and edges.

Definition 56 (Derivation)**.** *Let G and H be two port graphs, $L \overset{\xi}{\Rightarrow} R$ a port graph rewrite rule, and $g : L \to G$ an injective port graph morphism. Then G* directly derives *H by ξ and g, denoted by $G \Rightarrow_{\xi,g} H$ if the following alternating pushout exists:*

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

A derivation *from a port graph G to a port graph H, denoted by $G \Rightarrow^* H$, is a sequence of direct derivations $G = G_0 \Rightarrow G_1 \Rightarrow \ldots \Rightarrow G_n = H$ for some $n \geq 0$.*

We remark that the construction given for the alternating pushout above corresponds to a port graph rewriting of G into H using the port graph rewrite rule $L \overset{\xi}{\Rightarrow} R$ and the submatching morphism g.

Corollary 1 (Correspondence between rewrite relation and direct derivation)**.** Let G and H be two port graphs, $L \overset{\xi}{\Rightarrow} R$ a port graph rewrite rule, and $g : L \rightarrow G$ an injective port graph morphism. Then G rewrites to H using ξ and g if and only G directly derives H by ξ and g.

Informally, we say that two direct derivations are *parallel independent* if and only if the left-hand sides of the two rules may overlap in G only on elements not changed by either rule. Then, no matter the order of sequential application of the two direct derivations on a port graph, the resulting port graphs are the same, i.e., if $\Delta_i : G \Rightarrow_{\xi_i, g_i} H_i$, $i = 1, 2$ are independent direct derivations, then there exists a port graph H such that $H_1 \Rightarrow_{\xi_2, g_2} H$ and $H_2 \Rightarrow_{\xi_1, g_1} H$. This result concerning the commutativity of the application of two parallel independent direct derivations is proved in [Plu05] for hypergraphs.

Let $L_i \overset{\xi_i}{\Rightarrow} R_i$, $i = 1, 2$, be two port graph rewrite rules. The two one-step rewrites (or direct derivations) $G \Rightarrow_{\xi_i, g_i} H_i$, $i = 1, 2$, with $g_1 \neq g_2$ if $\xi_1 = \xi_2$ are *independent* if the intersection of the left-hand sides of r_1 and r_2 in G consists of elements not changed by the rule application. If the two steps are independent and $G = g_1(L_1) \cup g_2(L_2)$, then the pair of the two rewrites is a *critical pair*. A critical pair $H_1 \Leftarrow G \Rightarrow H_2$ is *joinable* if there exist two isomorphic port graphs I_1 and I_2 such that $H_i \Rightarrow^* I_i$.

The joinability of all critical pairs of a port graph rewrite system does not guarantee local confluence (as it does for term rewrite systems). In the following we illustrate a situation inspired from [Plu05] when the joinability condition of a critical pair does not suffice for assuring local confluence. Let us consider the port graph rewrite rule given in Figure 3.9 where the node identifiers, node names, and ports are all variables.

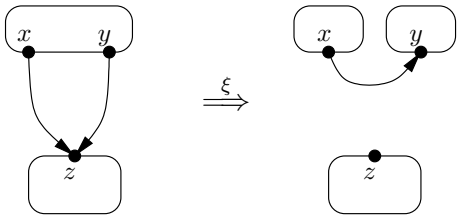


Figure 3.9: A port graph rewrite rule

Then the critical pair from Figure 3.10 is clearly joinable since the two resulting port graphs are isomorphic.

But when we embed such critical pair into a context, we may loose the isomorphism, hence the local confluence, as illustrated in Figure 3.11. The problem is that a node or

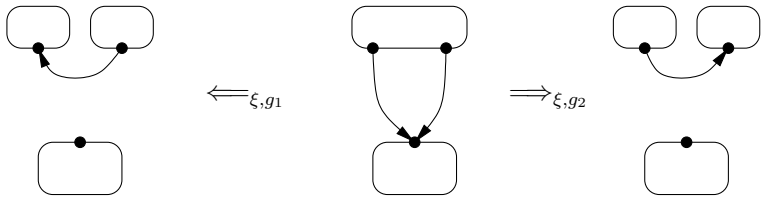


Figure 3.10: Example of critical pair produced by the port graph rewrite rule in Figure 3.9

port endpoints of a bridge and preserved by both derivations correspond to two nodes or ports which cannot be related by an isomorphism.

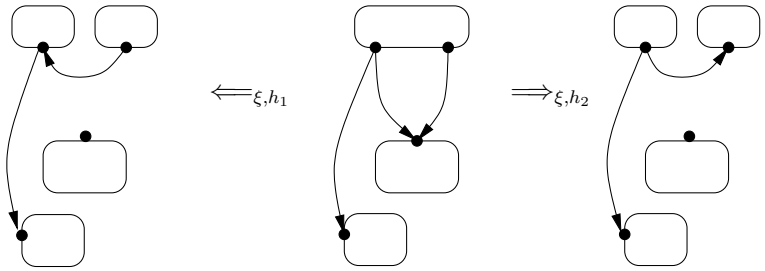


Figure 3.11: Example of embedding the critical pair from Figure 3.10 into a context that does not preserve local confluence

Then the joinability condition of a critical pair $H_1 \Leftarrow_{\xi,g_1} G \Rightarrow_{\xi,g_2} H_2$ must be strengthen such that for each node v in G preserved by both direct derivations as the same set of nodes $\{v_1, \ldots, v_n\}$, $n \geq 1$, there are derivations $H_1 \Rightarrow^* I_1$ and $H_2 \Rightarrow^* I_2$ and an isomorphism $f : I_1 \to I_2$ such that the set $\{v_1, \ldots, v_n\}$ has a decomposition into V' and V'' with

- all nodes in I_1 and I_2 derived from the nodes in V' are related by the isomorphism f, and
- all nodes from V'' are deleted during the derivations.

This condition is called *strong joinability*.

Then the results on the local confluence and confluence proved for hypergraph rewriting system in [Plu05] can be also used for port graphs rewriting systems, as well as for weak port graph rewriting systems.

Lemma 2 (Confluence)**.** A port graph rewriting system is locally confluent if all its critical pairs are strongly joinable. If in addition the port graph rewriting system is terminating, then it is confluent.

3.10 Comparison with Bigraphical Reactive Systems

Bigraphical reactive systems (BRSs) are a graphical model of computation where connectivity as well as locality are important [Mil01, JM04, Mil06]. A *bigraph* consists of a forest of trees describing the positions of the computational objects in the system, called *place graph*, and a hypergraph describing the links or connections between the objects, called *link graph*. Then a reaction rule in a BRS is pair of bigraphs, a redex represented by a pattern of nesting and linkage, and a reactum indicating how the reaction changes the pattern. Via such reaction rules, BRSs represent a suitable formalism for modeling mobile connectivity (by changing the link graph) and mobile locality (by changing the place graph).

An object in a bigraph is represented by a node with a fixed number of ports for connecting with other nodes given by an associated *control*. The set of all possible controls for the nodes of a bigraph represents its *signature*. In addition, since the locality and connectivity aspects are independent, *nodes can be nested* and the ports are linked independently of the nesting.

Port graphs are a variation of bigraphs where: the nodes are labeled with controls of fixed arity but (corresponding to the set of ports), the controls are all atomic (no nesting), the link graph is just a graph, not hypergraph (an edge connects exactly two ports of nodes). However, the ports in port graphs do not have a fixed incidence degrees, whereas for bigraphs the incidence degree of a port is of 1 or 0 (when the port is free).

The application of a reaction rule on a bigraph is based on decomposing the bigraph into the redex of the reaction rule, a context, and some discrete parameter, and then replacing the redex by the reactum. The problem of finding such decomposition, called matching, is tackled in [BDGM07] by using a class of basic bigraphs and operators for decomposing bigraph and then, based on such decompositions for both the redex and the bigraph to be transformed, by identifying inductively the redex and the context. The ways the matching problem is treated for bigraphs and for port graphs seem equivalent modulo the additional expressivity of the bigraphical formalism: the matching problem is reduced in both cases to finding a redex, a context, a parameter; the functionality of such a parameter is similar to the notion of the set of bridges used for port graphs. In consequence, by investigating more closely the decomposition-based matching algorithm provided for bigraphs might provide a more efficient, if not more elegant, matching algorithm for port graphs. However, we have defined more expressive reaction rules for port graphs which specify splitting a node, whereas in bigraphical systems we can only duplicate a node. It should be interesting to investigate what modification such a splitting operation would imply in the categorical model of bigraphs.

Based on the existing works on bigraphs and bigraphical reaction systems, we could improve the mathematical theory of port graphs using category theory, as well as, borrow techniques for defining extensions for port graphs, like, for instance, from the stochastic bigraphs [KMT08].

We have identified the structure of port graphs independently from the already existing bigraphs, in the context of modeling protein interactions, hence as a formalism based on the necessary information for modeling such biochemical processes. Adding the nesting

capabilities to nodes as for bigraphs, would then allows us to model other types of biological interactions; take, for instance, the example of membrane budding where molecules living inside a cell are transported to other cells which is already formalized in the context of stochastic bigraphs [KMT08]. Nonetheless, other interesting examples motivating the necessity of adding nesting to port graphs and not needing all specific features of bigraphs should be envisaged.

3.11 Conclusions and Perspectives

In this chapter we have formalized the structure of port graphs, the (sub)matching problem on port graphs, and a port graph rewriting relation. We have also analyzed the confluence the port graph rewriting relation based on some classical results on hypergraphs. The submatching and the rewriting relation defined for port graphs in this chapter are then used for instantiating the Abstract Biochemical Calculus in the next chapter to obtain a rewriting calculus for port graphs.

There is still work to be done in formalizing the mathematical foundations for port graph rewriting using elements of category theory, in particular to analyze the suitability of the port graph structure as a high-level structure in the context of High-Level Replacement Systems. Then, we should test which HRL-conditions are satisfied in the category of port graphs, such that we could benefit from the existing results in HLR systems, for instance on parallelism.

Recently, Lambers, Ehrig and Tanetzer [LET08] established some sufficient criteria for the applicability and non-applicability of sequences of graph transformation rules on a graph without trying to apply the whole rule sequence explicitly. In particular, this work fits in the context of finding properties of the strategic port graph rewriting, where the sequence combinator on port graph rewrite rule is a basic one. What needs to be done is to study the approach from [LET08] on graphs, adapt it for sequences of port graph rewrite rules, and then for other port graph rewrite strategies. Moreover, the problem of finding applicability criteria for rewrite strategies is also of great interest in the term rewriting community.

Another future direction, a more pragmatic one, concerns the (sub)matching problem on port graphs and finding a matching algorithm as much as efficient as possible. One possible solution is to relate port graph matching to bigraph matching; however, port graphs have a much more simpler structure than the bigraphs, hence a matching algorithm for bigraphs could be sometimes not very efficient for port graphs. Also some general algorithms for graph pattern matching could be considered [Ull76, ST99, LV02, Val02, Zün96]. Due to the structure of port graphs which is richer than the structure of a graph and, in the same time, is a particular kind of bigraph, it seems that a matching algorithm for port graphs will be situated somewhere in between classical matching algorithms for graphs and matching algorithms for bigraphs.

As already mentioned during the comparison with the bigraphical reactive systems, we should consider the stochastic dimension for port graph rewriting systems as it was done for bigraphs [KMT08]. This perspective comes naturally since port graphs are

biologically-inspired, and hence, their applications to modeling biological systems should consider stochastic aspects.

4 The ρ_{pg}-Calculus: a Biochemical Calculus Based on Strategic Port Graph Rewriting

4.1 Introduction

In Chapter 2 we have introduced an Abstract Biochemical Calculus, the $\rho_{\langle\Sigma\rangle}$-calculus, modeling interactions between abstract molecules over a structure described by the objects of a category Σ. In this chapter we instantiate the $\rho_{\langle\Sigma\rangle}$-calculus by **PGraph**: we consider the port graph structure and the port graph rewriting relation defined in Chapter 3 for the abstract molecules and their interactions respectively. The result is a port graph rewriting calculus, called the ρ_{pg}-calculus. The syntax and the semantics is identical. The structure of port graph brings more expressivity in the representation of the objects of the calculus as port graphs. As a consequence, we are able to define evaluation rules for matching and replacement using appropriate nodes and port graph transformations.

In the following we present the main features of the syntax (Section 4.2) and the semantics (Section 4.3) of the calculus brought by the structure of port graphs.

4.2 Syntax

We introduce stepwise all the elements defining the syntax of the calculus. We start by defining the object port graphs as port graphs and the abstractions (port graph rewrite rules) as port graphs as well. All entities of the calculus are presented graphically this time.

Nodes

A node is described by a unique identifier, a name, and a set of ports. The node identifiers are either integers, lowercase letters from i to n, or concatenations of them. The node names are strings in uppercase that can be indexed by or concatenated with integers, and concatenations of names. The names of the ports are strings in lowercase possibly indexed by integers. Let id, $pname$, and $nname$ range over the sets of node identifiers, port names, and node names.

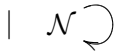
The syntax for ports and nodes is the following:

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

A port is a point with a port name. A node is either a variable or an entity with an unique identifier, a node name, and a set of ports. Graphically a node is represented as a box with points on the surface corresponding to the ports.

Object Port Graphs

We model the states of a system by port graphs called *object port graphs* which can be either a variable, an empty port graph, a node, two juxtaposed object port graphs, a node with a loop on the same port, a node with an edge connecting two different ports, or two connected object port graphs:

Objects Port Graphs	$\mathcal{O}$	::= $\mathcal{X}_{\mathcal{O}}$	variables
		| ε	empty graph
		| $\mathcal{N}$	node
		| $\mathcal{O}\ \mathcal{O}$	juxtaposition
		| 	loop
		| 	self-connection
		| 	connection

Abstractions

A first-order abstraction in the ρ_{pg}-calculus consists of two object port graphs for the left- and the right-hand sides, and an *arrow node* embedding the correspondence between the two sides. The arrow node has two particular ports, a *handler port* p_0 and a *black hole* port port bh. Some restrictions must be imposed on the connectivity of these two types of ports. A handler port can be connected only to another handler port, and a black hole port can only be the target of an edge whose source is neither a handler port nor a black hole port.

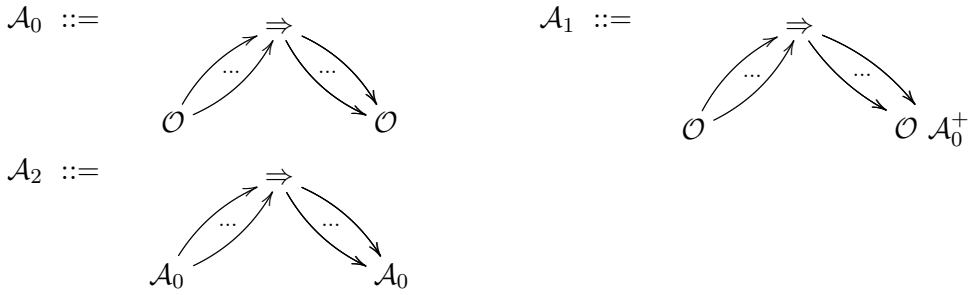
Based on Definition 49, a first-order abstraction from $\mathcal{A}_0$ is a port graph constructed as follows:

- the arrow node has a port for each port in the left-hand side which is preserved in the right-hand side, the black hole port and the handler port;
- an edge connects each preserved port p from the left-hand side to their corresponding port of the arrow node which in turn connects to the corresponding ports of p in the right-hand side;
- the deleted ports from the left-hand side are connected to the black hole.

For abstractions from $\mathcal{A}_1$, the handlers of the arrow nodes of the first-order abstractions in the right-hand sides are connected to the handler of the arrow node of the abstraction.

For second-order abstractions $A' \Rightarrow A''$ from $\mathcal{A}_2$, the main arrow node connects ports distinct from the black hole from the main arrow of A' to ports of the main arrow of A'' according to the morphism between the two abstractions.

The abstractions are graphically defined as follows:



where $\mathcal{A}_0^+$ stands for one or more first-order abstractions.

We identify the *main arrow* node of an abstractions as the arrow node whose handler is not put in correspondence with another arrow node. Hence, in an abstraction, the handler of the main arrow cannot be the target or the source of an edge whose source or target respectively is an arrow node.

By construction, an abstraction is a connected port graph.

Example 16. In Figure 4.1 we illustrate an abstraction from $\mathcal{A}_1$ which specifies that a node with any name and with two ports both connected to a port of another node splits into two nodes, each one having a port, the connections being preserved. In addition, the abstraction introduces a new abstraction that splits in a similar way any node having the same name as the one already split by the abstraction. Here we draw the wiring using dashes so as to make a clear distinction from the edges in the left- and right-hand sides of an abstraction.

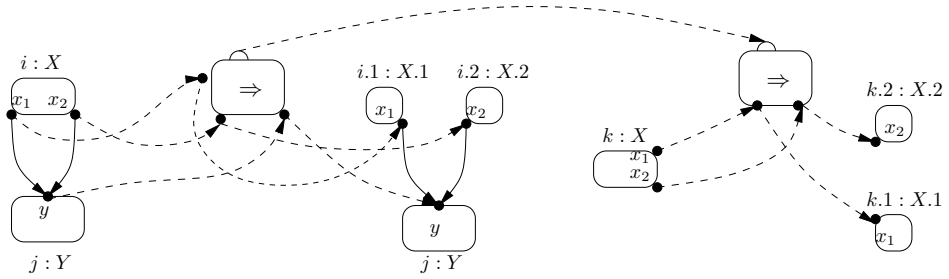


Figure 4.1: An abstraction with explicit wirings that specifies the splitting of a node identified by the couple of variables $i : X$ connected to a node $j : Y$ and introduces a rule for splitting any node having the same name as i

Port Graph Molecules

Let $\mathcal{X}$ be a set of variables. The set $\mathcal{G}$ of all elementary ρ_{pg}-objects of the calculus, called *port graph molecules*, is schematically defined by:

$$\mathcal{G} ::= \mathcal{X} \mid \mathcal{O} \mid \mathcal{A}_0 \mid \mathcal{A}_1 \mid \mathcal{A}_2 \mid \mathcal{G}\ \mathcal{G} \mid \varepsilon$$

Using the structure of port graphs for instantiating the parametric structure Σ in the $\rho_{\langle\Sigma\rangle}$-calculus, we obtain the definition of port graph molecules.

Based on the particular structure of port graphs, we define a structural congruence on port graph molecules.

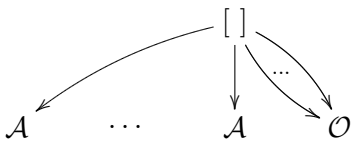
Definition 57 (Structural congruence on port graph molecules)**.** *The* structural congruence *on port graph molecules in a simple world is the smallest congruence closed with respect to a permutation of the ports of a node, to permutative juxtaposition* ___ *and to edge connection, hence satisfying the following axioms:*

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

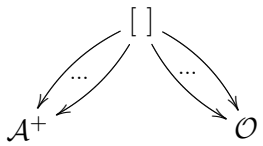
Worlds

We recall that a world is a container representing the port graph molecules in an *environment*. The environment is "aware" of all nodes it contains and connections between them. In order to express this awareness, we attach to each node in an object port graph an auxiliary *handler port*, usually denoted by p_0. Then we define the operator [ ] as a node with a handler port, and a *world* is a port graph where the handler of the node [ ] is connected to the handlers of the main arrows of all abstractions and to the handlers of all nodes of the object port graphs which are not part of abstractions. We recall that this node [ ] corresponds to a permutative variadic operator.

A world has the following graphical representation:



or, equivalently:



or just an object port graph if there are no abstractions in the state:

In writing, we use the notation $[G]$ for a world containing a port graph molecule $G \in \mathcal{G}$. For instance, the corresponding notations for the three port graphs drawn previously are $[\mathcal{A} \ldots \mathcal{A}\ \mathcal{O}]$, $[\mathcal{A}^+\ \mathcal{O}]$, and $[\mathcal{O}]$ respectively.

Let $\mathcal{Y}$ be a set of variables. Then the set $\mathcal{V}$ of worlds is defined as follows:

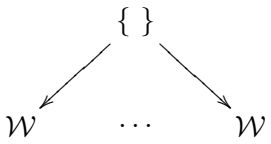
$$\mathcal{V} ::= \mathcal{Y} \mid [\mathcal{G}]$$

Multiverse

We introduce a node with a handler port for the operator { } which builds a structure of worlds (or multiverse). The handler of the node { } is connected to the handlers of all component worlds. A structure of worlds consists either of a variable in a set $\mathcal{Z}$, a world, or a multiset of structured of worlds:

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

Similarly to worlds, the notation $\{\mathcal{W}\ \mathcal{W}\}$ stands for the following port graph:



The underlying theory for multiset structure of the operator { } is follows the lines of the structural congruence defined in the abstract case in Section 2.2.6 based on the structural congruence for port graph molecules from Definition 57.

4.3 Semantics

4.3.1 Evaluation Rules as Port Graph Rewrite Rules

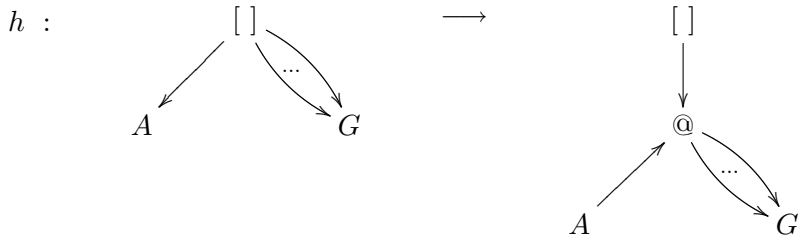
The ρ_{pg}-calculus expresses the evolution of a system whose initial state is a world of the form $\{[A_1 \ldots A_n\ O]\}$, with A_i an abstraction, for all i, $1 \leq i \leq n$, $n \geq 0$, and O an object port graph. The successful interaction between an abstraction and a port graph molecule gives rise to a port graph transformation. We instantiate the evaluation rules from the $\rho_{\langle\Sigma\rangle}$-calculus (Figure 2.7) with port graph molecules. The solving of the submatching

problem as well as the application of the matcher are defined at the meta-level of the calculus based on the port graph rewriting relation defined in Section 3.6.

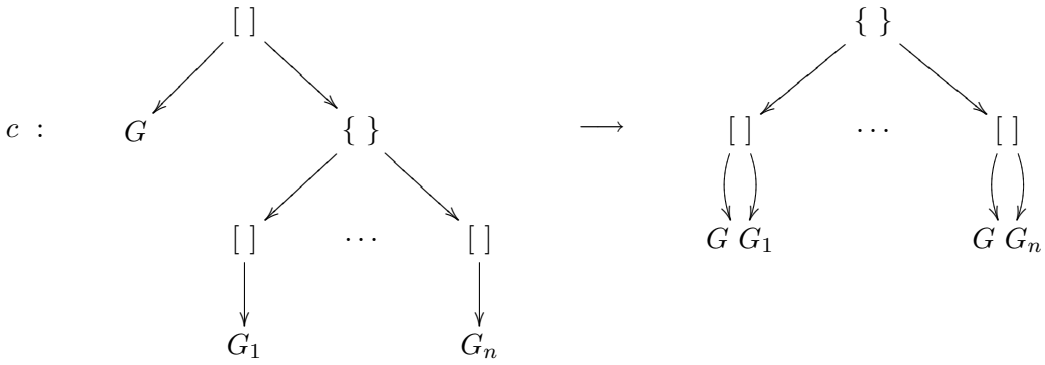
$$(L \Rightarrow R)\ G \longrightarrow \{[\varsigma_1(R)] \dots [\varsigma_n(R)]\} \quad \textbf{if} \quad \mathcal{S}ol(L \prec\!\!\!\cdot\ G) = \{\varsigma_1, \dots, \varsigma_n\} \tag{4.1}$$

$$A\ G \longrightarrow A\ G \quad \textbf{otherwise} \tag{4.2}$$

The heating and cooling rules have now a representation based on port graphs. The *heating* rule is the following port graph rewrite rule:



The *cooling* rule is represented by the following port graph rewrite rule:



The semantics of the $\rho_{\langle\Sigma\rangle}$-calculus including strategies can be easily instantiated using port graphs and all auxiliary operators for modeling the interaction correspond to nodes with ports. In particular, the failure construct stk is a node with a handler port.

4.3.2 The Application Mechanism as Port Graphs Rewrite Rules

All steps computing the application of an abstraction to an object port graph, including the (sub)matching and the replacement operations, are expressible using port graphs by considering some more auxiliary nodes (matching nodes, for instance) and extending the reduction relation with some graphical evaluation rules. This mechanism can be internalized in the calculus. Since this construction is quite technical, we present the evaluation rules in Appendix A. In this section we give only the main ideas of the evaluation rules.

The idea of using port graphs for encoding the calculus itself, the application and results, and now for encoding the matching algorithm as well, was inspired by the encoding of the ρ-calculus using Interaction Nets and Bigraphical Nets [FMS06].

Matching Nodes

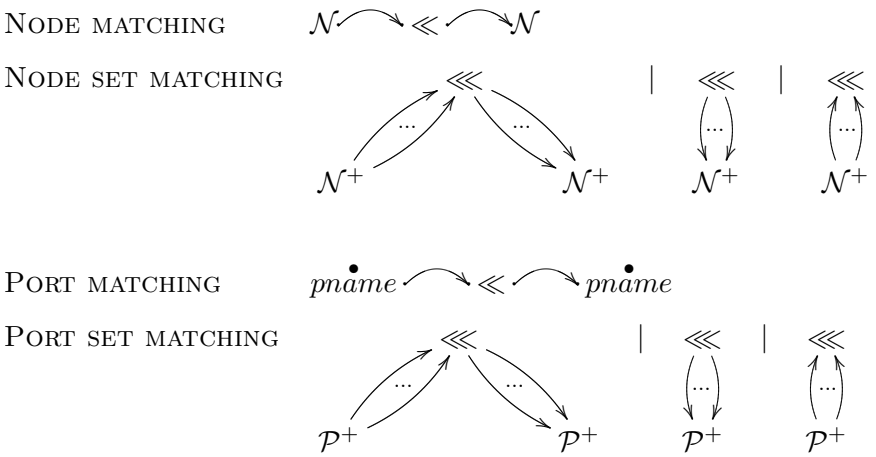
We consider two types of matching nodes:

1. $\ll$ for matching two ports; its handler port is the target of the edge connecting to the pattern port and, in the same time, it is the source of the edge connecting to the port subject to be matched;

2. $\lll$ for matching two sets of ports; all ports from the pattern are connected to the handler of $\lll$, which is in turn connected to all ports from the subject.

When a matching node connect handlers ports in both pattern and subject, then we say that we match nodes.

We consider two different matching nodes since we want to make a distinction when the matching node $\lll$ connects two singletons. A similar condition was used also in the matching algorithm given for port graph in Chapter 3 in the decomposition rule $\mathbf{D_1}$. We add the following port subgraph constructs to the set of elementary objects of the calculus:



Internal Rules for Matching and Replacement

In Appendix A we give the rules handling the matching and the replacement using port graph transformations. Besides the graphical representation, we also provide a term-like notation based on the algebraic signature from Chapter 5 where we emphasize the matching problem as a term matching and not as a matching node. We consider a substitution node connected to the main arrow node of the abstraction applied corresponding to the substitution built during the matching process. We denote it generically by σ. We do not represent the edges, the nodes, and the ports not relevant to the application of a reduction rule, and they are unchanged. For each internal evaluation rule we described its application result in terms of the rules from the matching algorithm for port graphs given in Chapter 3, or in terms of the definition of arrow-translation. Also we impose an application order in terms of a partial relation on the rules application priority.

4.4 Conclusions

In this chapter we have seen how the Abstract Biochemical Calculus is instantiated for the structure of port graphs. This structure brings more flexibility to the calculus by permitting the encoding of meta-level operations like matching and replacement used in the application as evaluation rules. This proves once more the universality of the port graph structure, in addition to the possibility of encoding a port graph rewrite rule as a port graph as we have shown in Chapter 3.

In [FMS06] the authors propose bigraphical nets for encoding the ρ-calculus in the context of defining efficient reduction strategies; in particular they use bigraphical nets to model the matching in the ρ-calculus. A bigraphical net is a combination between interaction nets and bigraphs, such that the controls associated to nodes have a distinguished principal port and the interactions occur between principal ports. Then an interesting future work would concern the study of a biochemical calculus on bigraphs and the suitability of bigraphical nets or general bigraph to express matching and replacement in the calculus as we already done for the ρ_{pg}-calculus.

5 Term Rewriting Semantics for Port Graph Rewriting

5.1 Introduction

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 3 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 heavily 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 3 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 5.2 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 5.3 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 5.4) and by defining some auxiliary operations and reduction relations necessary for in the replacement process (Section 5.5). In Section 5.6 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 5.7. In Section 5.8 we embed the term approach on port graph rewriting in the rewriting calculus obtaining a term rewriting calculus for port graphs.

5.2 Term Encoding of Port Graphs

5.2.1 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:

$$\begin{aligned}\mathcal{S} \;=\; \{&\mathit{Id},\ \mathit{Name},\ \mathit{Port},\ \mathit{Node},\ \mathit{Edge},\ \mathit{Neighbor},\ \mathit{AdjacencyEq},\ \mathit{PortSet},\ \mathit{NodeSet},\\ &\mathit{EdgeSet},\ \mathit{NeighborSet},\ \mathit{AdjacencyEqSet},\ \mathit{PGraph}\}\end{aligned}$$

The *subsort relation* is defined by $X < \mathit{XSet}$ for $X \in \{\mathit{Port}, \mathit{Node}, \mathit{Edge}, \mathit{Neighbor}, \mathit{AdjacencyEq}, \mathit{PGraph}\}$, 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 5.1 where X takes sort values from the set $\{\mathit{Node}, \mathit{Edge}, \mathit{Neighbor}, \mathit{AdjacencyEq}\}$. 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.

5.2.2 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.

$\bullet$: $\longrightarrow Id$	$\bullet$: $\longrightarrow Port$	$\bullet$: $\longrightarrow Node$ $\quad \epsilon_X$: $\longrightarrow XSet$
$\_,\_$	:	$XSet\ XSet \longrightarrow XSet\ [ACU(\epsilon_X)]$
$\langle \_ : \_ \parallel \_ \rangle$	:	$Id\ Name\ PortSet \longrightarrow Node$
$(\_,\_)$	:	$Port\ Port \longrightarrow Edge$
$\_^\frown\_$	:	$Id\ EdgeSet \longrightarrow Neighbor$
$\_ \doteq \_$	:	$Id\ NeighborSet \longrightarrow AdjacencyEq$
$\_(\!\lfloor\_\rfloor\!)$	:	$NodeSet\ AdjacencyEqSet \longrightarrow PGraph$

Figure 5.1: The operation set $\mathcal{F}$

Definition 58 (Encoding port graphs as terms)**.** *We encode a port graph $G = (V, E)$ as an algebraic term $\mathcal{E}(G) = T_1(\!\lfloor T_2 \rfloor\!)$ of sort PGraph where:*

- *$T_1 \in \mathcal{T}_{\Sigma,NodeSet}(\mathcal{X})$ represents the set of all node labels in G, and*
- *$T_2 \in \mathcal{T}_{\Sigma,AdjacencyEqSet}(\mathcal{X})$ 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.*

Example 17. The port graph G illustrated in Figure 3.1 is encoded as the following term:

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

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.

Definition 59 (Well-formed terms)**.** *A term $t \in \mathcal{T}_{\Sigma,PGraph}(\mathcal{X})$ 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:

Definition 60 (Canonical form)**.** *A term $t \in \mathcal{T}_{\Sigma,PGraph}(\mathcal{X})$ is in* canonical form *if:*

- *the right-hand sides of adjacency equations are non-empty sets of neighbors;*
- *only non-empty set of edges occur in neighbor terms.*

Example 18. The term in canonical form corresponding to G is the following:

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

5.3 pg-Rewrite Rules

For all rewrite rules over $\mathcal{T}_{\Sigma,PGraph}(\mathcal{X})$, according to Definition 48, we impose node identifiers occurring in the left-hand side to be variables. We say that a rewrite rule over $\mathcal{T}_{\Sigma,PGraph}(\mathcal{X})$ is well-formed if both t_1 and t_2 are well-formed. We call *pg-rewrite rule* a well-formed rewrite rule over $\mathcal{T}_{\Sigma,PGraph}(\mathcal{X})$.

Definition 61 (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 an 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 an pg-rewrite rule can be extracted automatically by means of an analysis on each identifier occurrences. We call this procedure GetMap. It produces a set of elementary mappings from $\mathcal{T}_{\Sigma,Node}(\mathcal{X})$ to $\mathcal{T}_{\Sigma,NodeSet}(\mathcal{X})$ 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.

Example 19. The encoding of the port graph rewrite rule (a) given in Figure 3.4 is:

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

with the node-morphism

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

5.4 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 $\mathcal{T}_{PGraph}(\mathcal{X})$ 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]. We usually denote by W an extension variable and by $\overline{t}$ the extension of term t. For each rewrite rule $t_1 \twoheadrightarrow t_2$, extension variables are appended to set-sorted terms to produce the extended rule $(\overline{t_1 \twoheadrightarrow t_2})$. An extension variable is added to each set-sorted subterm in the left-hand side (and accordingly in the right-hand side). This technical construction is formalized in the definition below. For this operation we consider also the trivial node-morphisms, $t \mapsto t$ with $t \in \mathcal{T}_{\Sigma,Node}(\mathcal{X})$.

Definition 62 (Extending an pg-rewrite rule)**.** *The* extension *of a pg-rewrite rule $t_1 \twoheadrightarrow t_2$, where ξ is its node-morphism, consists in adding extension variables for the set operators according to the following steps:*

1. **Extend the node-morphism in the case of weak port graphs**
   If $\xi = \{\xi_1, \ldots, \xi_m\}$ then its extension is $\overline{\xi} = \{\overline{\xi_1}, \ldots, \overline{\xi_m}\}$. For each elementary node-morphism $\xi_k = \{\langle i : t \parallel P\rangle \rightsquigarrow \langle i_1 : t_1 \parallel P_1\rangle, \ldots, \langle i_n : t_n \parallel P_n\rangle$ with $P, P_1, \ldots, P_n \in \mathcal{T}_{\Sigma,PortSet}(\mathcal{X})$, its extension is:

$$\overline{\xi_k} = \langle i : t \parallel P, W_1, \ldots, W_n\rangle \rightsquigarrow \langle i_1 : t_1 \parallel P_1, W_1\rangle, \ldots, \langle i_n : t_n \parallel P_n, W_n\rangle$$

   where $\{W_k\}_{k=1..n}$ are pairwise distinct and fresh extension variables of sort PortSet. If $\xi_k = \langle i : t \parallel P\rangle \rightsquigarrow \bullet$ then $\overline{\xi_k} = \langle i : t \parallel P, W\rangle \rightsquigarrow \bullet$, with W a fresh extension variable of sort PortSet.

2. **Extend the left-hand side of the rule**
   If $t_1 = N_1(\!|A_1|\!)$, its extension is $\overline{t_1} = \overline{N_1}(\!|\overline{A_1}|\!)$ where:
   - *$\overline{N_1}$ is obtained from N_1 by replacing each Node-term by its extension as computed for the node-morphism and appending a fresh extension variable of sort NodeSet;*
   - *$\overline{A_1}$ is obtained from A_1 by adding a new extension variable for each term of sort EdgeSet, NeighborSet, AdjacencyEqSet.*

3. **Extend the right-hand side of the rule**
   If $t_2 = N_2(\!|A_2|\!)$ and $\overline{t_1} = (\overline{u_1}, \ldots, \overline{u_m}, W_1)(\!|\overline{a_1}, \ldots, \overline{a_l}, W_2|\!)$ then $\overline{t_2} = \overline{N_2}(\!|\overline{A_2}|\!)$ where:
   - *$\overline{N_2} = (\overline{\xi}(\overline{u_1}), \ldots, \overline{\xi}(\overline{u_1})) \downarrow_{\mathcal{R}}, W_1, \{\langle i : v \parallel P\rangle \in N_2 \mid i \notin dom(\xi)\}$, where $\mathcal{R}$ are the rules for transforming a term in canonical form;*
   - *$\overline{A_2} = A_2, (\overline{a_1} \setminus a_1), \ldots, (\overline{a_l} \setminus a_l)$ where $\setminus$ computes the difference between adjacency equations which consists in removing the edges appearing in the right-hand side from the left-hand side (for example $(X \doteq (Y^\frown x, L_1), L_2) \setminus (X \doteq Y^\frown x)) = (X \doteq (Y^\frown L_1), L_2)$).*

Example 20. The extension of the rule given in Example 19 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*.

5.5 Auxiliary Operations and Reduction Relations

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

5.5.1 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:

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

5.5.2 Node-Morphism Application

The application of a node-morphism ξ to a term encoding a port graph as we have introduced in in Definition 44 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 5.2. 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 21.

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

Proof. The reduction of $\xi(N(\!|A|\!))$ using $\mathcal{A}$ terminates since ξ is a finite set of mappings and each mapping traverses the term $N(\!|A|\!)$ using the rules (Propagate), (Distribute), and (ApplySrc), to be eliminated in the end by the rule (ApplyTar).

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

(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)	$\{t \mapsto t_1, \ldots, t_n\}(i \doteq V) \to$
	$\texttt{if}\ id(t) \neq i\ \texttt{then}\ i \doteq (\{t \mapsto t_1, \ldots, t_n\}V)$
	$\texttt{else}\ id(t_1) \doteq (\{t \mapsto t_1, \ldots, t_n\}V), \ldots, id(t_n) \doteq (\{t \mapsto t_1, \ldots, t_n\}V)$
(ApplyTar)	$\{t \mapsto t_1, \ldots, t_n\}(i^\frown E) \to$
	$\texttt{if}\ id(t) \neq i\ \texttt{then}\ i^\frown E\ \texttt{else}\ id(t_1)^\frown E, \ldots, id(t_n)^\frown E$

Figure 5.2: Rules for node-morphism application ($\mathcal{A}$)

Since all mappings in ξ are pairwise distinct on the mapped node, there are no critical pairs among the rules in $\mathcal{A}$. Hence $\mathcal{A}$ is locally confluent, and, since it is terminating, $\mathcal{A}$ is also confluent. □

After applying a node-morphism on a *PGraph*-term, the resulting term may be neither well-formed, nor 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.

5.5.3 Rules for Ensuring Well-Formedness

Let $\mathcal{W}$ be the rewrite system defined by the rules presented in Figure 5.3. These rules transform terms with respect the condition of well-formedness of *PGraph*-sorted terms specified in Definition 59 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 21 we illustrate a reduction using the rules in $\mathcal{W}$.

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

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

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

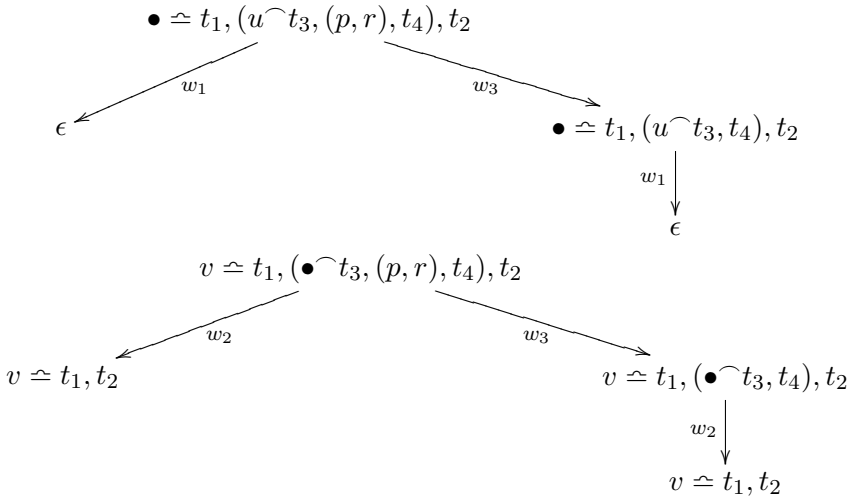
(w_3) $(v \doteq t_1, (u \frown t_3, (p, r), t_4), t_2) \rightarrow (v \doteq t_1, (u \frown t_3, t_4), t_2)$
$\texttt{if}\ p \notin ports(v)\ \texttt{or}\ r \notin ports(u)$

Figure 5.3: Rules for reducing a term to a well-formed term ($\mathcal{W}$)

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

Proof. $\mathcal{W}$ terminates since every rule is just removing a term element of a set-like term and the terms are finite.

$\mathcal{W}$ is locally confluent since the two critical pairs produced by (w_1) with (w_3) and (w_2) with (w_3) are joinable:



$\mathcal{W}$ is terminating and locally confluent, hence $\mathcal{W}$ is 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 $t{\downarrow_{\mathcal{W}}}$.

5.5.4 Computing the Canonical Form

Let $\mathcal{C}$ be the rewrite system defined by the rules in Figure 5.4. These rules transform terms in the canonical form specified by Definition 60 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.

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

$$(c_1)\ \langle v : n \parallel t_1 \rangle, \langle v : n \parallel t_2 \rangle \ \rightarrow\ \langle v : n \parallel t_1, t_2 \rangle$$
$$(c_2)\ v^\frown \epsilon_{Edge} \ \rightarrow\ \epsilon_{Neighbor}$$
$$(c_3)\ (v^\frown t_5), (v^\frown t_6) \ \rightarrow\ v^\frown t_5, t_6$$
$$(c_4)\ v \doteq \epsilon_{Neighbor} \ \rightarrow\ \epsilon_{AdjacencyEq}$$
$$(c_5)\ (v \doteq t_3), (v \doteq t_4) \ \rightarrow\ v \doteq t_3, t_4$$

Figure 5.4: 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.

Proof. $\mathcal{C}$ is terminating since each rule decrements the number of elements in a set-like term. The both critical pairs between (c_2) and (c_3), and between (c_4) and (c_5) are obviously joinable, hence $\mathcal{C}$ is locally confluent. □

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

5.6 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.

Definition 63 (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 $\xi = \mathsf{GetMap}(\overline{r})$, which is denoted by $t \xrightarrow{\overline{r}}_{\twoheadrightarrow} t'$, 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.*

Example 21. We present here a result of pg-rewriting the term $t = \mathcal{E}(G)$ from Example 17 encoding the port graph from Figure 3.1, using the rewrite rule $t_1 \twoheadrightarrow t_2$ given in

Example 19 and extended in Example 20 which encodes the port graph rewrite rule (a) from Figure 3.4. This result of the pg-rewriting corresponds to the port graph rewriting depicted in Figure 3.8.

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:

$$\xi^{\sigma}(\sigma(\overline{t_2}))\downarrow_{\mathcal{W}}\downarrow_{\mathcal{C}} \quad = \quad \xi^{\sigma}(\sigma(\overline{t_2}))\downarrow_{\mathcal{W}}$$

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 3.8.

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.

Proof. The uniqueness of the occurrences of a node identifier in the node set is ensured by the normalization with respect to (c_1), in the adjacency equation set as left-hand side of an adjacency equation by the normalization with respect to (c_5), and in the neighbor set by the normalization with respect to (c_3).

Since t is well-formed, the application of a pg-rewrite rule does not introduce new node identifiers in the adjacency equation list without introducing them as well in the node set. The occurrences of ports that no longer exist (since the node they were placed in was deleted) are removed from the adjacency equation set using the rules (w_1) and (w_2). In addition, the normalization with respect to the rule (w_3) ensures the exclusive presence of edges with valid ports.

The rules (c_2) and (c_4) eliminate adjacency equations with empty sets of neighbors and neighbors with empty sets of edges.

In conclusion, since t' is irreducible with respect to $\mathcal{W}$ and $\mathcal{C}$ by definition, it results that t' is well-formed and in canonical form. □

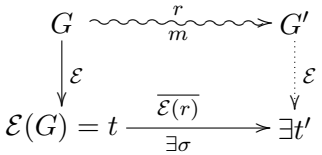
Remark 7. Since a port graph rewrite rule has a port graph representation, then the encoding presented here for the port graph rewriting relation handles as well all types of abstraction considered in the ρ_{pg}-calculus (Chapter 4).

5.7 Operational Correspondence

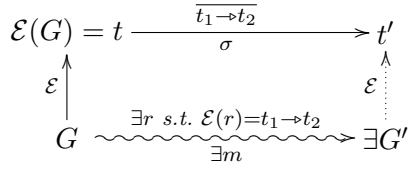
In this section we show that the encoding of the weak port graph rewrite relation from Chapter 3 as a term rewriting relation is sound and complete.

Theorem 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 $t' = \mathcal{E}(G')$.*



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 $\mathcal{E}(G') = t'$.*



Proof. 1) We start by encoding the weak port graph rewrite rule $r = G_1 \rightsquigarrow G_2$ as $\mathcal{E}(r) = t_1 \twoheadrightarrow t_2$ and extending it.

Let us compute a substitution σ' based on the matching morphism m. For each mapping of the type $m(i) = i'$, with $\mathit{Interface}(i) = (n, P)$ and $\mathit{Interface}(i') = (n', P')$ we define $\sigma'(i) = i'$, and if n is variable then $\sigma'(n) = n'$. While for each mapping $m(p) = p'$ with $p, p' : \mathit{Port}$, if p is variable then $\sigma'(p) = p'$.

At this point $\sigma'(\overline{t_1})$ contains as variables only extension variables which capture sets of nodes, ports, adjacency equations, neighbors, and edges. By solving the matching problem $\sigma'(\overline{t_1}) \ll t$ we recover a substitution σ'' for the extension variables. During the solving process of the matching problem $\sigma'(\overline{t_1}) \ll t$, we replace a matching equation $u \ll v$ with $u, v \in \mathcal{T}_{\Sigma, \mathit{Node}}(\mathcal{X})$ by $\xi(u) \ll \xi(v)$ and solve it. This ensures us to choose the same partition for the port sets as for the unmatched partial nodes in the port graph rewriting process.

Let us define σ as the composition of σ' and σ''. Then the mapping ξ^σ is built to mimic the connection process between nodes of G^- and $m(G_2)$ by handling the unmatched partial nodes, the unmatched edges and the bridges. The unmatched partial nodes are replaced by their correspondents, the endpoints of unmatched edges are updated, and the dangling bridges are redirected by applying ξ^σ to $\sigma(\overline{t_2})$. Consequently, $\xi^\sigma(\sigma(\overline{t_2}))$ is a representation of $G^-[m(G_2)]_{m(\xi)(\mathcal{U}_n), m(\xi)(\mathcal{U}_e), m(\xi)(\mathcal{B})}$. Then normalization with respect to the rules in $\mathcal{W}$ corresponds to removing deleted nodes and ports and their incident edges. In the end, we make sure the term is well-formed and in canonical form by normalizing with respect to the rules in $\mathcal{C}$. We thus get a term t' which is indeed $\mathcal{E}(G')$.

2) Let us first define recursively the inverse operation of the encoding $\mathcal{E}$, which we denote by $\mathcal{E}^{-1}$:

$$\mathcal{E}^{-1}(N(\!|A|\!)) = (\mathcal{E}^{-1}(N), \mathcal{E}^{-1}(A)), \text{ if } N(\!|A|\!) \text{ is well-formed and in canonical form}$$
$$\mathcal{E}^{-1}(T_1, \ldots, T_k) = \{\mathcal{E}^{-1}(T_1), \ldots, \mathcal{E}^{-1}(T_k)\}, \text{ for } T_1, \ldots, T_k \text{ a set-like term}$$
$$\mathcal{E}^{-1}(\langle i : n \parallel P \rangle = i \text{ with } \mathit{Interface}(i) = (n, \mathcal{E}^{-1}(P))$$

$$\mathcal{E}^{-1}(p) = p$$
$$\mathcal{E}^{-1}(i \doteq N_1, \ldots, N_k) = \mathcal{E}^{-1}(i \doteq N_1) \cup \mathcal{E}^{-1}(i \doteq N_k)$$
$$\mathcal{E}^{-1}(i \doteq j^\frown(p_1, r_1), \ldots, (p_k, r_k)) = \{(i^\frown p_1, j^\frown r_1), \ldots, (i^\frown p_k, j^\frown r_k)\}$$
$$\mathcal{E}^{-1}(t_1 \twoheadrightarrow t_2) = \mathcal{E}^{-1}(t_1) \rightsquigarrow \mathcal{E}^{-1}(t_2)$$

Let $G_1 \rightsquigarrow G_2 = \mathcal{E}^{-1}(t_1 \twoheadrightarrow t_2)$, and let σ' be the restriction of σ on non-extension variables (hence it maps variable node identifiers, node names and ports). If we consider the set of nodes encoded by t_1, the substitution σ' and the identity mapping for the rest of the node identifiers, node names, and ports not in the domain of σ', we are able to construct the corresponding matching morphism for G_1 and G.

Let t_1' be a term obtained from $\overline{t_1}$ by keeping only the structure along with the node identifiers, node names, and extension variables; then, once they are instantiated by σ:

- the subterms $\langle i : n \parallel W \rangle$ encode unmatched partial nodes,
- the extension variable of sort *NodeSet* encodes the context nodes,
- the subterms $i \doteq (j^\frown W_1), W_2$ encode unmatched edges and bridges if $i, j \neq \bullet$,
- the extension variable of sort *NeighborSet* encodes bridges and context edges.

Since $\mathcal{E}^{-1}(\sigma(\overline{t_1})) = G$ and $\mathcal{E}^{-1}(\sigma(t_1)) = m(G_1)$, we saw above that G is a composition of $m(G_1)$, unmatched partial nodes and edges, bridges, and a context graph. Hence we can write $G = G^-[m(G_1)]_{\mathcal{U}_n, \mathcal{U}_e, \mathcal{B}}$.

The ACU-matching algorithm returns all solutions for the extension variables corresponding to all partitions of the port sets in the unmatched partial nodes. Since among all solutions, the substitution σ is chosen, then ξ^σ provides the right partition of the port sets needed for updating the unmatched partial nodes via the node-morphism in the port graph rewriting. Hence we obtain a resulting graph $G' = G^-[m(R)]_{m(\xi)(\mathcal{U}_n), m(\xi)(\mathcal{U}_e), m(\xi)(\mathcal{B})}$ which is encoded by 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.

5.8 Relation to the ρ-Calculus

Based on the encoding of the port graph rewriting relation as defined in Section 5.6, we instantiate the ρ-calculus 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 $\mathcal{T}_{\Sigma,PGraph}(\mathcal{X})$;
- consider only port graph rewrite rules corresponding to the possible types of abstraction in the ρ_{pg}-calculus (Chapter 4).

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-calculus, 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:

$$(\rho_{tpg}) \quad (\overline{t_1 \twoheadrightarrow t_2})\ t_3 \rightarrow_\rho S(\varsigma_1(\overline{t_2})) \wr \ldots \wr S(\varsigma_n(\overline{t_2})),$$
$$\texttt{if } \mathcal{S}ol(\overline{t_1} \ll t_3) = \{\sigma_1, \ldots, \sigma_n\}, \xi = \mathsf{GetMap}(\overline{t_1 \twoheadrightarrow t_2}), \varsigma_i = \sigma_i \circ \xi^{\sigma_i}$$

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 ρ-calculus, and we call it the ρ_{tpg}-calculus.

5.8.1 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.

5.8.2 The Relation between the ρ_{pg}-Calculus and the ρ_{tpg}-Calculus

In Figure 5.5 we review the relations between the calculi we developed until now:

- the $\rho_{\langle\Sigma\rangle}$-calculus generalizing the γ-calculus based on the pattern matching idea from the ρ-calculus on an arbitrary structure described by Σ;
- the ρ_{pg}-calculus instantiating the $\rho_{\langle\Sigma\rangle}$-calculus 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}-calculus is based on the ρ_{tpg}-calculus in a similar way the $\rho_{\langle\Sigma\rangle}$-calculus is based on the ρ-calculus.

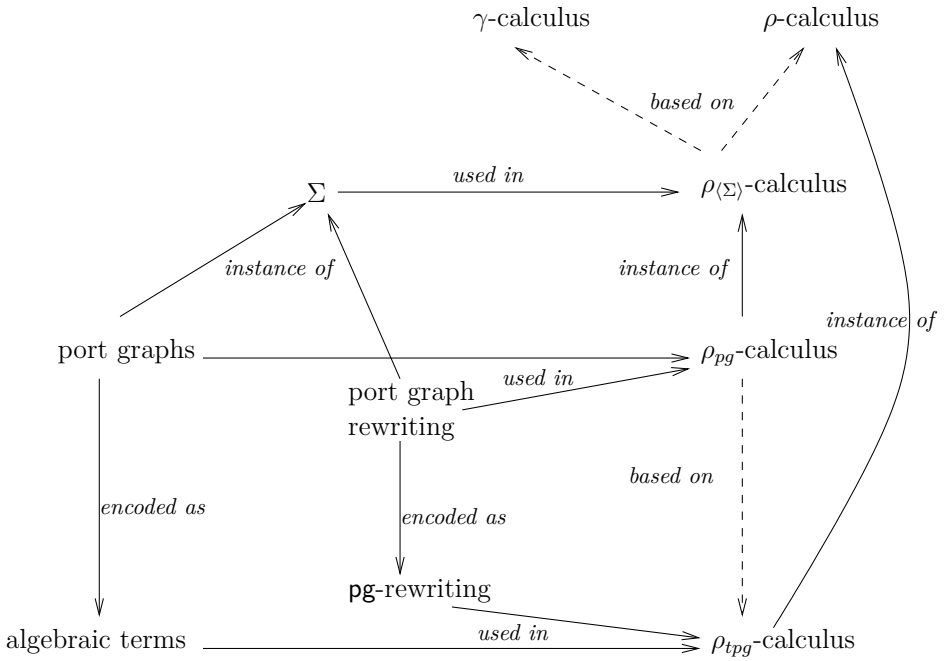


Figure 5.5: Relations between the $\rho_{\langle\Sigma\rangle}$-calculus, the ρ_{pg}-calculus, and the ρ_{tpg}-calculus

5.9 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-calculus with the algebraic signature for port graphs, the class of patterns and abstractions corresponding to the abstractions used in the ρ_{pg}-calculus, 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 6, 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 C. 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 calculus. 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.

6 Case Studies for the ρ_{pg}-calculus

In this chapter we analyze two applications for the ρ_{pg}-calculus. 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}-calculus 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.

6.1 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]. Beyond the chemical programming idea, another approach presented in [CBL04], called the Organic Grid, is similarly a radical departure from current approaches and is inspired by the self-organization property of complex biological systems.

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 non-deterministically. 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}-calculus, we are able to express rules and strategies as port graphs and so to rewrite them as well. The ρ_{pg}-calculus 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 [AK08c, AK08b].

6.1.1 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}-calculus. 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].

Example 22 (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 6.1 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.

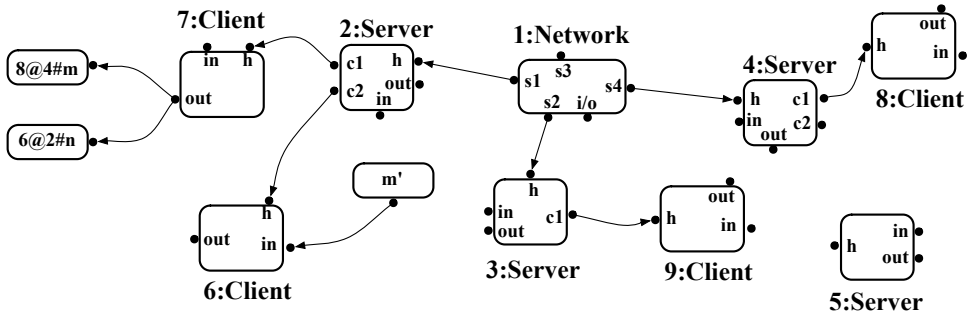


Figure 6.1: A mail system configuration

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.

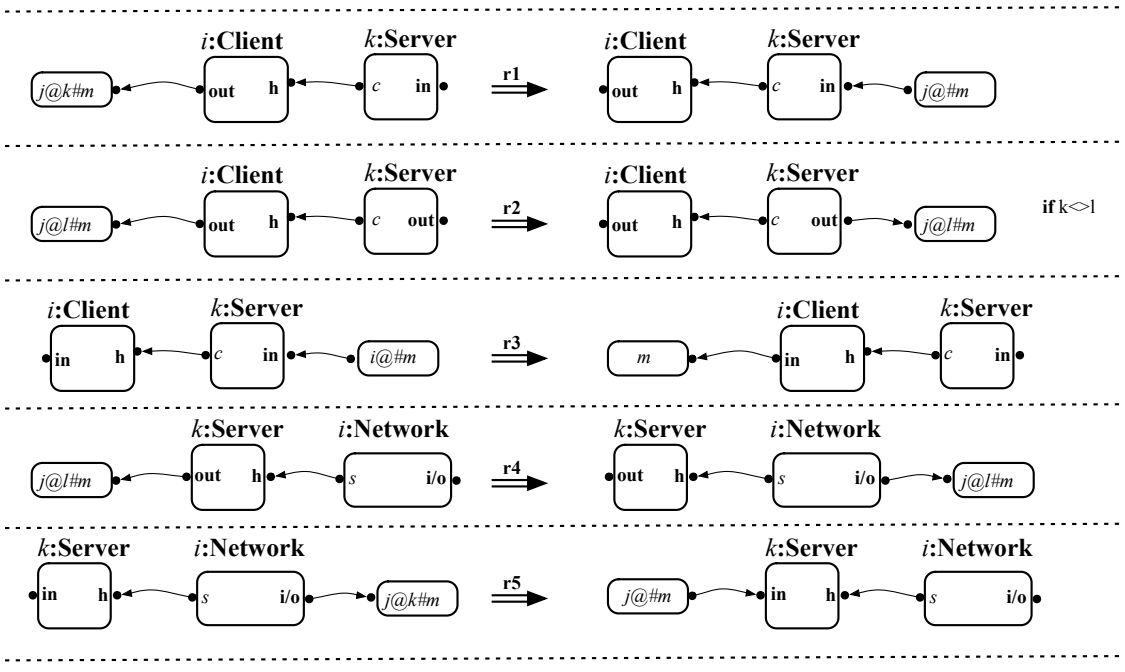


Figure 6.2: Basic rules for the mail delivery system

We illustrate in Figure 6.2 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 **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 6.2 using the reduction semantics of the ρ_{pg}-calculus.

Rule **r6** in Figure 6.3 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 $\texttt{first}(\texttt{repeat}(\mathbf{r3}), \mathbf{r6})$.

Another interesting problem may concern the operations of replacing a server by two or more server or, in the other way around, replace a group of servers by one server. Then biologically inspired port graph rules from Figure 3.4 (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

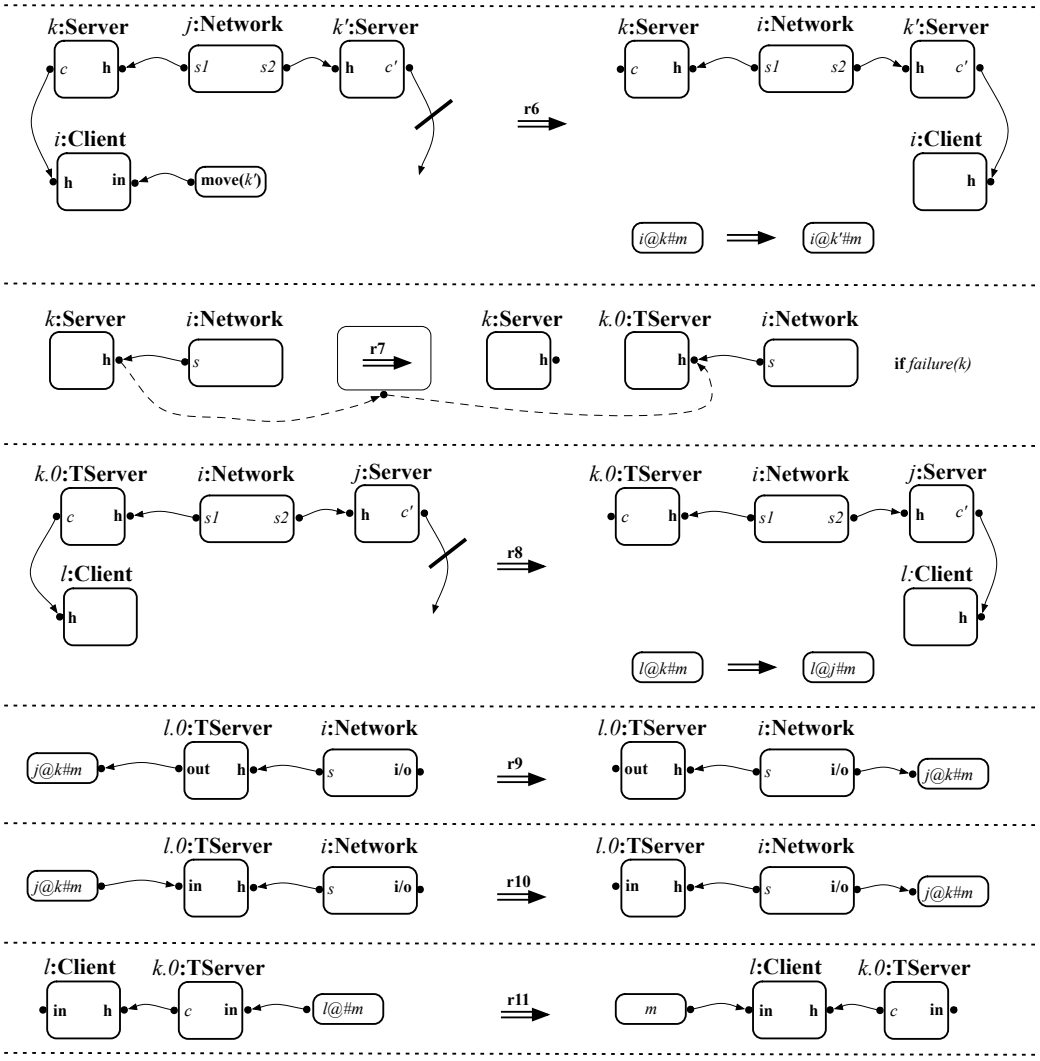


Figure 6.3: Rules for self-configuration and self-healing in the mail delivery system

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 $\mathbf{r7}; \texttt{repeat}(\mathbf{r11}); \texttt{repeat}(\texttt{first}(\mathbf{r8}, \mathbf{r9}, \mathbf{r10}))$. 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.

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 6.4.

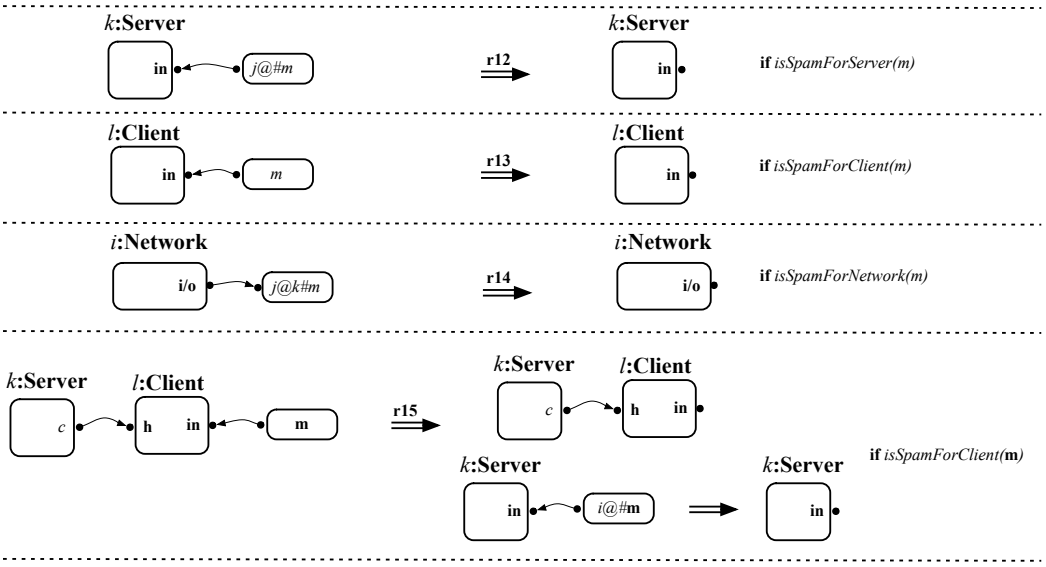


Figure 6.4: Rules for self-protection in the mail delivery system

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.

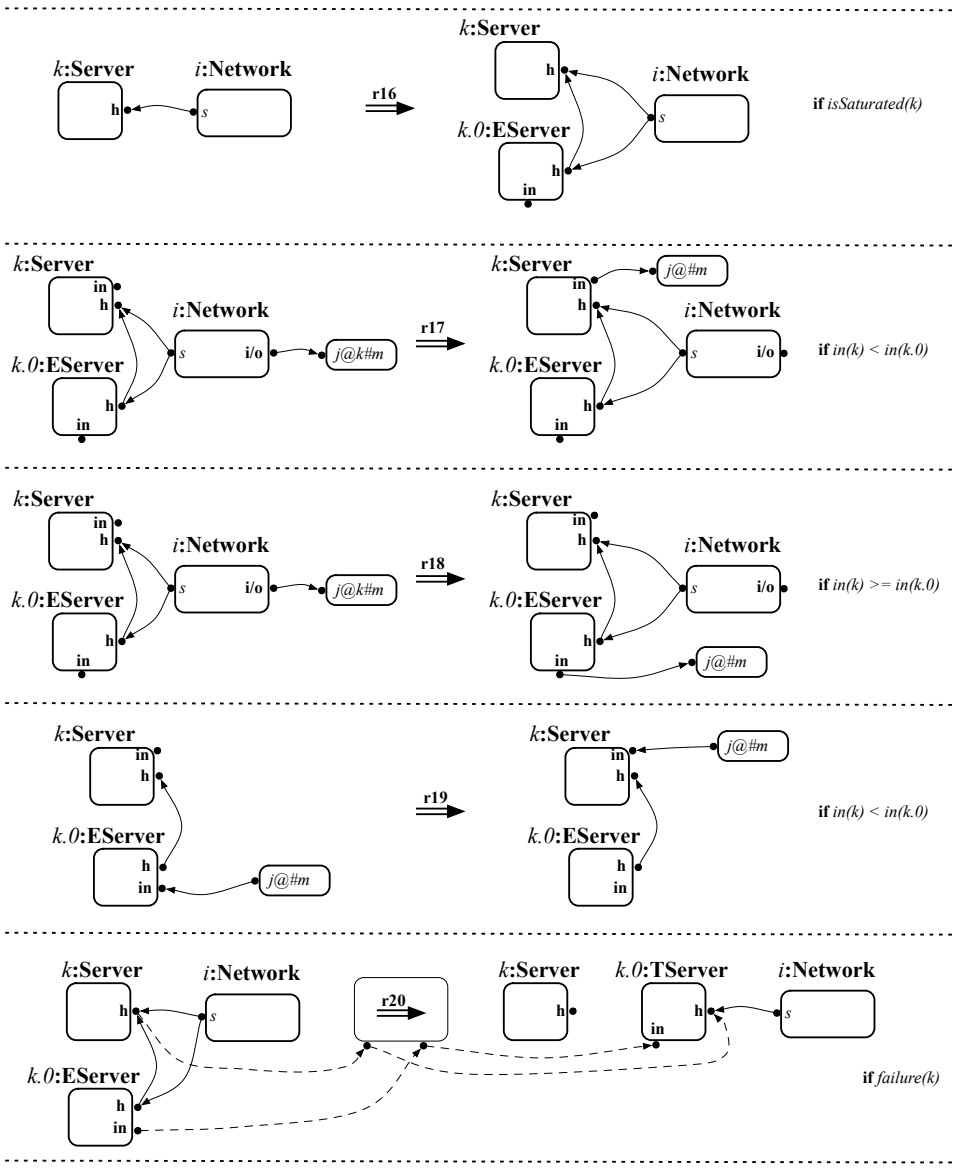


Figure 6.5: Rules for self-optimization in the mail delivery system

6.1.2 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}-calculus. The model should also ensure formally that the intended

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}-calculus. 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 $\texttt{first}(S, recovery(S))$. If the recovery strategy is id then we obtain $\texttt{try}(S)$ 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:

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

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:

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

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}-calculus. In particular, we show in Chapter 7 how we can increase the expressivity of the ρ_{pg}-calculus 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.

6.2 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 [AK07b], 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.

6.2.1 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

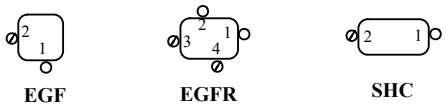
We represent molecular complexes as a particular class of port graphs. For a given biological model, we can extract a p-signature by associating to each protein name its site names.

Definition 64 (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.*

Example 23. 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 6.6 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 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.

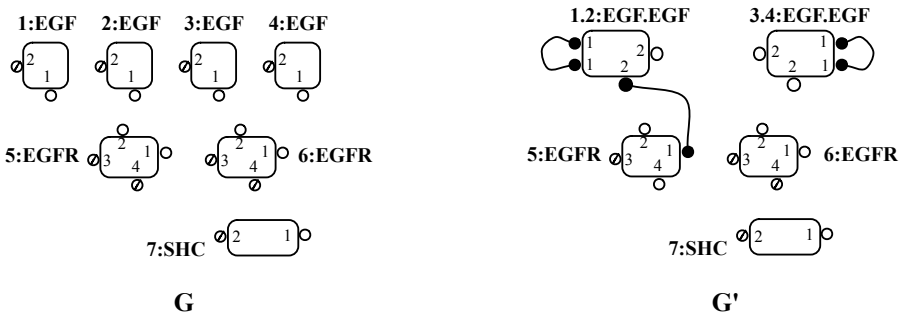


Figure 6.6: Initial and intermediate molecular graphs in the EGFR model

Rewriting Molecular Graphs

Definition 65 (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.

Example 24. In Figure 6.7 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.

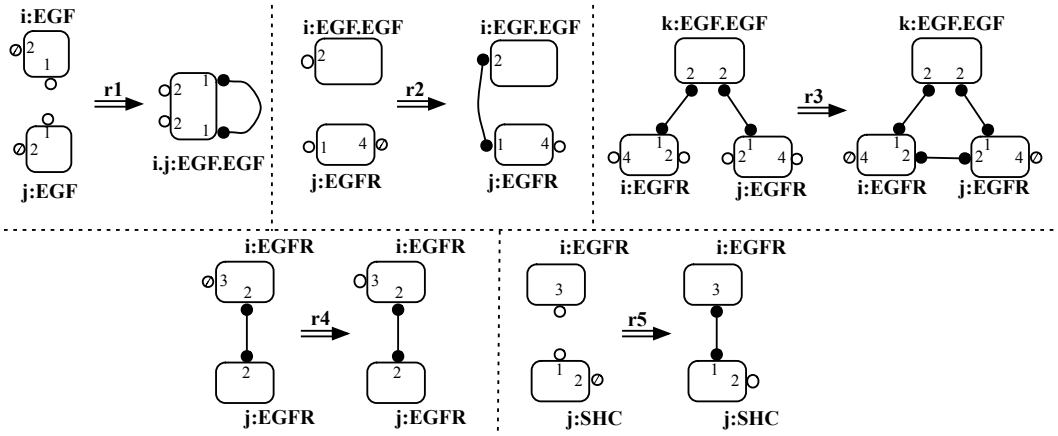


Figure 6.7: The reaction patterns in the EGFR signaling pathway fragment

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.

Example 25. We illustrate in Figure 6.8 two possible results of applying the rewrite rule **r2** on the molecular graph **G'**.

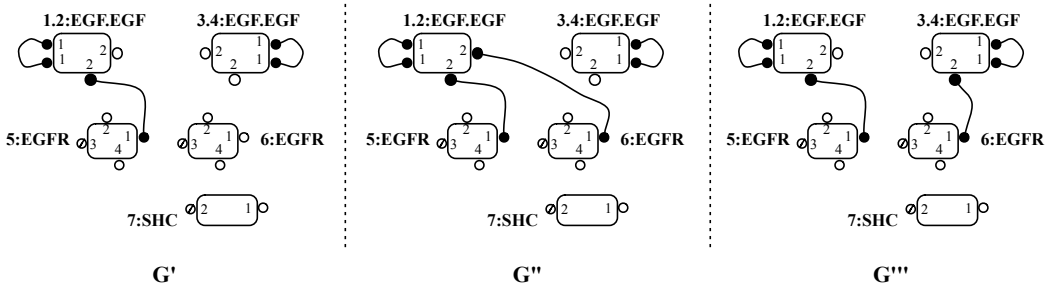


Figure 6.8: **G"** and **G"'** are obtained from **G'** by rewriting using the rule **r2** on different subgraphs

6.2.2 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 corresponding 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.

Example 26. 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:EGF 2:EGF 3:EGF 4:EGF 5:EGFR 6:EGFR 7: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, $\texttt{first}(\mathbf{r_2}, \mathbf{r_1})$ 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:

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

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 6.9 or **G"'** from Figure 6.8.

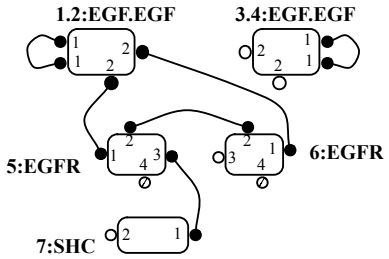


Figure 6.9: The molecular graph **H** for the EGFR signaling pathway fragment in the equilibrium state

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 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.

6.2.3 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 [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 5 is close to the Pathway Logic approach to use 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 based on a very similar idea used for the encoding in Chapter 5. In addition, we use the pointers introduced in TOM for rewriting term-graphs; 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 C after reviewing the main features of TOM in Appendix B.

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}-calculus 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.

6.3 Conclusions and Perspectives

In the first part of this chapter we saw that the biochemical calculus instantiated for the structure of port graphs has, 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}-calculus can provide by considering a port graph structure for the resources and by controlling the order of interactions.

In the second part of the chapter, we showed that the ρ_{pg}-calculus 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].

Cellular membranes can also form complexes, called *tissues*, due to the binding proteins on their surfaces. This kind of complex can be studied in (mem)brane-based formalisms like for instance the Brane Calculus [Car05b] or the Membrane Comput-

ing [Pau02]. The latter one already has models based on the tissue-like structure, the Population P Systems [BG04] for example. A population P system with active cells has the following characteristics: each cell contains a multiset of objects and two cells connect based on particular internal objects, bonds can be added or deleted by rules acting on the population, the objects in a cell are subject to transformation by the rules internal of that particular cell or by communication rules between cells, and the cells are subject to differentiation, division, or death rules. We remark that the port graph structure and the port rewrite rules are well-suited for modeling population P system with active cells. Moreover, the behavior of a P system is governed by various control mechanisms[Pau02, IYD05], for instance maximal or bounded parallel rewriting, maximal parallel rewriting with priorities and/or promoters and inhibitors. In some cases, we can express the control mechanisms by usual rewrite strategies, but in many cases new strategies or more general concepts like the strategy controllers [AL08] need to be defined. An interesting research direction is to adapt the ρ_{pg}-calculus for the population P systems with active cell with applications in modeling the immune system [BFFM06, SC01] which is a particular example of autonomous system.

7 Runtime Verification in the ρ_{pg}-Calculus

7.1 Introduction

In Chapter 3 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 4 we have defined a higher-order formalism, the ρ_{pg}-calculus, 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}-calculus 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 $\mathsf{AG}(p)$ (i.e., for all possible executions, at any moment, p is true), whereas a fatality property as $\mathsf{AF}(p)$ 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}-calculus. 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$^+$05, MRM$^+$08]. For instance, in [CRCFS04] the authors showed the expresiveness of CTL for formalizing a wide variety of biological queries on the possible behaviors of a biochemical system. Some typical biological queries are identified and they concern reachability of particular states from an initial state, properties about pathways (reachability of a state under a certain constraint), stability properties. Later, in [MRM$^+$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].

There are also various works on temporal logics for verifying graph transformation systems, for instance the $\mu\mathcal{L}2$ temporal logic [BCKK04]. Our goal is not to provide a new temporal logic for verifying properties of a port graph transition system, but to study a technique of embedding the temporal formulas in the state of the system together with behavior of the system, and reason on the satisfaction of the formulas as the system evolves.

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 7.2). 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 7.3 we extend the syntax of the ρ_{pg}-calculus with a set of CTL formulas that guard the worlds and multiverses. Then we extend the semantics of the ρ_{pg}-calculus 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}-calculus, for a biological system modeled using the ρ_{pg}-calculus. We end the chapter with some conclusions and perspectives.

7.2 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.

7.2.1 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 3. We briefly recall their construction in the following. Node identifiers are constructed based on integers, variables, and sequential composition (hence Id^*, where $Id = Int \cup VarId$). 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 $\langle \nabla_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{N}}, \nabla_{\mathscr{N}} \cup \mathcal{X}_{\mathscr{N}} \rangle$. 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 4: 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 7.1 summarizes these constructs.

Let $\sigma : \mathcal{X} \to Int^* \cup \nabla$ be a substitution 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 57.

Definition 66 (Satisfaction of port graph expressions)**.** *A port graph G satisfies a port graph expression γ if and only if there exists a substitution 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 $\mathcal{S}ol(\gamma \ll G) = \emptyset$.

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
		|	$\gamma_1 \curvearrowright \gamma_2$	connection
		|	$\gamma \circlearrowleft$	loop
		|	$\gamma \circlearrowleft$	self connection

Figure 7.1: Port graph expressions

7.2.2 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.

Definition 67 (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;*
- *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 7.2 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*

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

Figure 7.2: Structural formulas for port graphs

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 give produce the other Boolean operators, $\top$, $\wedge$, and $\rightarrow$.

Definition 68 (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:*

$$
\begin{array}{lcl}
G \models \top & & \\
G \not\models \bot & & \\
G \models \gamma & \Leftrightarrow & \exists \sigma \text{ such that } G \models^{\sigma} \gamma \\
G \models \neg\varphi & \Leftrightarrow & G \not\models \varphi \\
G \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow & G \models \varphi_1 \text{ and } G \models \varphi_2 \\
G \models \varphi_1 \vee \varphi_2 & \Leftrightarrow & G \models \varphi_1 \text{ or } G \models \varphi_2 \\
G \models \varphi_1 \rightarrow \varphi_2 & \Leftrightarrow & G \not\models \varphi_1 \text{ or } G \models \varphi_2 \\
G \models \diamondsuit\varphi & \Leftrightarrow & \exists G' \sqsubseteq G \text{ such that } G' \models \varphi
\end{array}
$$

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 $\mathcal{S}ol(\gamma \prec\!\!\ll G) \neq \emptyset$.

Proof. By definition $G \models \diamondsuit\gamma$ if and only if $\exists G' \sqsubseteq G$ such that $G' \models \gamma$, and $G' \models \gamma$ if and only if $\exists \sigma$ such that $G' \models^{\sigma} \gamma$. But $G' \models^{\sigma} \gamma$ is equivalent by definition to $G' \equiv \sigma^*(\gamma)$, i.e., $\sigma \in \mathcal{S}ol(\gamma \prec\!\!\ll G)$. □

The following proposition states that the structural satisfaction is up to the structural congruence relation on port graphs given by Definition 57. The proof is immediate using induction over the structure of φ.

Proposition 12. If $G \models \varphi$ and $G \equiv G'$ then $G' \models \varphi$.

7.2.3 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.

Definition 69 (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. *$L : S \to 2^{AP}$ 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 $\pi = s_0s_1s_2\ldots$ 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].

Definition 70 (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 $\mathcal{M} = (Ob(\mathbf{PGraph}), \rightarrow^{\bullet}_{\mathcal{R}}, L_{\mathbf{PGraph}})$ where:*

- *$Ob(\mathbf{PGraph})$ 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 $Ob(\mathbf{PGraph})$,*
- *$\rightarrow^{\bullet}_{\mathcal{R}}$ 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 $\mathcal{R}$);*
- *$L_{\mathbf{PGraph}} : Ob(\mathbf{PGraph}) \to \mathcal{P}(\mathcal{FS}(\nabla^{\mathcal{X}}))$ that maps each port graph G to a set of structural formulas that hold in G, that is, $L_{\mathbf{PGraph}}(G) = \{\varphi \in \mathcal{FS}(\nabla^{\mathcal{X}}) \mid G \models \varphi\}$.*

Definition 71 (Model for CTL for strategic port graph rewriting)**.** *A* model *for CTL for port graphs over $\nabla^{\mathcal{X}}$ and $\mathcal{S}(\mathcal{R})$ with strategic port graph rewriting is a Kripke structure system $\mathcal{M} = (Ob(\mathbf{PGraph}), \rightarrow^{\bullet}_{\mathcal{S}}, L_{\mathbf{PGraph}})$ where:*

- *$(\mathbf{PGraph})$ is the set of states given by the set of objects in the category* **PGraph***,*

- $\rightarrow_{\mathcal{S}(\mathcal{R})}$ *is the one-step strategic rewriting relation defined by a set* $\mathcal{S}(\mathcal{R})$ *of strategies built upon* $\mathcal{R}$ *on* $Ob(\mathbf{PGraph})$*,*
- $\rightarrow^{\bullet}_{\mathcal{S}(\mathcal{R})}$ *is the total relation associated to* $\rightarrow_{\mathcal{S}(\mathcal{R})}$ *which adds a self-loop for each irreducible state* G *(i.e.,* G *cannot be rewritten using strategies in* $\mathcal{S}(\mathcal{R})$*);*
- $L_{\mathbf{PGraph}} : Ob(\mathbf{PGraph}) \rightarrow \mathcal{P}(\mathcal{FS}(\nabla^{\mathcal{X}}))$ *that maps each port graph* G *to a set of structural formulas that hold in* G*, that is,* $L_{\mathbf{PGraph}}(G) = \{\varphi \in \mathcal{FS}(\nabla^{\mathcal{X}}) \mid G \models \varphi\}$*.*

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 7.3.

Definition 72 (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 $\mathsf{A}\psi$ and $\mathsf{E}\psi$ 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 $\mathsf{X}\phi$, $\mathsf{G}\phi$, $\mathsf{F}\phi$, $\phi_1 \mathsf{U} \phi_2$, $\phi_1 \mathsf{R} \phi_2$ 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 $\{\bot, \neg, \vee, \mathsf{EX}, \mathsf{EG}, \mathsf{EU}\}$.

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
		|	$\mathsf{A}\psi$	for all paths
		|	$\mathsf{E}\psi$	exists a path
Path formulas	ψ	::=		
		|	$\mathsf{X}\phi$	on the next state in the path
		|	$\mathsf{G}\phi$	globally on the path
		|	$\mathsf{F}\phi$	somewhere on the path
		|	$\phi_1 \mathsf{U} \phi_2$	until
		|	$\phi_1 \mathsf{R} \phi_2$	release

Figure 7.3: 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 $\mathcal{M}$.

Definition 73 (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:*

$$
\begin{array}{lcl}
\mathcal{M}, G \models \neg\phi & \Leftrightarrow & \mathcal{M}, G \not\models \phi \\
\mathcal{M}, G \models \phi_1 \wedge \phi_2 & \Leftrightarrow & \mathcal{M}, G \models \phi_1 \text{ and } \mathcal{M}, G \models \phi_2 \\
\mathcal{M}, G \models \phi_1 \vee \phi_2 & \Leftrightarrow & \mathcal{M}, G \models \phi_1 \text{ or } \mathcal{M}, G \models \phi_2 \\
\mathcal{M}, G \models \phi_1 \rightarrow \phi_2 & \Leftrightarrow & \mathcal{M}, G \not\models \phi_1 \text{ or } \mathcal{M}, G \models \phi_2 \\
\mathcal{M}, G \models \mathsf{A}\psi & \Leftrightarrow & \text{for every path } \pi \text{ starting from } G,\ \mathcal{M}, \pi \models \psi \\
\mathcal{M}, G \models \mathsf{E}\psi & \Leftrightarrow & \text{there is a path } \pi \text{ starting from } G \text{ such that } \mathcal{M}, \pi \models \psi
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{M}, \pi \models \mathsf{X}\phi & \Leftrightarrow & \mathcal{M}, \pi^1 \models \phi \\
\mathcal{M}, \pi \models \mathsf{G}\phi & \Leftrightarrow & \textit{for all } k \geq 0,\ \mathcal{M}, \pi^k \models \phi \\
\mathcal{M}, \pi \models \mathsf{F}\phi & \Leftrightarrow & \textit{there exists } k \geq 0 \textit{ such that } \mathcal{M}, \pi^k \models \phi \\
\mathcal{M}, \pi \models \phi_1 \mathsf{U} \phi_2 & \Leftrightarrow & \textit{there exists a } k \geq 0 \textit{ such that } \mathcal{M}, \pi^k \models \phi_2 \textit{ and} \\
 & & \textit{for all } 0 \leq i < k,\ \mathcal{M}, \pi^i \models \phi_1 \\
\mathcal{M}, \pi \models \phi_1 \mathsf{R} \phi_2 & \Leftrightarrow & \textit{for all } j \geq 0,\ \textit{if for every } i < j\ \mathcal{M}, \pi^i \not\models \phi_1 \textit{ then} \\
 & & \mathcal{M}, \pi^j \models \phi_2
\end{array}
$$

7.3 Embedding Verification in the ρ_{pg}-Calculus: the ρ^v_{pg}-Calculus

We extend the ρ_{pg}-calculus 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}-calculus with verification of formulas; we call this calculus the ρ^v_{pg}-calculus. 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.

7.3.1 Syntax

We add to the syntax of the ρ_{pg}-calculus 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}-calculus 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:

$$\varphi ::= \top \mid \ \bot \ \mid \ \diamondsuit\gamma \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \rightarrow \varphi_2$$

Any port graph expression γ must be quantified by a somewhere modality in order to be a structural formula since a formula $\diamondsuit\gamma$ supposes solving a submatching problem which is supported currently in the ρ_{pg}-calculus.

The port graph expressions used in the structural formulas presented for CTL for port graphs in Section 7.2 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 as follows:

$$\begin{array}{lcl}
\tau(\top) & = & \mathsf{id} \\
\tau(\bot) & = & \mathtt{fail} \\
\tau(\diamondsuit\gamma) & = & \gamma \Rightarrow \gamma \\
\tau(\neg\varphi) & = & \mathtt{not}(\tau(\varphi)) \\
\tau(\varphi_1 \wedge \varphi_2) & = & \mathtt{seq}(\tau(\varphi_1), \tau(\varphi_2)) \\
\tau(\varphi_1 \vee \varphi_2) & = & \mathtt{first}(\tau(\varphi_1), \tau(\varphi_2)) \\
\tau(\varphi_1 \rightarrow \varphi_2) & = & X \Rightarrow \mathtt{seq}(\tau(\varphi_1), \mathtt{first}(\mathsf{stk} \Rightarrow X, \tau(\varphi_2)))@X
\end{array}$$

Proposition 13. Let φ be a structural formula in the ρ^v_{pg}-calculus and G a port graph molecule. Then $\tau(\varphi)@G$ reduces either to $\{[G]\}$ or to $\{[\mathsf{stk}]\}$.

Proof. We proceed by structural induction on φ.

- If $(\varphi = \top)$, then $\tau(\varphi)@G = \mathsf{id}@G = (X \Rightarrow X)@G \longrightarrow^* \{[G]\}$.
- If $(\varphi = \bot)$, then $\tau(\varphi)@G = \mathtt{fail}@G = (X \Rightarrow \mathsf{stk})@G \longrightarrow^* \{[\mathsf{stk}]\}$.
- If $(\varphi = \diamondsuit\gamma)$, then $\tau(\varphi)@G = (\gamma \Rightarrow \gamma)@G$. There are two possibilities: either γ submatches G, and, in consequence $(\gamma \Rightarrow \gamma)@G \longrightarrow^* \{[G]\}$, or γ does not submatch G, hence $(\gamma \Rightarrow \gamma)@G \longrightarrow^* \{[\mathsf{stk}]\}$.
- If $(\varphi = \neg\varphi')$, then

  $$\tau(\varphi)@G = \mathtt{not}(\tau(\varphi'))@G \longrightarrow^* \{[\mathtt{first}(\mathsf{stk} \Rightarrow G, X \Rightarrow \mathsf{stk})@(\tau(\varphi')@G)]\}$$

  By induction hypothesis, either $\tau(\varphi')@G \longrightarrow^* \{[\mathsf{stk}]\}$ and, in consequence $\tau(\varphi)@G \longrightarrow^* \{[G]\}$, or $\tau(\varphi')@G \longrightarrow^* \{[G]\}$, hence $\tau(\varphi)@G \longrightarrow^* \{[\mathsf{stk}]\}$.
- If $(\varphi = \varphi_1 \wedge \varphi_2)$, then $\tau(\varphi)@G = \mathtt{seq}(\tau(\varphi_1), \tau(\varphi_2)) = \tau(\varphi_2)@\tau(\varphi_1)@G$. If $\tau(\varphi_1)@G$ reduces to $\{[\mathsf{stk}]\}$, then $\tau(\varphi)@G$ reduces to $\{[\mathsf{stk}]\}$ as well; otherwise, $\tau(\varphi_1)@G$ reduces to $\{[G]\}$, then $\tau(\varphi_2)@\{[G]\} \longrightarrow_{ac} \{[\tau(\varphi_2)@G]\}$; and since, by induction hypothesis, $\tau(\varphi_2)@G$ reduces to either $\{[G]\}$ or $\{[\mathsf{stk}]\}$, we obtain the same result for the reduction of $\tau(\varphi)@G$.
- If $(\varphi = \varphi_1 \vee \varphi_2)$, then $\tau(\varphi)@G = \mathtt{first}(\tau(\varphi_1), \tau(\varphi_2))$. If $\tau(\varphi_1)@G$ reduces to $\{[\mathsf{stk}]\}$, then the result of reducing $\tau(\varphi)@G$ coincides with the result of reducing $\tau(\varphi_2)@G$ which, by induction hypothesis, is either $\{[G]\}$ or $\{[\mathsf{stk}]\}$. If $\tau(\varphi_1)@G$ reduces to $\{[G]\}$, then this is also the result of reducing $\tau(\varphi)@G$.
- If $(\varphi = \varphi_1 \rightarrow \varphi_2)$, then

  $$\tau(\varphi)@G = (X \Rightarrow \mathtt{seq}(\tau(\varphi_1), \mathtt{first}(\mathsf{stk} \Rightarrow X, \tau(\varphi_2)))@X)@G \longrightarrow^* \{[\mathtt{seq}(\tau(\varphi_1), \mathtt{first}(\mathsf{stk} \Rightarrow G, \tau(\varphi_2)))@G]\}$$

If $\tau(\varphi_1)@G \longrightarrow^* \{[\mathsf{stk}]\}$ then the reduction continues with:

$$\mathtt{first}(\mathsf{stk} \Rightarrow G, \tau(\varphi_2)))@\{[\mathsf{stk}]\} \longrightarrow^* \{[G]\}$$

where the first argument of the strategy $\mathtt{first}(\mathsf{stk} \Rightarrow G, \tau(\varphi_2)))$ is applied.

If $\tau(\varphi_1)@G \longrightarrow^* \{[G]\}$ then the second argument of the strategy $\mathtt{first}(\mathsf{stk} \Rightarrow G, \tau(\varphi_2)))$ is applied, hence, by induction hypothesis, the result of the reduction is either $\{[G]\}$ or $\{[\mathsf{stk}]\}$.

□

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}-calculus. If $G \models \varphi$ then $\tau(\varphi)@G \longrightarrow^* \{[G]\}$, otherwise, if $G \not\models \varphi$ then $\tau(\varphi)@G \longrightarrow^* \{[\mathsf{stk}]\}$.

Proof. We prove the proposition inductively on the structure of the formula φ.

- If $\varphi = \top$, $G \models \top$ is always true, and $\tau(\top)@G = \mathsf{id}@G \longrightarrow^* \{[G]\}$.
- If $\varphi = \bot$, $G \not\models \bot$ is always true, and $\tau(\bot)@G = \mathtt{fail}@G \longrightarrow^* \{[\mathsf{stk}]\}$.
- If $\varphi = \Diamond\gamma$ such that $G \models \Diamond\gamma$, then $\mathcal{S}ol(\gamma \prec\!\!\prec G) \neq \emptyset$. In consequence, $\tau(\varphi)@G = (\gamma \Rightarrow \gamma)@G \longrightarrow^* \{[G]\}$. Otherwise, if $G \not\models \Diamond\gamma$, then γ does not submatch G, hence $\tau(\varphi)@G \longrightarrow^* \{[\mathsf{stk}]\}$.
- If $\varphi = \neg\varphi'$ such that $G \models \neg\varphi'$, then $G \not\models \varphi'$, hence, by induction hypothesis, $\tau(\varphi')@G \longrightarrow^* \{[\mathsf{stk}]\}$. Then

  $\tau(\varphi)@G = \mathtt{not}(\tau(\varphi'))@G \longrightarrow^* \{[\mathtt{first}(\mathsf{stk} \Rightarrow G, X \Rightarrow \mathsf{stk})@(\tau(\varphi')@G)]\} \longrightarrow^* \{[\mathtt{first}(\mathsf{stk} \Rightarrow G, X \Rightarrow \mathsf{stk})@\mathsf{stk}]\} \longrightarrow^* \{[G]\}$

  But if $G\neg \models \neg\varphi'$, then $G \models \varphi'$, and, by induction hypothesis, $\tau(\varphi')@G \longrightarrow^* \{[G]\}$. Then $\tau(\varphi)@G \longrightarrow^* \{[\mathtt{first}(\mathsf{stk} \Rightarrow G, X \Rightarrow \mathsf{stk})@G]\} \longrightarrow^* \{[\mathsf{stk}]\}$.
- If $\varphi = \varphi_1 \wedge \varphi_2$ such that $G \models \varphi_1 \wedge \varphi_2$, then $G \models \varphi_1$ and $G \models \varphi_2$. Then, by induction hypothesis, both $\tau(\varphi_1)@G$ and $\tau(\varphi_1)@G$ do not reduce to $\{[\mathsf{stk}]\}$; from Proposition 13 it follows that they both reduce to G. And since $\tau(\varphi)@G = \mathtt{seq}(\tau(\varphi_1), \tau(\varphi_2))@G = \tau(\varphi_2)@\tau(\varphi_1)@G$, then $\tau(\varphi)@G$ reduces to $\{[G]\}$.

  But if $G \not\models \varphi_1 \wedge \varphi_2$, then $G \models \neg\varphi_1 \vee \neg\varphi_2$, which implies that $G \not\models \varphi_1$ or $G \not\models \varphi_1$. Hence either $\tau(\varphi_1)@G \longrightarrow^* \{[\mathsf{stk}]\}$ or $\tau(\varphi_2)@G \longrightarrow^* \{[\mathsf{stk}]\}$. In consequence $\tau(\varphi)@G = \mathtt{seq}(\tau(\varphi_1), \tau(\varphi_2))@G \longrightarrow^* \{[\mathsf{stk}]\}$.

- If $\varphi = \varphi_1 \vee \varphi_2$ such that $G \models \varphi_1 \vee \varphi_2$, then $G \models \varphi_1$ or $G \models \varphi_2$. Then, by induction hypothesis, at least one of $\tau(\varphi_1)@G$ and $\tau(\varphi_1)@G$ do not reduce to $\{[\mathsf{stk}]\}$; from Proposition 13 it follows that at least of the them reduces to G. And since $\tau(\varphi)@G = \mathtt{first}(\tau(\varphi_1), \tau(\varphi_2))@G$ then $\tau(\varphi)@G$ reduces to $\{[G]\}$.

  If $G \not\models \varphi_1 \vee \varphi_2$, then $G \models \neg\varphi_1 \wedge \neg\varphi_2$, which implies that $G \not\models \varphi_1$ and $G \not\models \varphi_1$. Hence both $\tau(\varphi_1)@G$ and $\tau(\varphi_2)@G$ reduce to $\{[\mathsf{stk}]\}$. In consequence $\tau(\varphi)@G = \mathtt{first}(\tau(\varphi_1), \tau(\varphi_2))@G \longrightarrow^* \{[\mathsf{stk}]\}$.

- If $\varphi = \varphi_1 \rightarrow \varphi_2$ such that $G \models \varphi_1 \rightarrow \varphi_2$, then $G \models \neg\varphi_1 \vee \varphi_2$, hence $G \not\models \varphi_1$ or $G \models \varphi_2$. By induction hypothesis, we have $\tau(\varphi_1)@G \longrightarrow^* \{[\mathsf{stk}]\}$ or $\tau(\varphi_2)@G \longrightarrow^* \{[G]\}$.

  By definition, $\tau(\varphi)@G = (X \Rightarrow \mathtt{seq}(\tau(\varphi_1), \mathtt{first}(\mathsf{stk} \Rightarrow X, \tau(\varphi_2)))@X)@G$ which reduces to $\{[\mathtt{seq}(\tau(\varphi_1), \mathtt{first}(\mathsf{stk} \Rightarrow G, \tau(\varphi_2)))@G]\}$. If $\tau(\varphi_1)@G \longrightarrow^* \{[\mathsf{stk}]\}$ then $\tau(\varphi)@G$ will reduce to $\{[G]\}$, but if $\tau(\varphi_1)@G \longrightarrow^* \{[G]\}$, then $\tau(\varphi)@G$ will reduce to $\{[\tau(\varphi_2)@G]\}$ which reduces to $\{[G]\}$.

  But if $G \not\models \varphi_1 \rightarrow \varphi_2$, then $G \models \neg(\neg\varphi_1 \vee \varphi_2)$, hence $G \models \varphi_1$ and $G \not\models \varphi_2$. By induction hypothesis, we have $\tau(\varphi_1)@G \longrightarrow^* \{[G]\}$ and $\tau(\varphi_2)@G \longrightarrow^* \{[\mathsf{stk}]\}$. Then $\tau(\varphi)@G$ will reduce to $\{[\tau(\varphi_2)@G]\}$ which reduces to $\{[\mathsf{stk}]\}$.

□

Proposition 15 (Completeness of the encoding)**.** Let G be a port graph molecule and φ a structural formula in the ρ_{pg}-calculus. If $\tau(\varphi)@G \longrightarrow^* \{[G]\}$ then $G \models \varphi$, otherwise, if $\tau(\varphi)@G \longrightarrow^* \{[\mathsf{stk}]\}$ then $G \not\models \varphi$.

Proof. We proceed by structural induction on φ.

- If $\varphi = \top$, then it is only possible to have $\tau(\top)@G = \mathsf{id}@G \longrightarrow^* \{[G]\}$, and by definition $G \models \top$ is always true.

- If $\varphi = \bot$, then it is always the case that $\tau(\bot)@G = \mathtt{fail}@G \longrightarrow^* \{[\mathsf{stk}]\}$ and, by definition, $G \not\models \bot$.

- If $\varphi = \Diamond\gamma$ such that $\tau(\varphi)@G \longrightarrow^* \{[G]\}$, then, since $\tau(\varphi)@G = (\gamma \Rightarrow \gamma)@G$ it follows that $\mathcal{S}ol(\gamma \prec\!\!\prec G) \neq \emptyset$, hence, by definition, $G \models \Diamond\gamma$.

- If $\varphi = \neg\varphi'$ such that $\tau(\varphi)@G \longrightarrow^* \{[G]\}$, then $\tau(\varphi')@G \longrightarrow^* \{[\mathsf{stk}]\}$, hence $G \not\models \varphi'$. It follows that $G \models \neg\varphi'$, i.e., $G \models \varphi$.

  If $\tau(\varphi)@G \longrightarrow^* \{\mathsf{stk}\}$ then $\tau(\varphi')@G \longrightarrow^* \{\mathsf{stk}\}$, hence $G \models \varphi'$ hence $G \not\models \neg\varphi'$, i.e., $G \not\models \varphi$.

- If $\varphi = \varphi_1 \wedge \varphi_2$ such that $\tau(\varphi)@G \longrightarrow^* \{[G]\}$ then $\mathtt{seq}(\tau(\varphi_1), \tau(\varphi_2))@G = \tau(\varphi_2)@\tau(\varphi_1)@G$, i.e., $\tau(\varphi_1)@G \longrightarrow^* \{[G]\}$ and $\tau(\varphi_2)@G \longrightarrow^* \{[G]\}$. By induction hypothesis, $G \models \varphi_1$ and $G \models \varphi_2$, therefore $G \models \varphi_1 \wedge \varphi_2$.

  But if $\tau(\varphi)@G \longrightarrow^* \{[\mathsf{stk}]\}$, then either $\tau(\varphi_1)@G \longrightarrow^* \{[\mathsf{stk}]\}$ or $\tau(\varphi_2)@G \longrightarrow^* \{[\mathsf{stk}]\}$. By induction hypothesis, $G \not\models \varphi_1$ or $G \not\models \varphi_2$, in consequence $G \not\models \varphi_1 \wedge \varphi_2$.

- If $\varphi = \varphi_1 \vee \varphi_2$ such that $\tau(\varphi)@G \longrightarrow^* \{[G]\}$, then $\texttt{first}(\tau(\varphi_1), \tau(\varphi_2))@G \longrightarrow^* \{[G]\}$ which means that either $\tau(\varphi_1)@G \longrightarrow^* \{[G]\}$ or $\tau(\varphi_2)@G \longrightarrow^* \{[G]\}$. This is equivalent, by induction hypothesis, with $G \models \varphi_1$ or $G \models \varphi_2$, hence $G \models \varphi_1 \vee \varphi_2$.

  But if $\tau(\varphi)@G \longrightarrow^* \{[\mathsf{stk}]\}$, it follows that $\tau(\varphi_1)@G \longrightarrow^* \{[\mathsf{stk}]\}$ and $\tau(\varphi_2)@G \longrightarrow^* \{[\mathsf{stk}]\}$. By induction hypothesis we obtain $G \not\models \varphi_1$ and $G \not\models \varphi_2$, hence $G \models \neg\varphi_1$ and $G \models \neg\varphi_2$, equivalent to $G \models (\neg\varphi_1) \wedge (\neg\varphi_2)$, which is equivalent to $G \not\models \varphi_1 \vee \varphi_2$.

- If $\varphi = \varphi_1 \rightarrow \varphi_2$ such that $\tau(\varphi)@G \longrightarrow^* \{[G]\}$, then $\tau(\varphi)@G = (X \Rightarrow \texttt{seq}(\tau(\varphi_1), \texttt{first}(\mathsf{stk} \Rightarrow X, \tau(\varphi_2)))@X)@G \longrightarrow^* \{[\texttt{seq}(\tau(\varphi_1), \texttt{first}(\mathsf{stk} \Rightarrow G, \tau(\varphi_2)))@G]\} \longrightarrow^* \{[G]\}$. It follows that $\tau(\varphi_1)@G \longrightarrow^* \{[G]\}$ and $\tau(\varphi_2)@G \longrightarrow^* \{[G]\}$. Then, by induction hypothesis, $G \models \varphi_1$ and $G \models \varphi_2$, which implies that $G \models \varphi_1 \rightarrow \varphi_2$.

  If $\tau(\varphi)@G \longrightarrow^* \{[\mathsf{stk}]\}$ then $\tau(\varphi_1)@G \longrightarrow^* \{[G]\}$ and $\tau(\varphi_1)@G \longrightarrow^* \{[\mathsf{stk}]\}$. By induction hypothesis, $G \models \varphi_1$ and $G \not\models \varphi_2$; this is equivalent to $G \models \varphi_1 \wedge \neg\varphi_2$ which is equivalent to $G \models \neg(\neg\varphi_1 \vee \varphi_2)$, hence $G \not\models \varphi_1 \rightarrow \varphi_2$.

□

Remark 8. Since the conjunction and the disjunction are commutative then we also have:

$$\begin{aligned} \tau(\varphi_1 \wedge \varphi_2) &= \texttt{seq}(\tau(\varphi_2), \tau(\varphi_1)) \\ \tau(\varphi_1 \vee \varphi_2) &= \texttt{first}(\tau(\varphi_2), \tau(\varphi_1)) \end{aligned}$$

Obviously, this does not imply that the strategy operators `seq` and `first` are commutative.

Since we can consider for instance $\{\bot, \neg, \vee\}$ as a minimal set of boolean operators such that they can define the other boolean operators like $\top$, $\wedge$, $\rightarrow$, then we can deduce some equalities between the strategies encoding the structural formulas. For instance, since $\varphi_1 \wedge \varphi_2 = \neg(\neg\varphi_1 \vee \neg\varphi_2)$ and $\varphi_1 \rightarrow \varphi_2 = \neg\varphi_1 \vee \varphi_2$ we have respectively:

$$\texttt{seq}(\tau(\varphi_1), \tau(\varphi_2)) = \texttt{not}(\texttt{first}(\texttt{not}(\tau(\varphi_1)), \texttt{not}(\tau(\varphi_2))))$$

$$X \Rightarrow \texttt{seq}(\tau(\varphi_1), \texttt{first}(\mathsf{stk} \Rightarrow X, \tau(\varphi_2)))@X = \texttt{seq}(\texttt{not}(\varphi_1), \varphi_2)$$

We can continue replacing the strategy `not` by the abstraction it denotes such that we have only the strategies `seq` and `first`.

Example 27 (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}-calculus

We include in the ρ^{v}_{pg}-calculus 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}-calculus is the following:

$$
\begin{array}{lcl}
\mathcal{F}_1 & ::= & \varphi \\
\mathcal{F}_2 & ::= & \mathsf{A}\varphi \mid \mathsf{E}\varphi \\
\mathcal{F}_3 & ::= & \mathsf{AX}\varphi \mid \mathsf{EX}\varphi \mid \mathsf{AF}\varphi \mid \mathsf{EF}\varphi \mid \mathsf{AG}\varphi \mid \mathsf{EG}\varphi \mid \mathsf{A}(\varphi\mathsf{U}\varphi) \mid \mathsf{E}(\varphi\mathsf{U}\varphi) \mid \mathsf{A}(\varphi\mathsf{R}\varphi) \mid \mathsf{E}(\varphi\mathsf{R}\varphi) \\
\mathcal{F}_v & ::= & \mathcal{F}_1 \mid \mathcal{F}_3 \mid \neg\mathcal{F}_v \mid \mathcal{F}_v \vee \mathcal{F}_v \mid \mathcal{F}_v \wedge \mathcal{F}_v \\
\mathcal{F}_w & ::= & \mathcal{F}_2 \mid \mathcal{F}_3 \mid \neg\mathcal{F}_w \mid \mathcal{F}_w \vee \mathcal{F}_w \mid \mathcal{F}_w \wedge \mathcal{F}_w
\end{array}
$$

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, 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 $\mathsf{A}\varphi$ or $\mathsf{E}\varphi$ is equivalent to the juxtaposition of the component worlds guarded by F:

$$\{W_1 \ldots W_n\} \models^? F \equiv \{W_1 \models^? F \ldots W_n \models^? F\} \text{ if } F \neq \mathsf{A}\varphi \vee F \neq \mathsf{E}\varphi$$

7.3.2 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 then 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:

$$S^v_\varphi = [X] \Rightarrow \texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow [X] \models \varphi, X_2 \Rightarrow [X] \not\models \varphi)@X$$

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.

Proof. Since $G \not\models \varphi$ is equivalent to the failure of the application of the strategy $\tau(\varphi)$ on G, the result stated by the proposition is given by the definition of the strategy operator `ifThenElse`. □

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., $\tau(\varphi)@G \longrightarrow^* \{[\mathsf{stk}]\}$, 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:

- $\mathsf{A}\varphi$ is satisfied by a multiverse W if φ is satisfied in each world from W;

$$\{V_1 \dots V_n\} \models \mathsf{A}\varphi \Leftrightarrow \forall i,\ 1 \leq i \leq n, V_i \models \varphi$$

- $\mathsf{E}\varphi$ is satisfied by a multiverse W if φ is satisfied in one world from W:

$$\{V_1 \dots V_n\} \models \mathsf{E}\varphi \Leftrightarrow \exists i, 1 \leq i \leq n, V_i \models \varphi$$

We encode the formulas $\mathsf{A}\varphi$ and $\mathsf{E}\varphi$ 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 $S^w_{\mathsf{A}\varphi}$ 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:

$$\begin{aligned}
&S^w_{\mathsf{A}\varphi} = \texttt{first}(S_1, S_2)\\
&S_1 = \{[X]\} \Rightarrow \texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow \{[X]\} \models \mathsf{A}\varphi, X_2 \Rightarrow \{[X]\} \not\models \mathsf{A}\varphi)@X\\
&S_2 = \{[X]\ Z\} \Rightarrow\\
&\qquad \texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow \{[X] \models \varphi\ S^w_{\mathsf{A}\varphi}@Z\}, X_2 \Rightarrow \{[X]\ Z\} \not\models \mathsf{A}\varphi)@X
\end{aligned}$$

We define the strategy $S^w_{\mathsf{E}\varphi}$ for testing the satisfiability of $\mathsf{E}\varphi$ in a multiverse W also as a left-biased choice similarly to $S^w_{\mathsf{A}\varphi}$:

$$\begin{aligned}
&S^w_{\mathsf{E}\varphi} = \texttt{first}(S_3, S_4)\\
&S_3 = \{[X]\} \Rightarrow \texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow \{[X]\} \models \mathsf{E}\varphi, X_2 \Rightarrow \{[X]\} \not\models \mathsf{E}\varphi)@X\\
&S_4 = \{[X]\ Z\} \Rightarrow\\
&\qquad \texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow \{[X]\ Z\} \models \mathsf{E}\varphi, X_2 \Rightarrow \{[X] \not\models \varphi\ S^w_{\mathsf{E}\varphi}@Z\})@X
\end{aligned}$$

Proposition 17. Let $W = \{[G_1] \dots [G_n]\}$ be a multiverse, and φ a structural formula. Then $S^w_{\mathsf{A}\varphi}@W$ reduces to either $W \models \mathsf{A}\varphi$ if $G_i \models \varphi$ for all i, $1 \leq i \leq n$, and to $W \not\models \mathsf{A}\varphi$ otherwise.

Proposition 18. Let $W = \{[G_1] \dots [G_n]\}$ be a multiverse, and φ a structural formula. Then $S^w_{\mathsf{E}\varphi}@W$ reduces to either $W \models \mathsf{E}\varphi$ if there exists i, $1 \leq i \leq n$, such that $G_i \models \varphi$, and to $W \not\models \mathsf{E}\varphi$ 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 $\mathsf{A}\varphi$ or $\mathsf{E}\varphi$ is equivalent to the result of applying the strategy $S^w_{\mathsf{A}\varphi}$ or $S^w_{\mathsf{E}\varphi}$ respectively on the multiverse:

$$W \models^? \mathsf{A}\varphi \equiv S^w_{\mathsf{A}\varphi}@W \qquad\qquad W \models^? \mathsf{E}\varphi \equiv S^w_{\mathsf{E}\varphi}@W$$

The satisfiability test of a structural formula for a multiverse W is propagated by the strategies $S^w_{\mathsf{A}\varphi}$ and $S^w_{\mathsf{E}\varphi}$ 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:

$$\begin{array}{l}
\{V \models \varphi \; W \models \mathsf{A}\varphi\} \equiv \{V \; W\} \models \mathsf{A}\varphi \\
\{V \not\models \varphi \; W \models \mathsf{A}\varphi\} \equiv \{V \models \varphi \; W \not\models \mathsf{A}\varphi\} \equiv \{V \; W\} \not\models \mathsf{A}\varphi \\
\{V \not\models \varphi \; W \not\models \mathsf{E}\varphi\} \equiv \{V \; W\} \not\models \mathsf{E}\varphi \\
\{V \not\models \varphi \; W \models \mathsf{E}\varphi\} \equiv \{V \models \varphi \; W \not\models \mathsf{E}\varphi\} \equiv \{V \; W\} \models \mathsf{E}\varphi
\end{array}$$

Reduction Relation on Guarded Worlds and Multiverses

We define now the reduction relation $\Rrightarrow$ between guarded worlds and 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).

$\mathsf{AX}\varphi$ and $\mathsf{EX}\varphi$

If a world V having a guard $\models^? \mathsf{AX}\varphi$ reduces in a step to the multiverse W, the guard reduces as well to $\models \mathsf{AX}\varphi$ if every world in W satisfies φ, or $\not\models \mathsf{AX}\varphi$ otherwise. We reason similarly for a world guarded by $\models^? \mathsf{EX}\varphi$ by testing if the multiverse the world reduces to in a step contains a world satisfying φ.

$$V \models^? \mathsf{AX}\varphi \Rrightarrow \texttt{first}(W \models \mathsf{A}\varphi \Rightarrow W \models \mathsf{AX}\varphi, W \not\models \mathsf{A}\varphi \Rightarrow W \not\models \mathsf{AX}\varphi)@(W \models^? \mathsf{A}\varphi) \quad \text{if } V \mapsto W \tag{7.1}$$

$$V \models^? \mathsf{EX}\varphi \Rrightarrow \texttt{first}(W \models \mathsf{E}\varphi \Rightarrow W \models \mathsf{EX}\varphi, W \not\models \mathsf{E}\varphi \Rightarrow W \not\models \mathsf{EX}\varphi)@(W \models^? \mathsf{E}\varphi) \quad \text{if } V \mapsto W \tag{7.2}$$

$\mathsf{AG}\varphi$

If a formula $\mathsf{AG}\varphi$ 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.

$$[G] \models^? \mathsf{AG}\varphi \Rrightarrow \texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow W \models^? \mathsf{AG}\varphi, X_2 \Rightarrow W \not\models \mathsf{AG}\varphi)@G \quad \text{if } [G] \mapsto W \tag{7.3}$$

Remark 9. The satisfaction problem of a formula of the type $\mathsf{AG}\varphi$, $\mathsf{EG}\varphi$, $\mathsf{AF}\varphi$, or $\mathsf{EF}\varphi$ 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 $\mathsf{AG}\varphi$ and the irreducible world $[G]$, then we have:

$$[G] \models^? \mathsf{AG}\varphi \equiv \texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow [G] \models \mathsf{AG}\varphi, X_2 \Rightarrow [G] \not\models \mathsf{AG}\varphi)@G$$
$$\text{if } [G] \text{ is } \mapsto\text{-irreducible}$$

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:

$$\{V \not\models \mathsf{AG}\varphi\ W \models^? \mathsf{AG}\varphi\} \equiv \{V \models^? \mathsf{AG}\varphi\ W \not\models \mathsf{AG}\varphi\} \equiv \{V\ W\} \not\models \mathsf{AG}\varphi$$
$$\{V \not\models \mathsf{AG}\varphi\ W \models \mathsf{AG}\varphi\} \equiv \{V \models \mathsf{AG}\varphi\ W \not\models \mathsf{AG}\varphi\} \equiv \{V\ W\} \not\models \mathsf{AG}\varphi$$
$$\{V \not\models \mathsf{AG}\varphi\ W \not\models \mathsf{AG}\varphi\} \equiv \{V\ W\} \not\models \mathsf{AG}\varphi$$
$$\{V \models \mathsf{AG}\varphi\ W \models \mathsf{AG}\varphi\} \equiv \{V\ W\} \models \mathsf{AG}\varphi$$

$\boxed{\mathsf{EG}\varphi}$

The reduction in the ρ^v_{pg}-calculus of a world guarded by a formula $\mathsf{EG}\varphi$ is similar to the reduction of a world guarded by $\mathsf{AG}\varphi$. 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.

$$[G] \models^? \mathsf{EG}\varphi \Rrightarrow \texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow W \models^? \mathsf{EG}\varphi, X_2 \Rightarrow W \not\models \mathsf{EG}\varphi)@G$$
$$\text{if } [G] \mapsto W \tag{7.4}$$

The information on the satisfaction or in-satisfaction of a formula $\mathsf{EG}\varphi$ in a multiverse is regrouped as follows:

$$\{V \not\models \mathsf{EG}\varphi\ W \not\models \mathsf{EG}\varphi\} \equiv \{V\ W\} \not\models \mathsf{EG}\varphi$$
$$\{V \not\models \mathsf{EG}\varphi\ W \models \mathsf{EG}\varphi\} \equiv \{V \models \mathsf{EG}\varphi\ W \not\models \mathsf{EG}\varphi\} \equiv \{V\ W\} \models \mathsf{EG}\varphi$$

$\boxed{\mathsf{AF}\varphi}$

When testing the satisfiability of a formula $\mathsf{AF}\varphi$ 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.

$$[G] \models^? \mathsf{AF}\varphi \Rrightarrow$$
$$\texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow W \models \mathsf{AF}\varphi, X_2 \Rightarrow W \models^? \mathsf{AF}\varphi)@G \text{ if } [G] \mapsto W \tag{7.5}$$

We reassemble the information on the satisfaction of a formula $\mathsf{AF}\varphi$ on a multiverse as follows:

$$\{V \models \mathsf{AF}\varphi\ W \models \mathsf{AF}\varphi\} \equiv \{V\ W\} \models \mathsf{AF}\varphi$$
$$\{V \models \mathsf{AF}\varphi\ W \not\models \mathsf{AF}\varphi\} \equiv \{V \not\models \mathsf{AF}\varphi\ W \models \mathsf{AF}\varphi\} \equiv \{V\ W\} \not\models \mathsf{AF}\varphi$$
$$\{V \not\models \mathsf{AF}\varphi\ W \not\models \mathsf{AF}\varphi\} \equiv \{V\ W\} \not\models \mathsf{AF}\varphi$$

$\boxed{\mathsf{EF}\varphi}$

The reduction in ρ^v_{pg}-calculus of a world guarded by a formula $\mathsf{EF}\varphi$ is similar to the reduction of a world guarded by $\mathsf{AF}\varphi$ on the same basis as for $\mathsf{EG}\varphi$ and $\mathsf{AG}\varphi$.

$$[G] \models^? \mathsf{EF}\varphi \Rrightarrow$$
$$\texttt{ifThenElse}(\tau(\varphi), X_1 \Rightarrow W \models \mathsf{EF}\varphi, X_2 \Rightarrow W \models^? \mathsf{EF}\varphi)@G \text{ if } [G] \Mapsto W \tag{7.6}$$

The satisfaction information for $\mathsf{EF}\varphi$ is grouped in a multiverse as follows:

$$\{V \models \mathsf{EF}\varphi\ W \models^? \mathsf{EF}\varphi\} \equiv \{V \models^? \mathsf{EF}\varphi\ W \models \mathsf{EF}\varphi\} \equiv \{V\ W\} \models \mathsf{EF}\varphi$$
$$\{V \models \mathsf{EF}\varphi\ W \not\models \mathsf{EF}\varphi\} \equiv \{V \not\models \mathsf{EF}\varphi\ W \models \mathsf{EF}\varphi\} \equiv \{V\ W\} \models \mathsf{EF}\varphi$$
$$\{V \models \mathsf{EF}\varphi\ W \models \mathsf{EF}\varphi\} \equiv \{V\ W\} \models \mathsf{EF}\varphi$$
$$\{V \not\models \mathsf{EF}\varphi\ W \not\models \mathsf{EF}\varphi\} \equiv \{V\ W\} \not\models \mathsf{EF}\varphi$$

$\boxed{\mathsf{A}(\varphi_1\mathsf{U}\varphi_2)}$

If the second structural formula φ_2 is satisfied by G, then the formula $\mathsf{A}(\varphi_1\mathsf{U}\varphi_2)$ 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 $\mathsf{A}(\varphi_1\mathsf{U}\varphi_2)$ does not hold.

$$\begin{aligned}
&[G] \models^? \mathsf{A}(\varphi_1\mathsf{U}\varphi_2) \Rrightarrow \\
&\texttt{ifThenElse}(\tau(\varphi_2), \\
&\qquad X_1 \Rightarrow W \models \mathsf{A}(\varphi_1\mathsf{U}\varphi_2), \\
&\qquad X_2 \Rightarrow (\texttt{ifThenElse}(\tau(\varphi_1), \\
&\qquad\qquad X_3 \Rightarrow W \models^? \mathsf{A}(\varphi_1\mathsf{U}\varphi_2), \\
&\qquad\qquad X_4 \Rightarrow W \not\models \mathsf{A}(\varphi_1\mathsf{U}\varphi_2))@G))@G \\
&\quad \text{if } [G] \Mapsto W
\end{aligned} \tag{7.7}$$

If the current world $[G]$ is irreducible with respect to $\Mapsto$, then the formula $\mathsf{A}(\varphi_1\mathsf{U}\varphi_2)$ holds if and only if φ_2 holds:

$$\begin{aligned}
&[G] \models^? \mathsf{A}(\varphi_1\mathsf{U}\varphi_2) \equiv \\
&\texttt{ifThenElse}(\tau(\varphi_2), X_1 \Rightarrow [G] \models \mathsf{A}(\varphi_1\mathsf{U}\varphi_2), X_2 \Rightarrow [G] \not\models \mathsf{A}(\varphi_1\mathsf{U}\varphi_2))@G \\
&\quad \text{if } [G] \text{ is } \Mapsto\text{-irreducible}
\end{aligned}$$

We reassemble partial information on the satisfaction of a formula $\mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)$ inside a multiverse:

$$\begin{aligned}
\{V \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\ W \models^? \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\} &\equiv \{V \models^? \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\ W \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\} \\
&\equiv \{V\ W\} \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2) \\
\{V \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\ W \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\} &\equiv \{V\ W\} \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2) \\
\{V \models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\ W \models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\} &\equiv \{V\ W\} \models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2) \\
\{V \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\ W \models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\} &\equiv \{V \models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\ W \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)\} \\
&\equiv \{V\ W\} \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)
\end{aligned}$$

$\boxed{\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)}$

Again the reasoning over the satisfaction of the formula $\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)$ on a world is similar to the one for the counterpart universally quantified formula.

$$\begin{aligned}
&[G] \models^? \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) \Rrightarrow \\
&\texttt{ifThenElse}(\tau(\varphi_2), \\
&\qquad X_1 \Rightarrow W \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2), \\
&\qquad X_2 \Rightarrow (\texttt{ifThenElse}(\tau(\varphi_1), \\
&\qquad\qquad X_3 \Rightarrow W \models^? \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2), \\
&\qquad\qquad X_4 \Rightarrow W \not\models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2))@G))@G \\
&\quad \text{if } [G] \Mapsto W
\end{aligned} \tag{7.8}$$

The result on the satisfaction of $\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)$ on a $\Mapsto$-irreducible world is similar to case of $\mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)$.

The satisfaction of a formula $\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)$ on a multiverse is deduced from the satisfaction information on the component worlds as follows:

$$\begin{aligned}
\{V \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\ W \models^? \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\} &\equiv \{V \models^? \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\ W \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\} \\
&\equiv \{V\ W\} \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) \\
\{V \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\ W \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\} &\equiv \{V \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\ W \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\} \\
&\equiv \{V\ W\} \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) \\
\{V \not\models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\ W \not\models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\} &\equiv \{V\ W\} \not\models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2) \\
\{V \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\ W \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)\} &\equiv \{V\ W\} \models \mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)
\end{aligned}$$

$\boxed{\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)}$

The formula $\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)$ 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 $\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)$ 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 $\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)$ holds.

$$
\begin{aligned}
&[G] \models^{?} \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2) \Rrightarrow \\
&\texttt{ifThenElse}(\tau(\varphi_2), \\
&\qquad X_1 \Rightarrow (\texttt{ifThenElse}(\tau(\varphi_1), \\
&\qquad\qquad X_2 \Rightarrow W \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2), \\
&\qquad\qquad X_3 \Rightarrow W \models^{?} \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2))@G), \\
&\qquad X_4 \Rightarrow \texttt{ifThenElse}(\tau(\varphi_1), \\
&\qquad\qquad X_5 \Rightarrow W \models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2), \\
&\qquad\qquad X_6 \Rightarrow W \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2))@G))@G \\
&\quad \text{if } [G] \Rrightarrow W
\end{aligned}
\tag{7.9}
$$

If the current world $[G]$ is irreducible with respect to $\Rrightarrow$, then the formula $\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)$ holds if and only if φ_2 holds and φ_1 does not hold:

$$
\begin{aligned}
&[G] \models^{?} \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2) \equiv \\
&\texttt{ifThenElse}(\tau(\neg\varphi_1 \wedge \varphi_2), X_1 \Rightarrow [G] \models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2), X_2 \Rightarrow [G] \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2))@G \\
&\quad \text{if } [G] \text{ is } \Rrightarrow\text{-irreducible}
\end{aligned}
$$

The information on the satisfaction of a formula $\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)$ inside a multiverse is regrouped as follows:

$$
\begin{aligned}
\{V \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\ W \models^{?} \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\} &\equiv \{V \models^{?} \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\ W \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\} \\
&\equiv \{V\ W\} \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2) \\
\{V \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\ W \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\} &\equiv \{V\ W\} \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2) \\
\{V \models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\ W \models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\} &\equiv \{V\ W\} \models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2) \\
\{V \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\ W \models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\} &\equiv \{V \models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\ W \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)\} \\
&\equiv \{V\ W\} \not\models \mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)
\end{aligned}
$$

$\boxed{\mathsf{E}(\varphi_1 \mathsf{R} \varphi_2)}$

The reduction is defined similarly to the one for $\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)$ where we replace the formula $\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)$ by $\mathsf{E}(\varphi_1 \mathsf{R} \varphi_2)$.

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}-calculus 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.

Definition 74 (Model)**.** *A* model *over* $\nabla^{\mathcal{X}}$ *for the* ρ_{pg}*-calculus 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}*-calculus 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.,* $L_{\mathcal{Q}}(W) = \{\mathsf{A}\varphi \mid \varphi \in \mathcal{FS}(\nabla^{\mathcal{X}})$ *such that* $W \models \mathsf{A}\varphi\} \cup \{\mathsf{E}\varphi \mid \varphi \in \mathcal{FS}(\nabla^{\mathcal{X}})$ *such that* $W \models \mathsf{E}\varphi\}$.

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 $\mathsf{AG}\varphi$, $\mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)$, or $\mathsf{A}(\varphi_1 \mathsf{R} \varphi_2)$, 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 $\mathsf{EG}\varphi$, $\mathsf{E}(\varphi_1 \mathsf{U} \varphi_2)$, or $\mathsf{E}(\varphi_1 \mathsf{R} \varphi_2)$, 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 $\mathsf{AF}\varphi$ or $\mathsf{EF}\varphi$, we have $\mathcal{K}_{pg}, W \models \phi$ if there exists W' such that $W \models^{?} \phi \Rrightarrow^{*} W' \models \phi$;
- for ϕ either $\mathsf{AX}\varphi$ or $\mathsf{EX}\varphi$, we have $\mathcal{K}_{pg}, W \models \phi$ if there exist W' such that $W \models^{?} \phi \Rrightarrow W' \models \phi$.

7.3.3 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}-calculus, a failure can be tested by the in-satisfaction 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}-calculus 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 (7.3) concerning the reduction with the formula $\mathsf{AG}\varphi$. We replace the rule $X_2 \Rightarrow W \not\models \mathsf{AG}\varphi$ by $X_2 \Rightarrow (X' \Rightarrow (X' \not\models \mathsf{AG}\varphi)@W) \not\models \mathsf{AG}\varphi$. 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 $\mathsf{AG}\neg\, Virus$ and the associated repairing strategy could be:

   $$X \not\models \mathsf{AG}\neg\, Virus \Rightarrow X\, Antibiotic \models^? \mathsf{AG}\neg\, Virus$$

   which introduces an antibiotic in the system, or, something simpler:

   $$X\, Virus \not\models \mathsf{AG}\neg\, Virus \Rightarrow X \models^? \mathsf{AG}\neg\, Virus$$

   which removes the virus.

2. A formula of the type $\mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)$ 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 $V \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)$, it means that the signal protein disappeared from the system. Then a repairing strategy would be:

   $$X \not\models \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2) \Rightarrow X\ M \models^? \mathsf{A}(\varphi_1 \mathsf{U} \varphi_2)$$

   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.

7.4 Conclusions and Perspectives

In this chapter we have embedded in the ρ_{pg}-calculus 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}-calculus, 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}-calculus. We summarize them in the following.

The structural formulas for the ρ^v_{pg}-calculus 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}-calculus, 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}-calculus. A solution of this problem is to consider the possible approach mentioned in the perspectives of the Chapter 2, 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}-calculus 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}-calculus. 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.

For expressing more structural properties on the port graphs, new strategies should be defined. For instance a strategy that traverses a port graph in order to verify if every node satisfies a particular formula; such a strategy will permit us to define universally quantified structural formulas.

Conclusions and Perspectives

Conclusions

In this dissertation, we defined a biochemical calculus that fits the needs of modeling biological interactions and autonomous systems, explored its expressivity and studied some of its properties. Our approach departs from the classical chemical model, its high-level formalization, and the use of strategic graph rewriting for modeling biochemical networks.

The basic entities of the abstract biochemical calculus (or $\rho_{\langle\Sigma\rangle}$-calculus) are molecules over a structure Σ which endows them with connectivity capabilities, abstraction over such structured molecules and abstraction application. The major difference from the chemical model is provided by the structure of molecules and the possibility of defining abstraction over complex patterns. From the computational point of view, we designed this calculus to model concurrent interactions between abstractions and molecules. With the help of strategic rewriting, some interactions may be designed with more control. Thanks to the expressivity of the biochemical aspect of the $\rho_{\langle\Sigma\rangle}$-calculus, we encoded strategies as objects of the calculus.

We identified a graph structure and formalized it as port graphs with a corresponding matching algorithm and rewriting relation. In parallel, we provided an encoding of port graphs and the port graphs rewriting as algebraic terms over a first-order signature and term rewriting. We then used the structure of port graphs to instantiate the abstract biochemical calculus. The result is a biochemical calculus based on strategic port graph rewriting. In the context of this calculus we showed the universality of the port graph structure for expressing abstractions and metalevel operations like matching and replacement used in the application of an abstraction to a molecule.

We validated our biochemical model based on strategic port graph rewriting in practice on the example of an autonomous system and on a fragment of a biochemical system. The application to autonomous computing is inherited from the higher-order chemical model and the biological application represents one of the motivating aspects of this thesis. We saw that the port graph structure is well-suited for modeling both types of application and, in addition, the strategic rewriting allows us to model self-management properties of the autonomic systems and to generate biochemical networks.

In the end, we proved once more the extensibility and expressivity of the calculus by embedding a runtime verification technique. The main motivation raised in the context of modeling self-healing properties in autonomic computing. We were able to put into practice runtime verification in the calculus thanks to the capabilities of the calculus for encoding strategies and handling failure.

Perspectives

A first improvement of the calculus could be the definition of new strategies for controlling the interactions between molecules. We have briefly presented some possible parallel strategies in Chapter 2, but further study needs to be done on encoding them as well as finding other strategies. The suitable balance between controlled and uncontrolled interactions is an interesting question to address for a given application and biological systems may provide us with valuable intuitions.

On the long term, the work on the calculus introduced in this thesis is concerned with the problem of adequacy of the strategies for describing biochemical systems and self-management properties of autonomous systems in general. We think that the use of strategies can help biologists and computer scientists to better understand the behavior of such systems, their self-management properties for instance, to deduce new organization and behavioral principles, as well as to define new bio-inspired computational models. For this purpose, we should try to model as many relevant examples of biological systems as possible and see if their behavior can be described using existing strategies or if new strategies should be defined.

The calculus we proposed in this thesis is a qualitative formalism. An extension of the calculus could be to endow it with a quantitative semantics. In particular for the ρ_{pg}-calculus it should be easy to define a stochastic rewriting relation on port graphs, since it was already done for bigraphs [KMT08]. Then we should also integrate stochastic rewriting strategies. In the context of modeling biochemical networks, the stochastic dimension of the model will allow us the derivation of more realistic quantitative models as ODE-based models or stoichiometric models.

Another direction worth pursuing involves the similarity of port graphs to bigraphs. On one hand we could enrich the port graph rewriting formalism by adapting results, properties and applications on bigraphs. On the other hand, we could define a biochemical calculus based on the bigraphical reactive systems, study its properties and expressivity and see if we could express the application of a bigraphical reaction rule via bigraphical transformations as we can do with port graphs. Also concerning alternative structures for instantiating the abstract biochemical calculus, we should investigate which of the existing examples inspired from the literature on high-level replacement systems could provide the calculus with interesting applications.

It would be very useful to have an implementation of the biochemical calculus. In a first stage we should consider the implementation of the chemical model [Rad07]. In a second stage we should provide an implementation for port graph rewriting. In this thesis we have defined an operational semantics for port graph rewriting based on term rewriting, and we implemented it in TOM for molecular graphs; however, the current implementation is not very efficient for large port graphs. We already have some ideas for improving its efficiency, either by using TOM's library on term-graphs or by considering more efficient matching algorithms for general graphs or for bigraphs. Our graph-based formalism is visual, so hopefully easy to understand. We should take advantage of this aspect by providing as well a graphical interface to ease the visualization of port graphs and their transformation.

A Internal Evaluation Rules for the Application in the ρ_{pg}-Calculus

A.1 Matching

Start Matching

The rule ι_1 in Figure A.1 starts the matching process. When an abstraction is applied on an object port graph, a matching node $\lll$ connects the handlers of all nodes from the left-hand side of the abstraction and the handlers of all nodes of the subject port graph. This rules is applied only one time, at the beginning of the matching.

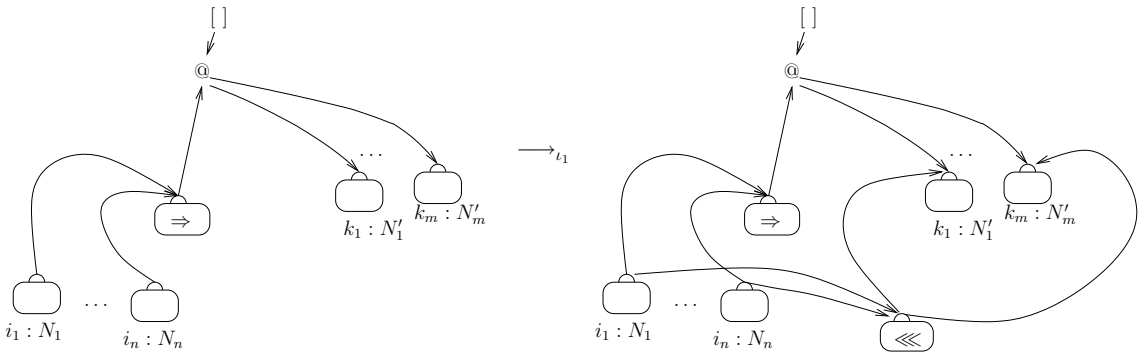


Figure A.1: Internal evaluation rule ι_1: Start matching

Match Sets of Nodes

The matching node $\lll$ between two sets of node handlers reduces to a matching node $\ll$ between two node handlers and a matching node $\lll$ between the remaining sets of nodes. For every possible pair of node handlers, one from the left-hand side and the other from the right-hand side, a new world is created, hence it results a multiset of worlds.

In Figure A.2 we illustrate graphically the rule, while above we describe it schematically using a term syntax. This rule corresponds to the decomposition rule ($\mathbf{D_1}$) from the matching algorithm on port graphs given in Chapter 3.

$$[(\langle i_1 : N_1 \parallel h\rangle, \ldots, \langle i_n : N_n \parallel h\rangle) \lll (\langle k_1 : N'_1 \parallel h\rangle, \ldots, \langle k_m : N'_m \parallel h\rangle)]$$

$$
\begin{aligned}
&\longrightarrow_{\iota_2}\\
&[\{[\langle i_1 : N_1 \parallel h\rangle \ll \langle k_1 : N_1' \parallel h\rangle,\\
&\quad (\langle i_2 : N_2 \parallel h\rangle, \ldots, \langle i_n : N_n \parallel h\rangle) \lll (\langle k_2 : N_2' \parallel h\rangle, \ldots, \langle k_m : N_m' \parallel h\rangle)]\\
&\ldots\\
&[\langle i_1 : N_1 \parallel h\rangle \ll \langle k_m : N_m' \parallel h\rangle,\\
&\quad (\langle i_2 : N_2 \parallel h\rangle, \ldots, \langle i_n : N_n \parallel h\rangle) \lll (\langle k_1 : N_1' \parallel h\rangle, \ldots, \langle k_{m-1} : N_{m-1}' \parallel h\rangle)]\\
&\ldots\\
&[\langle i_n : N_n \parallel h\rangle \ll \langle k_1 : N_1' \parallel h\rangle,\\
&\quad (\langle i_1 : N_1 \parallel h\rangle, \ldots, \langle i_{n-1} : N_{n-1} \parallel h\rangle) \lll (\langle k_2 : N_2' \parallel h\rangle, \ldots, \langle k_m : N_m' \parallel h\rangle)]\\
&\ldots\\
&[\langle i_n : N_n \parallel h\rangle \ll \langle k_m : N_m' \parallel h\rangle,\\
&\quad (\langle i_1 : N_1 \parallel h\rangle, \ldots, \langle i_{n-1} : N_{n-1} \parallel h\rangle) \lll (\langle k_1 : N_1' \parallel h\rangle, \ldots, \langle k_{m-1} : N_{m-1}' \parallel h\rangle)]\}]
\end{aligned}
$$

If the pattern set of nodes is empty, then the subject set of nodes belongs to the context graph; then the matching node $\lll$ is simply deleted.

$$\epsilon \lll (\langle k_1 : N_1' \parallel h\rangle, \ldots, \langle k_m : N_m' \parallel h\rangle) \longrightarrow_{\iota_3} (\langle k_1 : N_1' \parallel h\rangle, \ldots, \langle k_m : N_m' \parallel h\rangle)$$

Whereas if the subject set of nodes is empty, then the matching fails.

$$(\langle i_1 : N_1 \parallel h\rangle, \ldots, \langle i_n : N_n \parallel h\rangle) \lll \epsilon,\ \sigma \longrightarrow_{\iota_4} (\langle i_1 : N_1 \parallel h\rangle, \ldots, \langle i_n : N_n \parallel h\rangle) \lll \epsilon,\ \mathsf{stk}$$

These two rules ι_3 and ι_4, illustrated in Figure A.3 and Figure A.4, have a lower priority than ι_2, and they correspond to the matching rules ($\mathbf{D_3}$) and ($\mathbf{D_2}$) respectively.

In Figure A.5 we illustrate the evaluation rule which replaces a set matching node $\lll$ between two singletons by a matching node $\ll$. This rule has a lower priority in application than the rule ι_2.

$$\langle i : N \parallel h\rangle \lll \langle k : N' \parallel h\rangle \longrightarrow_{\iota_5} \langle i : N \parallel h\rangle \ll \langle k : N' \parallel h\rangle$$

Match Nodes

We investigate now the ways a matching node connecting the handlers of two nodes (which we denote by the pattern node and the subject node) is transformed.

We consider first the case when the pattern node has a variable name. In the rule ι_6 from Figure A.6 both the pattern node and the subject node have ports. Then the substitution is extended with the assignment of the node identifier and name of the subject to the variable identifier and name of the pattern. The matching node is transformed into

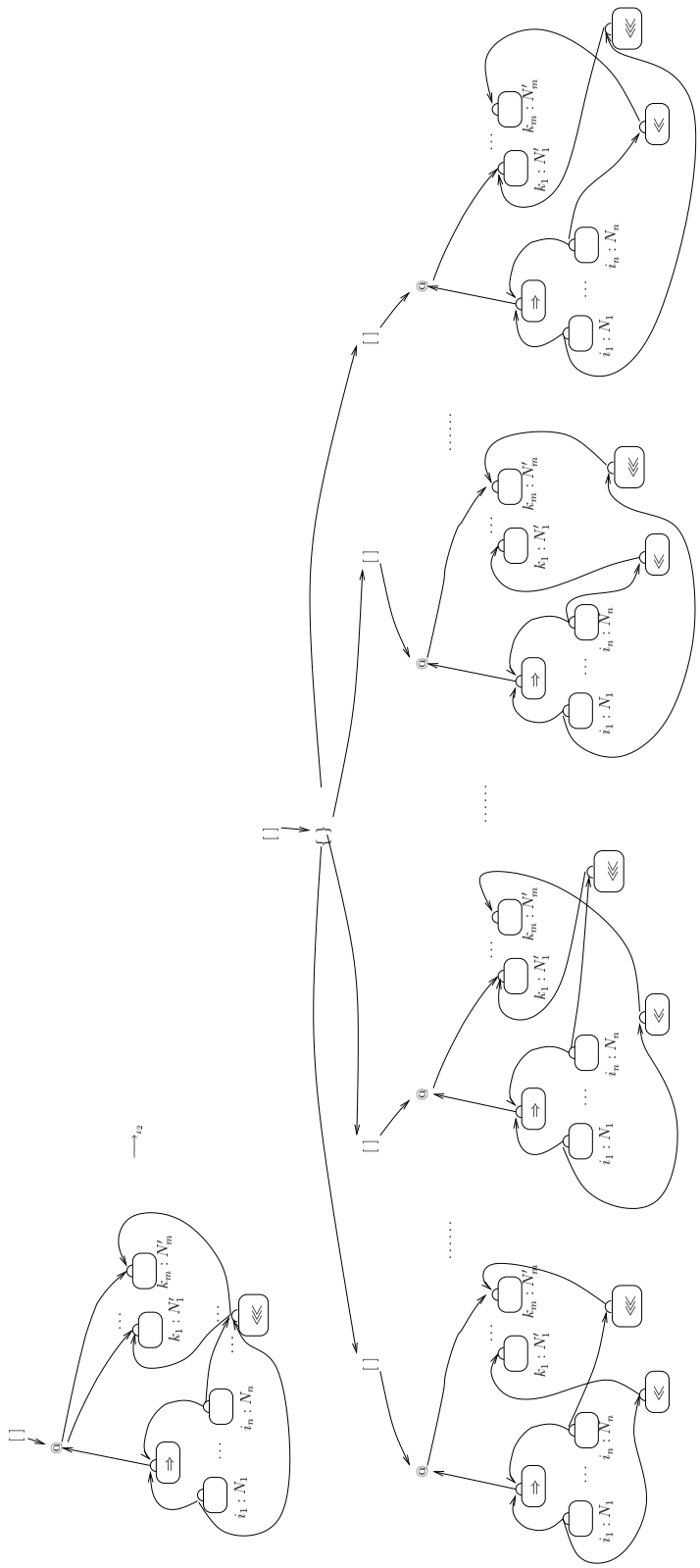


Figure A.2: Internal evaluation rule ι_2: Decompose a matching between two sets of nodes

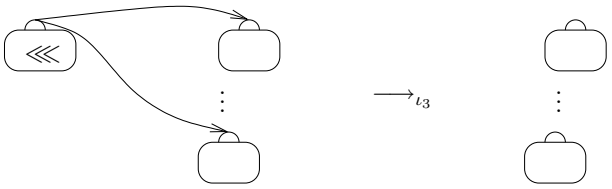


Figure A.3: Internal evaluation rule ι_3: Empty set of nodes as pattern

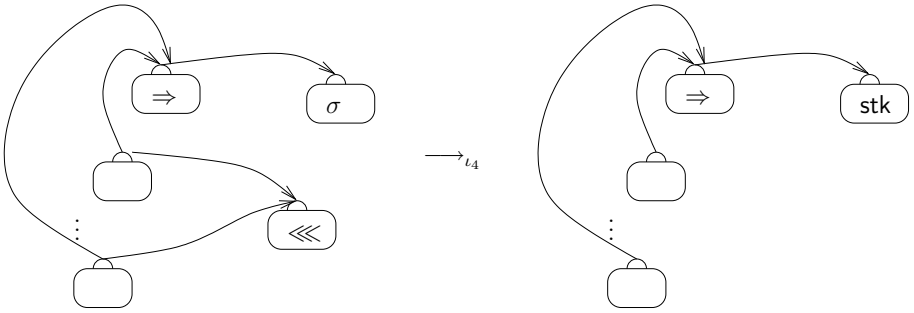


Figure A.4: Internal evaluation rule ι_4: Empty set of nodes as subject

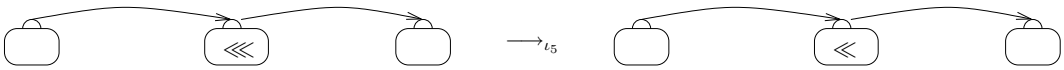


Figure A.5: Internal evaluation rule ι_5: Match singletons of nodes

a matching node connecting the sets of ports of the two nodes. In addition, the handler of the instantiated pattern node is no longer connected to the handler of the arrow node, but to the black hole of the arrow node. We do not replace i by k for the matched node from the pattern because we want to have well-formed nodes, where all node identifiers are pairwise distinct. This rule corresponds to the collective behavior of the matching rules ($\mathbf{D_6}$), ($\mathbf{S_2}$), and ($\mathbf{S_1}$) without pursuing the matching on the neighbors.

$$
\begin{array}{l}
\langle i : X \parallel h \rangle \ll \langle k : A \parallel h \rangle, \\
(\langle i : X \parallel h, p_1, \ldots, p_n \rangle, \langle j :\Rightarrow \parallel h, bh, p'_1, \ldots, p'_n \rangle, \langle k : A \parallel h, r_1, \ldots, r_m \rangle)(\!| \\
\ i \doteq j^\frown(h,h), (p_1, p'_1), \ldots, (p_n, p'_n) |\!) \\
\sigma \\
\ \longrightarrow_{\iota_6} \\
\langle i : A \parallel p_1, \ldots, p_n \rangle \ll \langle k : A \parallel r_1, \ldots, r_m \rangle, \\
(\langle i : A \parallel h, p_1, \ldots, p_n \rangle, \langle j :\Rightarrow \parallel h, bh, p'_1, \ldots, p'_n \rangle, \langle k : A \parallel h, r_1, \ldots, r_m \rangle)(\!| \\
\ i \doteq j^\frown(h,bh), (p_1, p'_1), \ldots, (p_n, p'_n) |\!) \\
\sigma \cup \{i \mapsto k, X \mapsto A\}
\end{array}
$$

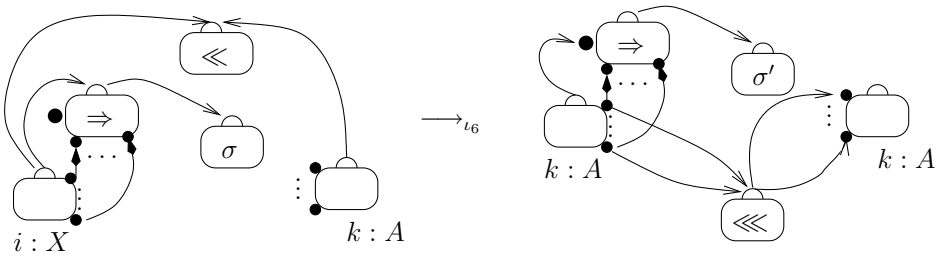


Figure A.6: Internal evaluation rule ι_6: Match variable node

If both the pattern and the subject nodes do not have ports, then, obviously, no matching node for sets of ports is introduced. The evaluation rule is given in Figure A.7 and it corresponds to the collective behavior of the matching rules ($\mathbf{S_2}$) and ($\mathbf{S_1}$) without pursuing the matching on the neighbors.

$$
\begin{array}{l}
\langle i : X \parallel h\rangle \ll \langle k : A \parallel h\rangle, \\
(\langle i : X \parallel h\rangle, \langle j : \Rightarrow \parallel h, bh\rangle, \langle k : A \parallel h\rangle)(\!|i \doteq j^\frown(h, h)|\!) \\
\sigma \\
\longrightarrow_{\iota_7} \\
(\langle i : A \parallel h\rangle, \langle j : \Rightarrow \parallel h, bh\rangle, \langle k : A \parallel h\rangle)(\!|i \doteq j^\frown(h, bh)|\!) \\
\sigma \cup \{i \mapsto k, X \mapsto A\}
\end{array}
$$

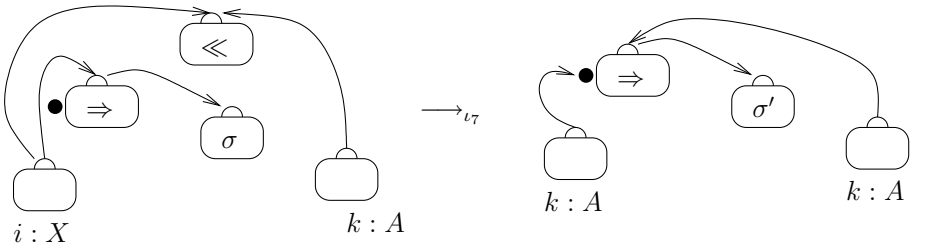


Figure A.7: Internal evaluation rule ι_7: Match variable node without ports

In Figure A.8 the pattern (subject) node does not have ports whereas the subject (pattern) node does. This situation yields a matching failure and we represent it by replacing the substitution node with a stk node. If the subject node does not have port, then the rule is he following:

$$
\begin{array}{l}
\langle i : X \parallel h\rangle \ll \langle k : A \parallel h\rangle, \\
(\langle i : X \parallel h, p_1, \ldots, p_n\rangle, \langle j : \Rightarrow \parallel h, bh, p'_1, \ldots, p'_n\rangle, \langle k : A \parallel h\rangle)(\!|
\end{array}
$$

$$
\begin{array}{l}
i \doteq j^\frown(h,h),(p_1,p_1'),\ldots,(p_n,p_n')\rangle\!\rangle \\
\sigma \\
\longrightarrow_{\iota_8} \\
\langle i : X \parallel h\rangle \ll \langle k : A \parallel h\rangle, \\
(\langle i : X \parallel h, p_1, \ldots, p_n\rangle, \langle j : \Rightarrow \parallel h, bh, p_1', \ldots, p_n'\rangle, \langle k : A \parallel h\rangle)\langle\!\langle \\
i \doteq j^\frown(h,h),(p_1,p_1'),\ldots,(p_n,p_n')\rangle\!\rangle \\
\mathsf{stk}
\end{array}
$$

and for the other case, the rule is the following:

$$
\begin{array}{l}
\langle i : X \parallel h\rangle \ll \langle k : A \parallel h\rangle, \\
(\langle i : X \parallel h\rangle, \langle j : \Rightarrow \parallel h\rangle, \langle k : A \parallel h, r_1, \ldots, r_m\rangle)\langle\!\langle i \doteq j^\frown(h,h)\rangle\!\rangle \\
\sigma \\
\longrightarrow_{\iota_9} \\
\langle i : X \parallel h\rangle \ll \langle k : A \parallel h\rangle, \\
(\langle i : X \parallel h\rangle, \langle j : \Rightarrow \parallel h\rangle, \langle k : A \parallel h, r_1, \ldots, r_m\rangle)\langle\!\langle i \doteq j^\frown(h,h)\rangle\!\rangle \\
\mathsf{stk}
\end{array}
$$

The rule ι_8 corresponds to the matching rules ($\mathbf{D_2}$) and ($\mathbf{D_6}$), while the rule ι_9 corresponds to the matching rules ($\mathbf{D_5}$) and ($\mathbf{D_6}$).

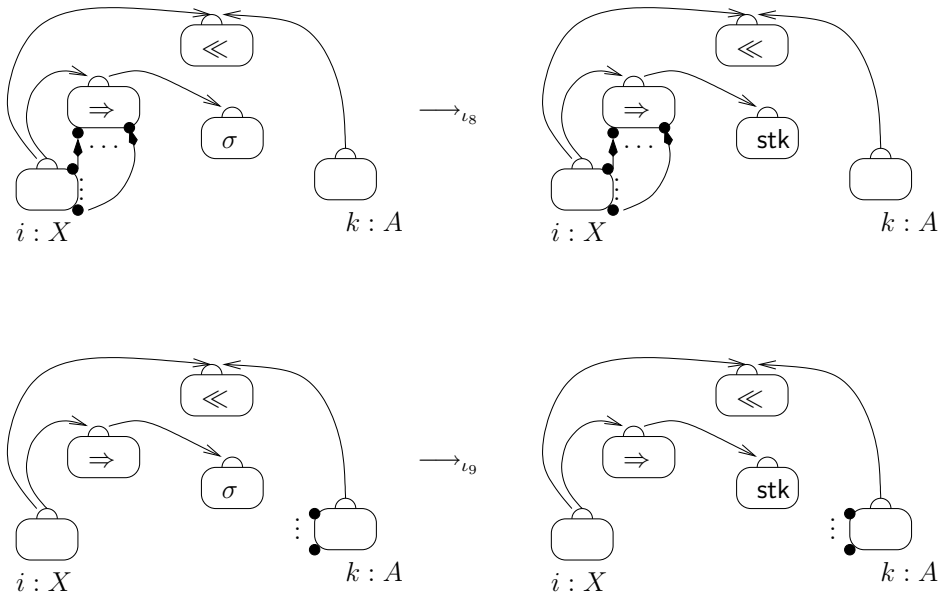


Figure A.8: Internal evaluation rules ι_8 and ι_9: Match variable node when exactly one of the pattern and the subject nodes does not have ports besides the handler but the other does

If the pattern node has a constant name equal to that of the subject node, the transformation is similar with the one for variable node names except that the substitution is extended only for assigning the identified of the subject node to the identifier of the pattern node. But if the constant name of the pattern node if different from that of the subject node, the matching fails and the stk node replaces the substitution node.

The rules ι_{10} and ι_{11} in Figure A.9 consider the case of constant node names with non-empty sets of ports. For equal node names the rule is:

$$
\begin{array}{l}
\langle i : A \parallel h\rangle \ll \langle k : A \parallel h\rangle, \\
(\langle i : A \parallel h, p_1, \ldots, p_n\rangle, \langle j : \Rightarrow \parallel h, bh, p'_1, \ldots, p'_n\rangle, \langle k : A \parallel h, r_1, \ldots, r_m\rangle)(\!| \\
\ i \doteq j^\frown(h,h), (p_1, p'_1), \ldots, (p_n, p'_n) |\!) \\
\sigma \\
\longrightarrow_{\iota_{10}} \\
\langle i : A \parallel p_1, \ldots, p_n\rangle \ll \langle k : A \parallel r_1, \ldots, r_m\rangle, \\
(\langle i : A \parallel h, p_1, \ldots, p_n\rangle, \langle j : \Rightarrow \parallel h, bh, p'_1, \ldots, p'_n\rangle, \langle k : A \parallel h, r_1, \ldots, r_m\rangle)(\!| \\
\ i \doteq j^\frown(h,bh), (p_1, p'_1), \ldots, (p_n, p'_n) |\!) \\
\sigma \cup \{i \mapsto k\}
\end{array}
$$

whereas, for distinct node names, a failure node is produced:

$$
\begin{array}{l}
\langle i : A \parallel h\rangle \ll \langle k : B \parallel h\rangle, \\
(\langle i : A \parallel h, p_1, \ldots, p_n\rangle, \langle j : \Rightarrow \parallel h, bh, p'_1, \ldots, p'_n\rangle, \langle k : B \parallel h, r_1, \ldots, r_m\rangle)(\!| \\
\ i \doteq j^\frown(h,h), (p_1, p'_1), \ldots, (p_n, p'_n) |\!) \\
\sigma \\
\longrightarrow_{\iota_{11}} \\
(\langle i : A \parallel h, p_1, \ldots, p_n\rangle, \langle j : \Rightarrow \parallel h, bh, p'_1, \ldots, p'_n\rangle, \langle k : B \parallel h, r_1, \ldots, r_m\rangle)(\!| \\
\ i \doteq j^\frown(h,h), (p_1, p'_1), \ldots, (p_n, p'_n) |\!) \\
\mathsf{stk} \\
\text{if } A \neq B
\end{array}
$$

These rules corresponds to the collective behavior of the matching rules ($\mathbf{D_6}$), ($\mathbf{D_9}$), and ($\mathbf{S_1}$) (only for ι_{10} and ι_{12}).

The evaluation rules in Figure A.10 apply when matching nodes without ports:

$$
\begin{array}{l}
\langle i : A \parallel h\rangle \ll \langle k : A \parallel h\rangle, \\
(\langle i : A \parallel h\rangle, \langle j :\Rightarrow \parallel h, bh\rangle, \langle k : A \parallel h\rangle)(\!| i \doteq j^\frown(h,h) |\!) \\
\sigma \\
\longrightarrow_{\iota_{12}} \\
(\langle i : A \parallel h\rangle, \langle j :\Rightarrow \parallel h, bh\rangle, \langle k : A \parallel h\rangle)(\!| i \doteq j^\frown(h,bh) |\!) \\
\sigma \cup \{i \mapsto k\}
\end{array}
$$

and

$$
\begin{array}{l}
\langle i : A \parallel h\rangle \ll \langle k : B \parallel h\rangle, \\
(\langle i : A \parallel h\rangle, \langle j :\Rightarrow \parallel h, bh\rangle, \langle k : B \parallel h\rangle)(\!| i \doteq j^\frown(h,h) |\!) \\
\sigma
\end{array}
$$

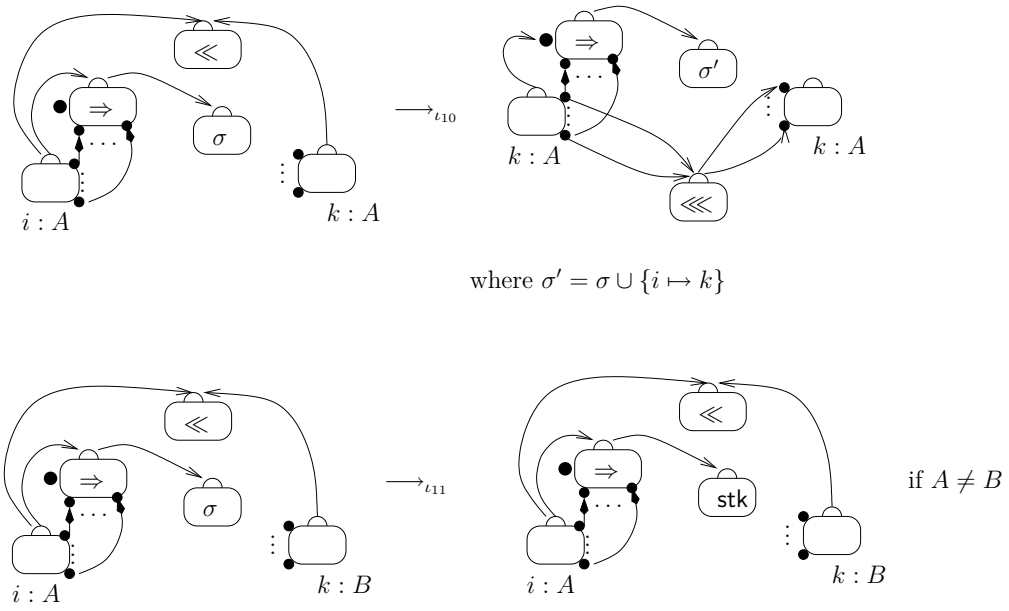


Figure A.9: Internal evaluation rules ι_{10} and ι_{11}: Match constant node with ports

$$
\begin{array}{l}
\longrightarrow_{\iota_{13}} \\
(\langle i : A \parallel h \rangle, \langle j :\Rightarrow \parallel h, bh \rangle, \langle k : B \parallel h \rangle)(\!|i \doteq j^\frown(h,h)|\!) \\
\mathsf{stk} \\
\text{if } A \neq B
\end{array}
$$

When exactly one of the pattern node and the subject node does not have ports, and both nodes have constant names, the evaluation rules ι_{14} and ι_{15} from Figure A.11 are applicable and they are represented using terms as follows:

$$
\begin{array}{l}
\langle i : A \parallel h \rangle \ll \langle k : B \parallel h \rangle, \\
(\langle i : A \parallel h, p_1, \ldots, p_n \rangle, \langle j :\Rightarrow \parallel h, bh, p'_1, \ldots, p'_n \rangle, \langle k : B \parallel h \rangle)(\!| \\
\ i \doteq j^\frown(h,h), (p_1, p'_1), \ldots, (p_n, p'_n)|\!) \\
\sigma \\
\longrightarrow_{\iota_{14}} \\
\langle i : A \parallel h \rangle \ll \langle k : B \parallel h \rangle, \\
(\langle i : A \parallel h, p_1, \ldots, p_n \rangle, \langle j :\Rightarrow \parallel h, bh, p'_1, \ldots, p'_n \rangle, \langle k : B \parallel h \rangle)(\!| \\
\ i \doteq j^\frown(h,h), (p_1, p'_1), \ldots, (p_n, p'_n)|\!) \\
\mathsf{stk}
\end{array}
$$

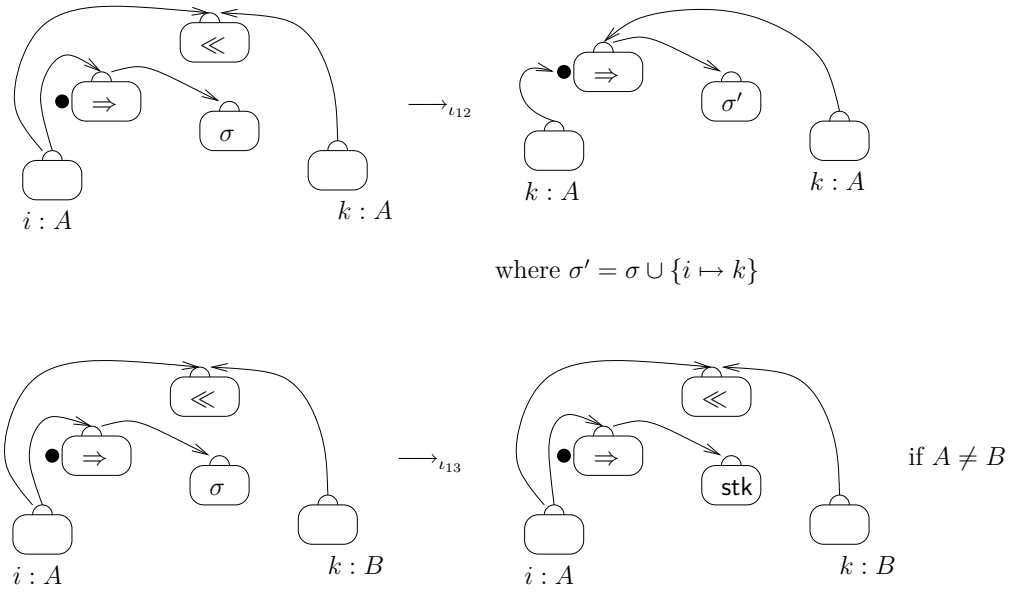


Figure A.10: Internal evaluation rules ι_{12} and ι_{13}: Match constant node without ports

and for the other case, the rule is the following:

$$
\begin{aligned}
&\langle i : A \parallel h\rangle \ll \langle k : B \parallel h\rangle,\\
&(\langle i : A \parallel h\rangle, \langle j :\Rightarrow \parallel h\rangle, \langle k : B \parallel h, r_1, \ldots, r_m\rangle)(\!|i \doteq j^\frown(h,h)|\!)\\
&\sigma\\
&\longrightarrow_{\iota_{15}}\\
&\langle i : A \parallel h\rangle \ll \langle k : B \parallel h\rangle,\\
&(\langle i : A \parallel h\rangle, \langle j :\Rightarrow \parallel h\rangle, \langle k : B \parallel h, r_1, \ldots, r_m\rangle)(\!|i \doteq j^\frown(h,h)|\!)\\
&\mathsf{stk}
\end{aligned}
$$

The rule ι_{14} has the same behavior as the matching rules $(\mathbf{D_6})$ and $(\mathbf{D_2})$, while ι_{15} has the same behavior as the matching rules $(\mathbf{D_6})$ and $(\mathbf{D_5})$.

Match Sets of Ports

A matching node between two sets of ports is decomposed in smaller matching problems. For a port from the pattern set and for each port in the subject set a new simple world is created where the two ports are being matched and the rest of the sets of ports as well. The corresponding evaluation rule is given in Figure A.12 and the term-like syntax in the following:

$$[x_1, \ldots, x_n \lll a_1, \ldots, a_m]$$

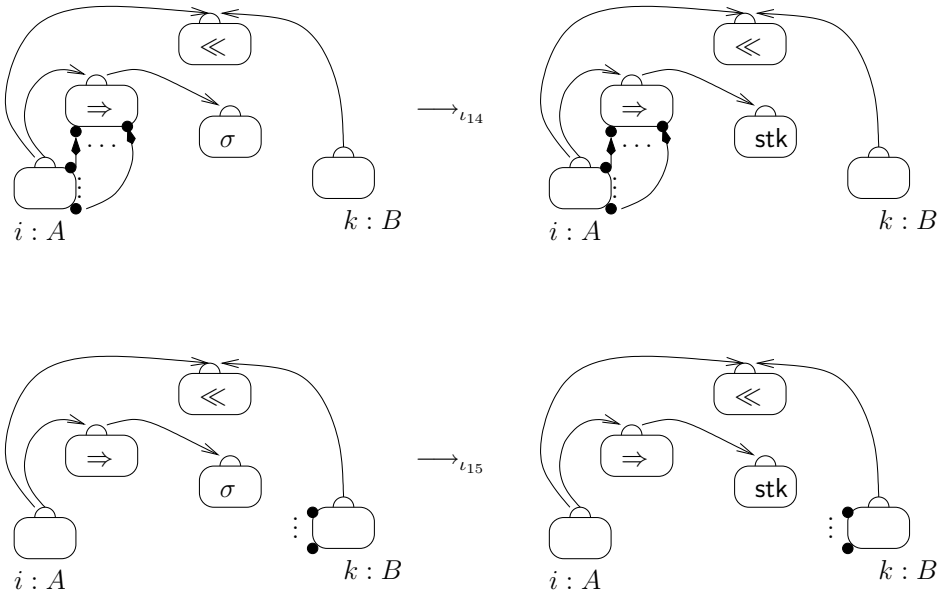


Figure A.11: Internal evaluation rules ι_{14} and ι_{15}: Match constant node when exactly one of the pattern and the subject nodes does not have ports besides the handler but the other does

$$
\begin{aligned}
&\longrightarrow_{\iota_{16}} \\
&[\{[(x_1 \ll a_1),\ (x_2, \ldots, x_n \lll a_2, \ldots, a_m)] \\
&\ldots \\
&[(x_1 \ll a_m),\ (x_2, \ldots, x_n \lll a_1, \ldots, a_{m-1})] \\
&\ldots \\
&[(x_n \ll a_1),\ (x_1, \ldots, x_{n-1} \lll a_2, \ldots, a_m)] \\
&\ldots \\
&[(x_n \ll a_m),\ (x_1, \ldots, x_{n-1} \lll a_1, \ldots, a_{m-1})]\}]
\end{aligned}
$$

If the pattern set of ports is empty, then the matching fails (Figure A.13):

$$\epsilon \lll a_1, \ldots, a_m,\ \sigma \longrightarrow_{\iota_{17}} a_1, \ldots, a_m,\ \mathsf{stk}$$

And similarly, if the subject set of ports is empty, then the matching fails (Figure A.14):

$$x_1, \ldots, x_m \lll \epsilon,\ \sigma \longrightarrow_{\iota_{18}} x_1, \ldots, x_m,\ \mathsf{stk}$$

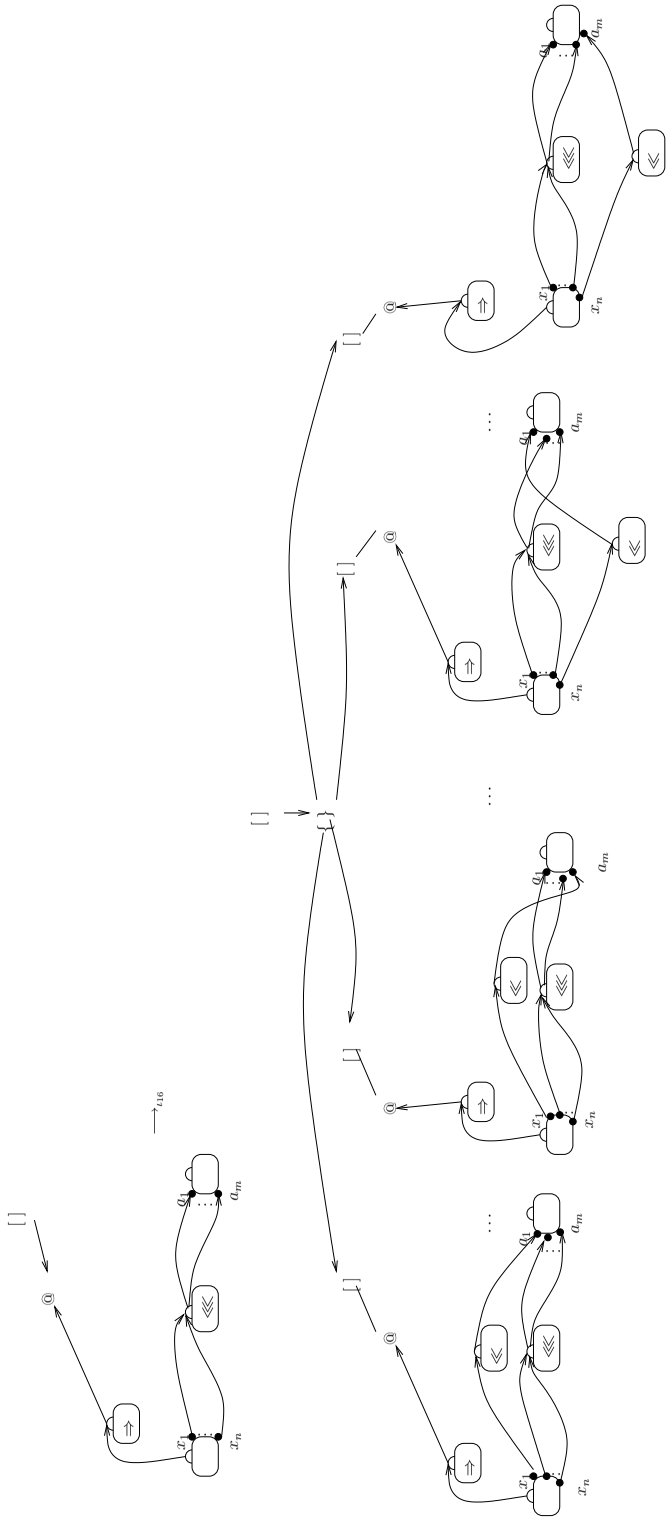


Figure A.12: Internal evaluation rule ι_{16}: Match sets of ports

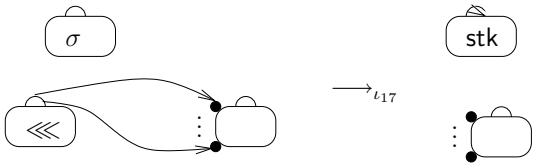


Figure A.13: Internal evaluation rule ι_{17}: Empty set of ports as pattern

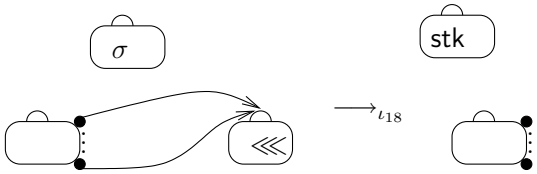


Figure A.14: Internal evaluation rule ι_{18}: Empty set of ports as subject

We replace a matching node between two singletons with a matching node between two ports (Figure A.15):

$$\langle i : A \parallel x \rangle \lll \langle k : A \parallel a \rangle \longrightarrow_{\iota_{19}} \langle i : A \parallel x \rangle \ll \langle k : A \parallel a \rangle$$

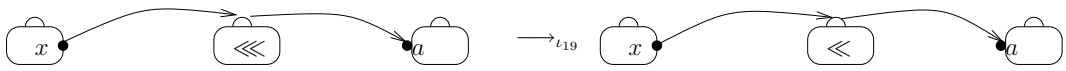


Figure A.15: Internal evaluation rule ι_{19}: Match singletons of ports

The evaluation rule ι_{16} corresponds to the matching rule ($\mathbf{D_1}$), ι_{17} to ($\mathbf{D_5}$), and ι_{18} to ($\mathbf{D_2}$). In addition, the rules ι_{17}, ι_{18}, and ι_{19} have a lower application priority than ι_{16}.

Match Ports

If the pattern is a variable port, then for a neighbor of the variable port, and for each neighbor of the subject port, a new intermediary world is created where the neighbor nodes and ports are matched (Figure A.16). This rule ι_{20} corresponds to the behavior of the matching rules ($\mathbf{S_1}$) and ($\mathbf{S_3}$) for visiting the neighbors. The term-like representation is the following:

$$\begin{aligned}
&[\langle i : A \parallel x \rangle \ll \langle k : A \parallel a \rangle \\
&\langle i : A \parallel x \rangle, \langle j : Y \parallel h, y \rangle, \langle k : A \parallel a \rangle, \langle k_1 : B_1 \parallel h, a_1 \rangle, \ldots, \langle k_n : B_n \parallel h, a_n \rangle (\!| \\
&i \doteq j\frown(x, y), \\
&k \doteq k_1\frown(a, a_1), \ldots, k_n\frown(a, a_n) |\!) \\
&\sigma]
\end{aligned}$$

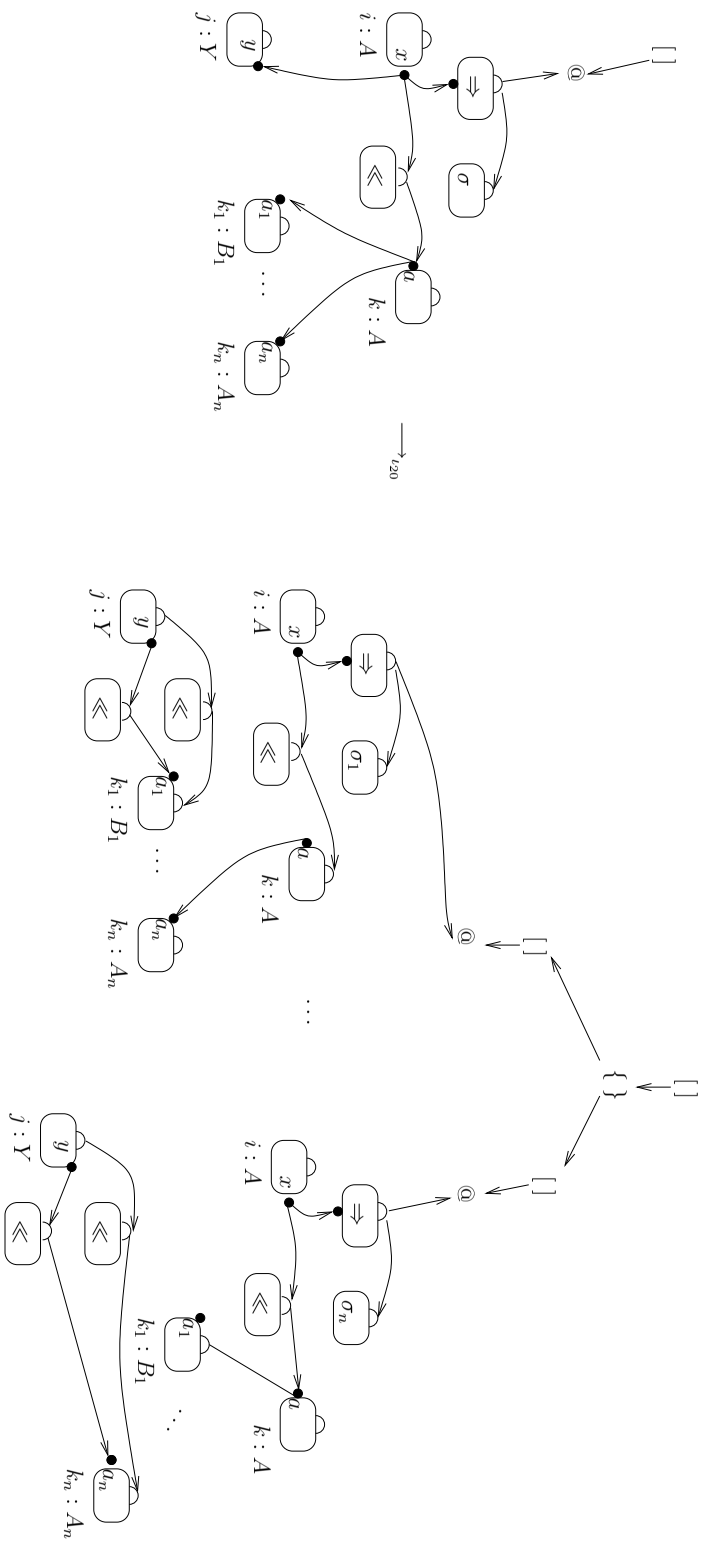
$$
\begin{array}{l}
\longrightarrow_{\iota 20} \\
[\{[\langle i : A \parallel x\rangle \ll \langle k : A \parallel a\rangle, \\
\langle j : Y \parallel h\rangle \ll \langle k_1 : B_1 \parallel h\rangle, \\
\langle j : Y \parallel y\rangle \ll \langle k_1 : B_1 \parallel a_1\rangle \\
\langle k : A \parallel x\rangle, \langle j : Y \parallel h, y\rangle, \langle k : A \parallel a\rangle, \langle k_1 : B_1 \parallel h, a_1\rangle, \ldots, \langle k_n : B_n \parallel h, a_n\rangle (\!| \\
k \doteq k_2^\frown(a, a_2), \ldots, k_n^\frown(a, a_n) |\!) \\
\sigma \cup \{x \mapsto a_1\}], \\
\ldots, \\
[\langle i : A \parallel x\rangle \ll \langle k : A \parallel a\rangle, \\
\langle j : Y \parallel h\rangle \ll \langle k_n : B_n \parallel h\rangle, \\
\langle j : Y \parallel y\rangle \ll \langle k_n : B_n \parallel a_n\rangle \\
\langle k : A \parallel x\rangle, \langle j : Y \parallel h, y\rangle, \langle k : A \parallel a\rangle, \langle k_1 : B_1 \parallel h, a_1\rangle, \ldots, \langle k_n : B_n \parallel h, a_n\rangle (\!| \\
k \doteq k_1^\frown(a, a_1), \ldots, k_{n-1}^\frown(a, a_{n-1}) |\!) \\
\sigma \cup \{x \mapsto a_n\}]\}]
\end{array}
$$

If the variable port has neighbors, but the subject port does not, the matching fails (Figure A.17). The rule ι_{21} resembles the matching rule ($\mathbf{D_2}$).

$$
\begin{array}{l}
\langle i : A \parallel x\rangle \ll \langle k : A \parallel a\rangle \\
\langle i : A \parallel x\rangle, \langle j : Y \parallel h, y\rangle, \langle k : A \parallel a\rangle (\!| i \doteq j^\frown(x, y) |\!) \\
\sigma \\
\longrightarrow_{\iota 21} \\
\langle i : A \parallel x\rangle \ll \langle k : A \parallel a\rangle \\
\langle i : A \parallel x\rangle, \langle j : Y \parallel h, y\rangle, \langle k : A \parallel a\rangle (\!| i \doteq j^\frown(x, y) |\!) \\
\mathsf{stk}
\end{array}
$$

If the pattern port and the subject port have no neighbors or all their neighbors as matched, then the connection of the pattern port to the arrow node is passed to the subject port, and the matching for the variable port is finished (Figure A.18). The rule ι_{22} has the same behavior as the matching rule $\mathbf{S_3}$ and it has a lower application priority than ι_{20} and ι_{21}.

$$
\begin{array}{l}
\langle i : A \parallel x\rangle \ll \langle k : A \parallel a\rangle \\
\langle i : A \parallel x\rangle, \langle k : A \parallel a\rangle, \langle j :\Rightarrow \parallel p\rangle (\!| i \doteq j^\frown(x, p) |\!) \\
\sigma \\
\longrightarrow_{\iota 22} \\
\langle i : A \parallel a\rangle, \langle k : A \parallel a\rangle, \langle j :\Rightarrow \parallel p\rangle (\!| k \doteq j^\frown(a, p) |\!) \\
\sigma \cup \{x \mapsto a\}
\end{array}
$$



where $\sigma_i = \sigma \cup \{x \mapsto a_i\}$, for all i, $1 \leq i \leq n$

Figure A.16: Internal evaluation rule ι_{20}: Match a variable port

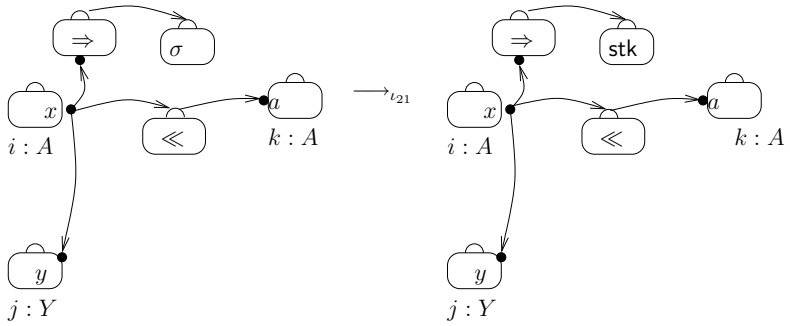
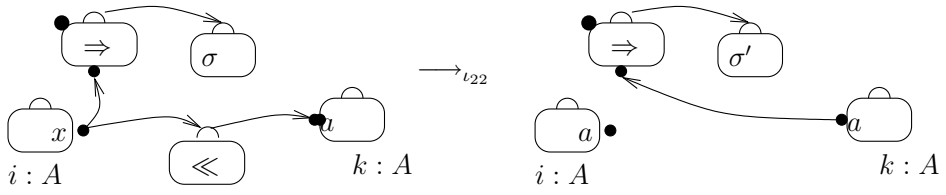


Figure A.17: Internal evaluation rule ι_{21}: Match a variable port with a neighborless port



where $\sigma' = \sigma \cup \{x \mapsto a\}$

Figure A.18: Internal evaluation rule ι_{22}: Match successfully a variable port

The following internal evaluation rule is similar to ι_{23} for a constant port name as pattern (see Figure A.19):

$$
\begin{aligned}
&[\langle i : A \parallel a\rangle \ll \langle k : A \parallel a\rangle \\
&\langle i : A \parallel a\rangle, \langle j : Y \parallel h, y\rangle, \langle k : A \parallel a\rangle, \langle k_1 : B_1 \parallel h, a_1\rangle, \ldots, \langle k_n : B_n \parallel h, a_n\rangle\langle\!| \\
&i \doteq j^\frown(a, y), \\
&k \doteq k_1^\frown(a, a_1), \ldots, k_n^\frown(a, a_n)|\!\rangle] \\
&\longrightarrow_{\iota_{23}} \\
&[\{[\langle i : A \parallel a\rangle \ll \langle k : A \parallel a\rangle, \\
&\langle j : Y \parallel h\rangle \ll \langle k_1 : B_1 \parallel h\rangle, \\
&\langle j : Y \parallel y\rangle \ll \langle k_1 : B_1 \parallel a_1\rangle \\
&\langle k : A \parallel a\rangle, \langle j : Y \parallel h, y\rangle, \langle k : A \parallel a\rangle, \langle k_1 : B_1 \parallel h, a_1\rangle, \ldots, \langle k_n : B_n \parallel h, a_n\rangle\langle\!| \\
&k \doteq k_2^\frown(a, a_2), \ldots, k_n^\frown(a, a_n)|\!\rangle], \\
&\ldots, \\
&[\langle i : A \parallel x\rangle \ll \langle k : A \parallel a\rangle,
\end{aligned}
$$

Figure A.19: Internal evaluation rule ι_{23}: Match a constant port

$\langle j : Y \parallel h \rangle \ll \langle k_n : B_n \parallel h \rangle,$
$\langle j : Y \parallel y \rangle \ll \langle k_n : B_n \parallel a_n \rangle$
$\langle k : A \parallel x \rangle, \langle j : Y \parallel h, y \rangle, \langle k : A \parallel a \rangle, \langle k_1 : B_1 \parallel h, a_1 \rangle, \ldots, \langle k_n : B_n \parallel h, a_n \rangle \langle\!|$
$k \doteq k_1^\frown(a, a_1), \ldots, k_{n-1}^\frown(a, a_{n-1}) |\!\rangle]\}]$

When matching two constant ports with different names, a failure is raised (see Figure A.20):

$\langle i : A \parallel a \rangle \ll \langle k : A \parallel b \rangle$
$\langle i : A \parallel x \rangle, \langle j : Y \parallel h, y \rangle, \langle k : A \parallel b \rangle \langle\!|\ |\!\rangle$
σ
$\longrightarrow_{\iota_{24}}$
$\langle i : A \parallel a \rangle \ll \langle k : A \parallel b \rangle$
$\langle i : A \parallel a \rangle, \langle j : Y \parallel h, y \rangle, \langle k : A \parallel b \rangle \langle\!|\ |\!\rangle$
stk
if $a \neq b$

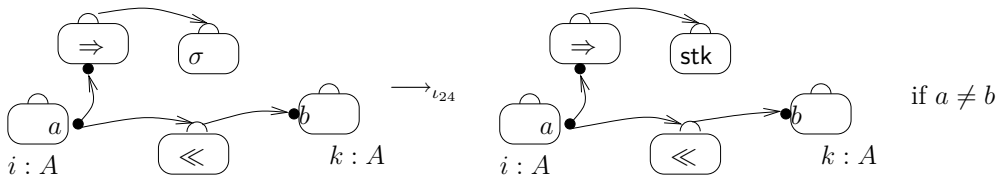


Figure A.20: Internal evaluation rule ι_{24}: Failure at matching a constant port

Similar to rule ι_{21}, the rule ι_{25} corresponds to the situation where the matching fails if the pattern port has neighbors, but the subject port does not (Figure A.21):

$\langle i : A \parallel a \rangle \ll \langle k : A \parallel a \rangle$
$\langle i : A \parallel a \rangle, \langle j : Y \parallel h, y \rangle, \langle k : A \parallel a \rangle \langle\!| i \doteq j^\frown(a, y) |\!\rangle$
σ
$\longrightarrow_{\iota_{25}}$
$\langle i : A \parallel x \rangle \ll \langle k : A \parallel a \rangle$
$\langle i : A \parallel x \rangle, \langle j : Y \parallel h, y \rangle, \langle k : A \parallel a \rangle \langle\!| i \doteq j^\frown(a, y) |\!\rangle$
stk

If both the pattern port and the subject port are constant and they do not have (unmatched) neighbors, the matching for the variable port is successfully finished (Figure A.22).

$\langle i : A \parallel a \rangle \ll \langle k : A \parallel a \rangle$
$\langle i : A \parallel a \rangle, \langle k : A \parallel a \rangle, \langle j :\Rightarrow \parallel p \rangle \langle\!| i \doteq j^\frown(a, p) |\!\rangle$
$\longrightarrow_{\iota_{26}}$
$\langle i : A \parallel a \rangle, \langle k : A \parallel a \rangle, \langle j :\Rightarrow \parallel p \rangle \langle\!| k \doteq j^\frown(a, p) |\!\rangle$

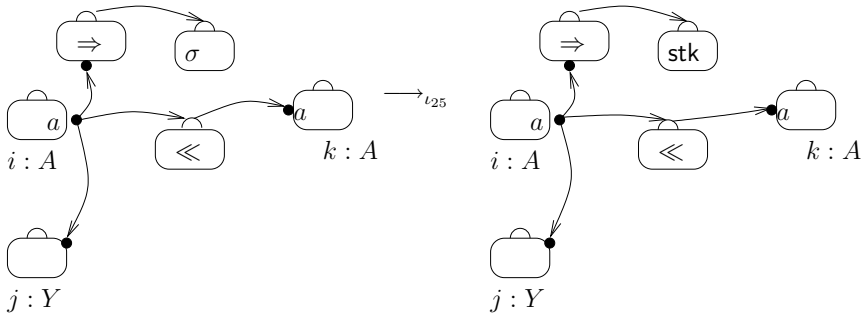


Figure A.21: Internal evaluation rule ι_{25}: Failure at matching a constant neighborless port

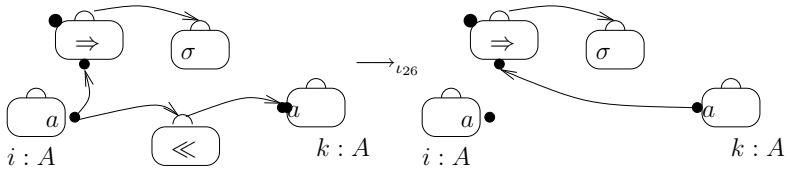


Figure A.22: Internal evaluation rule ι_{26}: Successfully matching a constant port without ports

Handling Failure

When a node stk is connected to the main arrow node of the abstractions, as a witness of a matching failure, all edges and nodes in the simple world are deleted: first the matching nodes with the incident edges, then the edges between nodes in the object port graphs of the abstraction, and, in the end, the nodes of the left- and right-hand sides and the abstraction and of the object port graph. Such rules have the highest priority over all matching rules and they correspond to the matching rule ($\mathbf{C_2}$).

Propagate the Substitution on Node and Port Names

In Figure A.23 we give the rules that apply the existing substitution on node and port names. The rules ι_{27} and ι_{28} can be applied at anytime. However, for avoiding variable clashes during matching, these rules applying substitution should be applied with the highest priority.

$$\langle i : X \parallel h\rangle, \langle j : \Rightarrow \parallel h\rangle(\!|i \doteq j^\frown(h, bh)|\!),\ \sigma \longrightarrow_{\iota_{27}}$$
$$\langle i : A \parallel h\rangle, \langle j : \Rightarrow \parallel h\rangle(\!|i \doteq j^\frown(h, bh)|\!),\ \sigma$$
$$\text{if } \sigma(X) = A$$

and

$$\langle i : \parallel x\rangle, \langle j : \Rightarrow \parallel p\rangle (\!| i \doteq j^\frown(x,p) |\!),\ \sigma \longrightarrow_{\iota 28}$$
$$\langle i : \parallel a\rangle, \langle j : \Rightarrow \parallel p\rangle (\!| i \doteq j^\frown(x,p) |\!),\ \sigma$$
$$\text{if } \sigma(x) = a$$

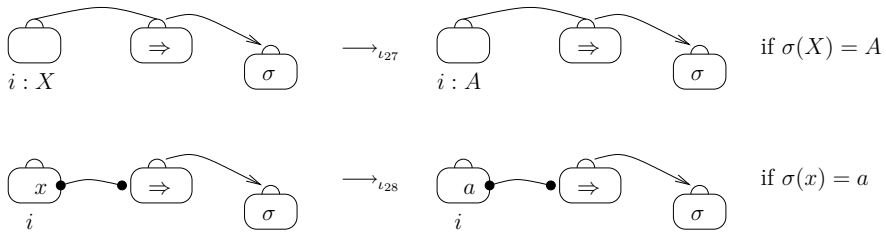


Figure A.23: Internal evaluation rules ι_{27} and ι_{28}: Applying the substitution on node and port names

A.2 Replacement

After all evaluation rules describing the matching are exhaustively applied, the replacement rules can be applied.

The rule in Figure A.24 describes the procedure of translating a bridge. The node i comes from the matched subgraph of the subject G, and the node j is either from the context port graph or from the matched subgraph too. Since the node i' from the instantiated right-hand side will replace the node i, the edge (i, j) is translated into (i', j). The orientation of the edge between i and j is no relevant, however it is the same with the one for the edge between i' and j in the right-hand side of the rule. In the term-like representation, we consider that i is the source node.

$$\begin{array}{l}\langle i : A \parallel a\rangle, \langle j : B \parallel b\rangle, \langle k :\Rightarrow \parallel p\rangle, \langle i' : A' \parallel a'\rangle (\!| \\ i \doteq j^\frown(a,b), k^\frown(a,p), \\ k \doteq i'^\frown(p,a') |\!) \\ \longrightarrow_{\iota 29} \\ \langle i : A \parallel a\rangle, \langle j : B \parallel b\rangle, \langle k :\Rightarrow \parallel p\rangle, \langle i' : A' \parallel a'\rangle (\!| \\ i' \doteq j^\frown(a',b), \\ i \doteq k^\frown(a,p), \\ k \doteq i'^\frown(p,a') |\!)\end{array}$$

The instantiated nodes from the right-hand side of the applied abstraction are properly connected to the application node, instead of the arrow node (see Figure A.25). The substitution is applied on node identifiers only in the nodes from the right-hand side of the abstractions during the replacement, since, otherwise, different nodes with the same

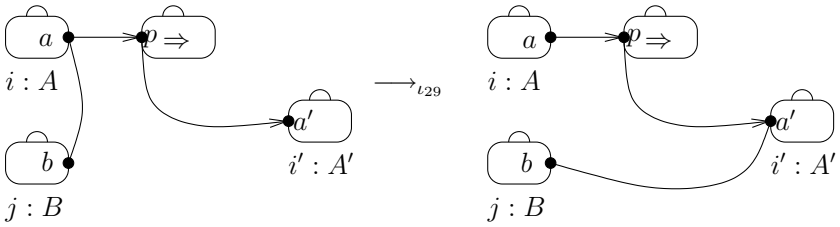


Figure A.24: Internal evaluation rule ι_{29}: Translating a bridge

identifier may occur in the port graph. The evaluation rule ι_{30} has a lower priority than ι_{29}.

$$\langle j : @ \parallel h\rangle, \langle k :\Rightarrow \parallel p\rangle, \langle i' : A' \parallel a', h\rangle(\!|k \doteq i'^\frown(p, a')|\!), \sigma \longrightarrow_{\iota_{30}}$$
$$\langle j : @ \parallel h\rangle, \langle k :\Rightarrow \parallel p\rangle, \langle \sigma(i') : A' \parallel a'\rangle(\!|j \doteq i'^\frown(h, h)|\!), \sigma$$

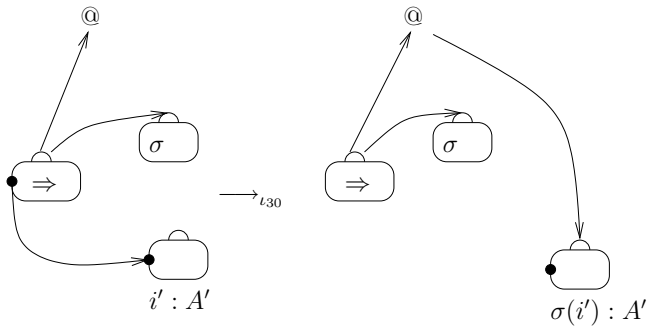


Figure A.25: Internal evaluation rule ι_{30}: Replacement

After the matching process, the handlers of the matched nodes from the left-hand side of the applied abstraction are connected to the black hole of the main arrow of the abstraction. The rule ι_{31} from Figure A.26 deletes all these nodes.

$$\langle i : A \parallel h\rangle, \langle j :\Rightarrow \parallel h, bh\rangle(\!|i \doteq j^\frown(h, bh)|\!) \longrightarrow_{\iota_{31}} \langle j :\Rightarrow \parallel h, bh\rangle(\!|\ |\!)$$

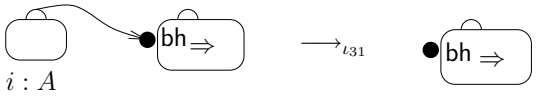


Figure A.26: Internal evaluation rule ι_{31}: Deleting a matched node

After all bridges are translated, the nodes from the matched subgraph in the subject

port graph are also deleted by repeated reductions using the rule ι_{32}:

$$\langle i : A \parallel a\rangle, \langle k : \Rightarrow \parallel p\rangle(\!|i \backsimeq k^\frown(a, p)|\!) \longrightarrow_{\iota_{32}} \langle k : \Rightarrow \parallel p\rangle(\!|\,|\!)$$

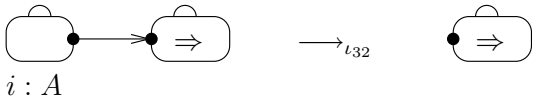


Figure A.27: Internal evaluation rule ι_{32}: Removing the nodes from the matched subgraph after all bridges are translated

The evaluation rule ι_{33} moves every node connected to the application node in the simple world above, while the rule ι_{34} removes the arrow node and the substitution, as well as the application node (Figure A.28). These rules as well as ι_{32} have a lower priority than the other replacement rules. Also, ι_{34} has a lower application priority than ι_{32} and ι_{33}.

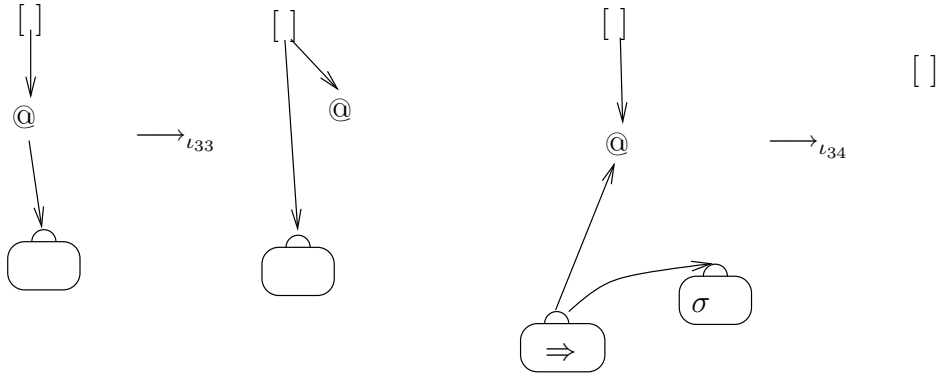


Figure A.28: Internal evaluation rules ι_{33} and ι_{34}: Removing the application node and the arrow node

B Overview of the TOM System

TOM[1][BBK+07a] is a language extension which adds strategic rewriting capabilities to Java. A TOM program is the combination of a host program in Java with code fragments delimited by some special purpose constructs to define term rewrite systems.

The functionalities introduced by TOM are syntactic matching, associative matching with unit axioms, anti-patterns [KKM07], conditional rewriting, support for built-in data-types, XML transformation facilities, and a modular strategy library. TOM has been used to implement various applications but one of the biggest applications is the TOM compiler itself, written in TOM and Java.

The constructs of the TOM language useful for our purposes are the following:

- `%match` corresponds to an extension of `switch/case` construct in functional programming languages, which allows discriminating among a list of patterns.
- ‘ (the backquote construct) is used to build terms from Java values.
- `%strategy` groups rules to form the basic building blocks for constructing more complex strategies, e.g. *innermost, outermost, top down, etc.*.

Therefore, a program is a list of TOM constructs interleaved with some sequences of characters (from the ocean language). During the compilation process, all TOM constructs are translated and replaced by instructions of the host-language.

The following example shows how the addition of Peano integers can be defined. This supposes the existence of a data-structure and a mapping (defined using `%op`) where Peano integers are constructed using *zero* and *successor*: the integer 3 is denoted by *suc(suc(suc(zero)))* for example.

```
public class PeanoExample {
  %op Term zero() { ... }
  %op Term suc(Term) { ... }
  ...
  Term plus(Term t1, Term t2) {
    %match(t1, t2) {
      x,zero   -> { return ‘x; }
      x,suc(y) -> { return ‘suc(plus(x,y)); }
    }
  }
  void run() {
   System.out.println("plus(1,2) = " +plus(‘suc(zero),‘suc(suc(zero))));
  }
}
```

[1] http://tom.loria.fr

In this example, given two terms t_1 and t_2 (that represent Peano integers), the evaluation of `plus` returns the sum of t_1 and t_2. This is implemented by pattern matching: t_1 is matched by x, t_2 is possibly matched by the two patterns $zero$ and $suc(y)$. When $zero$ matches t_2, the result of the addition is x (with $x = t_1$, instantiated by matching). When $suc(y)$ matches t_2, this means that t_2 is rooted by a suc symbol: the subterm y is added to x and the successor of this number is returned, using the ‘ construct. The definition of `plus` is given in a functional programming style, but the `plus` function can be used in Java to perform computations. This example illustrates how the `%match` construct can be used in conjunction with the considered native language.

In order to transform terms, it is necessary to state a relation between a TOM signature (the set of operation symbols in the corresponding term rewrite system) and Java objects. This relation is a mapping, either defined by hand or through an auxiliary tool, called Gom [Rei07], which is distributed with the TOM environment. Gom automatically generates the data structure implementation for a given term signature.

It is possible to define preferred canonical forms for algebraic terms in Gom through the *hook* mechanism [Rei07]. Hooks in Gom can contain TOM and Java code, which is executed every time a new term is created. Hooks are executed until a normal form for the term created is reached. In the code that follows we show an example of the use of hooks. The code implements *constant folding*: every time a term rooted by `Add` and `Mul` has subterms of sort `Nat`, in a small language for arithmetic expressions.

```
module Expressions
  imports String int
  abstract syntax
  Bool = True()
       | False()
       | Eq(lhs:Expr, rhs:Expr)
  Expr = Id(stringValue:String)
       | Nat(intValue:int)
       | Add(lhs:Expr, rhs:Expr)
       | Mul(lhs:Expr, rhs:Expr)
  Add:make(l,r) {
    %match(Expr l, Expr r) {
      Nat(lvalue), Nat(rvalue) -> { return ‘Nat(lvalue + rvalue); }
    }
  }
  Mul:make(l,r) {
    %match(Expr l, Expr r) {
      Nat(lvalue), Nat(rvalue) -> { return ‘Nat(lvalue * rvalue); }
    }
  }
```

The use of strategies in TOM is illustrated by the following example:

```
import eval.mydsl.types.*;
import tom.library.sl.*;
  public class Eval {
```

```
    %include { sl.tom }
     %gom {
        module mydsl
          imports int String
          abstract syntax
              Expr = val(v:int)
                   | var(n:String)
                   | plus(Expr*)
                   | mult(Expr*)
     }

   %strategy EvalExpr() extends `Identity() {
     visit Expr {
       plus(l1*,val(x),l2*,val(y),l3*) -> {
           return `plus(l1*,l2*,l3*,val(x + y));
       }
       mult(l1*,val(x),l2*,val(y),l3*) -> {
           return `mult(l1*,l2*,l3*,val(x * y));
       }
       plus(v@val(_)) -> {
           return `v;
       }
       mult(v@val(_)) -> {
           return `v;
       }
     }
    }

   public static void main(String[] args) throws VisitFailure {
     Expr e = `plus(val(2), var("x"), mult(val(3), val(4)), var("y"), val(5));
     Expr res = (Expr) `Innermost(EvalExpr()).visit(e);
     System.out.println(e + "\n" + res);
    }
}
```

Above, symbols defined in the `%gom` signature followed by `*` are "list" symbol. They are associative with unit functions, where the empty list is the neutral element. The declaration `EvalExpr` corresponds to a term rewrite system which simplifies expressions. Variables followed by `*` match lists of elements. It is meant to be applied on any sub-expression but not in top positions. We choose to apply the rewrite system in an innermost way.

TOM also offers the possibilities of representing and rewriting term graphs. Term-graphs are represented as terms with pointers where the pointers are specified using label constructors. These pointers are defined by a relative path inside the term [BM08]. Using the Gom option `--termgraph` or `--termpointer`, it is possible to automatically generate from a signature the extended version for term graphs or terms with pointers respectively. As the Tom terms are implemented with maximal sharing, so are the term graphs. For every sort `T`, two new constructors are added to the program:

`LabT(label:String, term:T)` for labeling a position in the term, and `RefT(label:String)` for a pointer to a labeled position. Then a cyclic term can be defined as follows:

```
Term cyclicTerm = `LabTerm("l",f(RefTerm("l")));
```

C 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 5.2. 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 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 6.7 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.

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:

`seq(repeat(first(`**r2**, **r1**`)), seq(`**r3**, `seq(`**r4**, **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 6. 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)
...
```

Bibliography

[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. 118

[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. 4, 121

[AK07a] Oana Andrei and Hélène Kirchner – "A Rewriting Calculus for Multigraphs with Ports.", *Proceedings of RULE'07*, 2007. 4

[AK07b] — , "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. 4, 5, 114

[AK08a] — , "A Biochemical Calculus Based on Strategic Graph Rewriting", *Preproceedings of the 3rd International Conference on Algebraic Biology*, 2008. 5, 114

[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. 5, 108

[AK08c] — , "Strategic Port Graph Rewriting for Autonomic Computing", *TFIT*, 2008. 5, 108

[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. 106

[AL08] Oana Andrei and Dorel Lucanu – "Strategy-Based Proof Calculus for Membrane Systems", *Proceedings of the 7th International Workshop on Rewriting Logic and its Applications*, 2008. 122

[Ale99] Vladimir Alexiev – "Non-deterministic Interaction Nets", Thesis, University of Alberta, 1999. 106

[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. 2, 34

[BB92] Gérard Berry and Gérard Boudol – "The Chemical Abstract Machine.", *Theoretical Computer Science* **96** (1992), no. 1, p. 217–248. 2, 25

[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. 106

[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. 28

[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. 21, 171

[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. 106

[BBK+07c] — , Tom Manual, LORIA, Nancy (France), version 2.5 ed., July 2007. 21

[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. 4, 118

[BCKK04] Paolo Baldan, Andrea Corradini, Barbara König and Bernhard König – "Verifying a Behavioural Logic for Graph Transformation Systems", *Electronic Notes in Theoretical Computer Science* **104** (2004), p. 5–24. 124

[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. 79

[BFFM06] Luca Bianco, Federico Fontana, Giuditta Franco and Vincenzo Manca – "P Systems for Biological Dynamics", Applications of Membrane Computing (G. Ciobanu, M. J. Pérez-Jiménez and G. Paun, eds.), Natural Computing Series, Springer, 2006, p. 83–128. 122

[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. 2, 25

[BFR06a] — , "A Generalized Higher-Order Chemical Computation Model", *Electronic Notes in Theoretical Computer Science* **135** (2006), no. 3, p. 3–13. 2, 26, 107

[BFR06b] — , "Generalised multisets for chemical programming", *Mathematical Structures in Computer Science* **16** (2006), no. 4, p. 557–580. 53

[BFR06c] — , "Towards chemical coordination for grids", *SAC* (H. Haddad, ed.), ACM, 2006, p. 445–446. 2, 26

[BFR07] — , "Programming Self-Organizing Systems with the Higher-Order Chemical Language", *International Journal of Unconventional Computing* **3** (2007), no. 3, p. 161–177. 2, 26, 107

[BG04] Francesco Bernardini and Marian Gheorghe – "Population P Systems", *Journal of Universal Computer Science* **10** (2004), no. 5, p. 509–539. 122

[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. 52

[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. 4, 118

[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. 52

[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. 52

[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. 21

[BKN87] Dan Benanav, Deepak Kapur and Paliath Narendran – "Complexity of Matching Problems", *J. Symb. Comput.* **3** (1987), no. 1/2, p. 203–216. 15

[BM86] Jean-Pierre Banatre and Daniel Le Metayer – "A New Computational Model and Its Discipline of Programming.", Tech. Report RR-566, INRIA, 1986. 1, 25

[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. 173

[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. 124

[BN98] Franz Baader and Tobias Nipkow – *Term Rewriting and All That.*, Cambridge University Press, 1998. 9, 11, 14

[BRF04] Jean-Pierre Banâtre, Yann Radenac and Pascal Fradet – "Chemical Specification of Autonomic Systems", *IASSE*, ISCA, 2004, p. 72–79. 28, 108

[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. 124

[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. 13

[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. 73

[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. 4, 120

[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. 3

[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. 107, 120, 121

[CBL04] Arjav J. Chakravarti, Gerald Baumgartner and Mario Lauria – "Application-Specific Scheduling for the Organic Grid", *GRID* (R. Buyya, ed.), IEEE Computer Society, 2004, p. 146–155. 107

[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. 120

[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. 27

[CG00] Luca Cardelli and Andrew D. Gordon – "Mobile ambients.", *Theoretical Computer Science* **240** (2000), no. 1, p. 177–213. 107

[CGP00] Edmund M. Clarke, Orna Grumberg and Doron A. Peled – *Model Checking*, MIT Press, 2000. 6, 51, 123, 125, 128, 129

[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. 27

[CK01] — , "The Rewriting Calculus - Part I and II", *Logic Journal of the IGPL* **9** (2001), no. 3, p. 427—498. 3, 27, 28, 34, 53

[CKL02] Horatiu Cirstea, Claude Kirchner and Luigi Liquori – "Rewriting Calculus with(out) Types", *Electronic Notes in Theoretical Computer Science* **71** (2002), p. 3–19. 27

[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. 27, 41

[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. 18, 19

[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. 6, 124

[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. 3, 28

[Dau92] Max Dauchet – "Simulation of Turing Machines by a Regular Rewrite Rule", *Theoretical Computer Science* **103** (1992), no. 2, p. 409–420. 14

[Der82] Nachum Dershowitz – "Orderings for Term-Rewriting Systems", *Theoretical Computer Science* **17** (1982), p. 279–301. 14

[DL04] Vincent Danos and Cosimo Laneve – "Formal Molecular Biology.", *Theoretical Computer Science* **325** (2004), no. 1, p. 69–110. 4, 115, 120

[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. 107

[DZB01] Peter Dittrich, Jens Ziegler and Wolfgang Banzhaf – "Artificial Chemistries - A Review.", *Artificial Life* **7** (2001), no. 3, p. 225–275. 3

[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. 18, 19, 55

[EGPP99] Hartmut Ehrig, M. Gajewsky and Francesco Parisi-Presicce – "High-level replacement systems applied to algebraic specifications and Petri nets", Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism, and Distribution (H. Ehrig, H.-J. Kreowski, U. Montanari and G. Rozenberg, eds.), World Scientific, 1999, p. 341–399. 30

[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. 18

[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. 75

[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. 120

[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. 18, 55

[EMS04] Steven Eker, Jose Meseguer and Ambarish Sridharanarayanan – "The Maude LTL Model Checker", *Electronic Notes in Theoretical Computer Science* **71** (2004), p. 162–187. 128

[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. 55

[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. 19, 65

[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. 2, 37

[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. 121

[FGK03] Olivier Fissore, Isabelle Gnaedig and Hélène Kirchner – "Simplification and termination of strategies in rule-based languages.", *PPDP*, ACM, 2003, p. 124–135. 121

[FM98] Pascal Fradet and Daniel Le Métayer – "Structured Gamma.", *Sci. Comput. Program.* **31** (1998), no. 2-3, p. 263–289. 2

[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. 104, 106

[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. 88, 90

[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. 2, 54

[Gil77] Daniel T. Gillespie – "Exact stochastic simulation of coupled chemical reactions.", *J. Phys. Chem.* **81** (1977), no. 25, p. 2340–2361. 52

[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. 11

[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. 2

[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. 53

[GM02b] — , "The Topological Structures of Membrane Computing", *Fundam. Inform.* **49** (2002), no. 1–3, p. 123–145. 54

[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). 120

[HHT96] Annegret Habel, Reiko Heckel and Gabriele Taentzer – "Graph Grammars with Negative Application Conditions", *Fundam. Inform.* **26** (1996), no. 3/4, p. 287–313. 108

[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. 4, 118

[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. 52, 122

[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. 15

[JM04] Ole Hogh Jensen and Robin Milner – "Bigraphs and mobile processes (revised)", Tech. Report 580, University of Cambridge, February 2004. 79

[KC03] Jeffrey O. Kephart and David M. Chess – "The Vision of Autonomic Computing", *IEEE Computer* **36** (2003), no. 1, p. 41–50. 107

[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. 10, 19, 21

[KKM07] Claude Kirchner, Radu Kopetz and Pierre-Etienne Moreau – "Anti-Pattern Matching.", *Proceedings of the 16th European Symposium on Programming - ESOP'07*, Lecture Notes in Computer Science, vol. 4421, Springer, 2007, p. 110–124. 171

[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. 21

[Klo80] Jan Willem Klop – "Combinatory Reduction Systems", Thesis, CWI, Amsterdam, 1980. 104

[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. 95

[KMT08] Jean Krivine, Robin Milner and Angelo Troina – "Stochastic Bigraphs", *The 24th Conference on the Mathematical Foundations of Programming Semantics*, 2008. 79, 80, 148

[Laf90] Yves Lafont – "Interaction Nets", *POPL*, 1990, p. 95–108. 106

[LET08] Leen Lambers, Hartmut Ehrig and Gabriele Taentzer – "Sufficient Criteria for Applicability and Non-Applicability of Rule Sequences", *International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT)*, 2008. 80

[Lin68] Aristid Lindenmayer – "Mathematical models for cellular interaction in development", *Journal of Theoretical Biology* **I and II** (1968), no. 18, p. 280–315. 1

[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. 107, 115, 120

[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. 19, 80

[Mac98] Saunders Mac Lane – *Categories for the Working Mathematician*, 2nd ed., Graduate Texts in Mathematics, Springer, 1998. 15, 75

[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. 19

[Mes92] José Meseguer – "Conditioned Rewriting Logic as a United Model of Concurrency.", *Theoretical Computer Science* **96** (1992), no. 1, p. 73–155. 51

[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. 91, 93, 106

[Mil99] Robin Milner – *Communicating and Mobile Systems: the Pi-Calculus*, Cambridge University Press, 1999. 5, 120

[Mil01] — , "Bigraphical Reactive Systems.", *CONCUR* (K. G. Larsen and M. Nielsen, eds.), Lecture Notes in Computer Science, vol. 2154, Springer, 2001, p. 16–35. 79

[Mil06] — , "Pure bigraphs: Structure and dynamics.", *Inf. Comput.* **204** (2006), no. 1, p. 60–122. 79, 107

[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. 21

[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. 1

[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. 6, 124

[Neu66] John Von Neumann – *Theory of Self-Reproducing Automata*, University of Illinois Press, Champaign, Illinois, 1966. 1

[New42] M.H.A. Newman – "On theories with combinatorial definition of "equivalence"", *Annals of Mathematics* **43** (1942), no. 2, p. 223–243. 10

[Pau02] Gheorghe Paun – *Membrane Computing. An Introduction*, Springer, 2002. 2, 52, 122

[Pau06] — , "Twenty Six Research Topics About Spiking Neural P Systems.", Available at the P Systems Web Page: `http://psystems.disco.unimib.it`., 2006. 121

[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. 124

[PE93] Marinus J. Plasmeijer and Marko C. J. D. van Eekelen – *Functional Programming and Parallel Graph Rewriting*, Addison-Wesley, 1993. 19

[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. 106

[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. 75, 77, 78

[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. 2

[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. 15

[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. 107

[Rad07] Yann Radenac – "Programmation "chimique" d'ordre supérieur", Thèse de doctorat, Université de Rennes 1, April 2007. 2, 5, 25, 53, 148

[Rei07] Antoine Reilles – "Canonical Abstract Syntax Trees", *WRLA'06*, vol. 176, Electronic Notes in Theoretical Computer Science, no. 4, 2007, p. 165–179. 172

[Roz97] G. Rozenberg (ed.) – *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, World Scientific, 1997. 18, 55

[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. 107

[SC01] L. A. Segel and I. R. Cohen (eds.) – *Design Principles for the Immune System and Other Distributed Autonomous Systems*, Oxford University Press, 2001. 122

[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. 19, 73

[Sch99] H.J. Schneider – "Describing systems of processes by means of high-level replacement", Handbook of Graph Grammars and Computing by Graph Transformations, Volume 3: Concurrency, Parallelism, and Distribution (H. Ehrig, H.-J. Kreowski, U. Montanari and G. Rozenberg, eds.), World Scientific, 1999, p. 401–450. 30

[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. 54

[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. 52

[Spi06] Antoine Spicher – "Transformation de collections topologiques de dimension arbitraire. Application à la modélisation de systèmes dynamiques", Thesis, Université d'Évry, 2006. 2, 52, 53

[ST99] Ron Shamira and Dekel Tsura – "Faster Subtree Isomorphism", *Journal of Algorithms* **33** (1999), no. 2, p. 267–280. 80

[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. 19, 65, 73

[Ull76] J. R. Ullman – "An Algorithm for Subgraph Isomorphism", *Journal of the ACM* **23** (1976), no. 1, p. 31–42. 19, 80

[Val02] Gabriel Valiente – *Algorithms on Trees and Graphs*, Springer, 2002. 19, 80

[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. 21

[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. 19, 80

List of Figures