# Type Disciplines for Systems Biology

Livio Bioglio

March 8, 2013

Tesi di Dottorato
Scuola di Dottorato in Scienze e Alta Tecnologia, XXV ciclo
Università di Torino
Dipartimento di Informatica

*Ph.D. Thesis*
*Doctoral School on Science and High Technology, XXV cycle*
*University of Torino*
*Department of Computer Science*

Supervisore:
*Advisor*:
Prof. Mariangiola Dezani

# Abstract

*During the last years Computer Science and Biology are getting closer, helping and inspiring each other: for example, the Human Genome Project (HGP), the scientific research project with the goal of determining the sequence of chemical base pairs which make up DNA, would have been impossible without computer clusters and pattern recognition algorithms. On the other hand, Computer Scientists are studying vertebrate immune systems for applying their processes and solutions to antivirus softwares. A new interdisciplinary research field was born, the Natural Computing, encompassing both the application of Computer Science tools to Biology, and the use of biological solutions to computational issues. In the first sub-field we can find Systems Biology, a discipline that aims to study complex biological systems by means of computational models. Because of the complexity of biological behaviors, formalisms are usually designed ad-hoc for the biological topics of interest, or they need to be tuned by means of a long set of rules. In this Thesis we present a different approach: we define biological properties through type disciplines, leaving the formalisms as general as possible. From this starting point, we explore several applications of types in Systems Biology: the first one checks that biological compartments contain rules having certain features defined by the modeler, the second one allows to specify a set of elements required or avoided by other elements, the third one is an improvement of the previous one, the fourth one computes the rate of a stochastic transition by means of the types of the involved elements, and finally we end with a minimal Object Oriented Calculus for rewrite systems.*

# Contents

# Chapter 1

# Introduction

Biological systems are usually composed by several simple components. The behavior of a biological system is not obvious from the properties of the individual parts, because it is the result of the interplay among these integrated parts. These kinds of systems are present in several scientific disciplines, such as Economics, Metereology, Sociology, and so on: they are called complex systems. Due to their complex nature, they can be modeled only through elaborate tools.

The most common approach of biologists to describe biological systems is based on the use of deterministic mathematical means (like, e.g., Ordinary Differential Equations), and makes it possible to abstractly reason on the behavior of biological systems and to perform a quantitative *in silico* investigation. This kind of modeling, however, becomes more and more difficult, both in the specification phase and in the analysis processes, when the complexity of the biological systems taken into consideration increases. This has probably been one of the main motivations for the application of Computer Science formalisms to the description of biological systems [60]. Other motivations can be found in the fact that the use of formal means of Computer Science permits the application of analysis methods that are practically unknown to biologists, such as static analysis and model checking.

According to this idea, in the last few years many formalisms originally developed by computer scientists to model systems of interacting components have been applied to Biology, such as Petri Nets [49] and Hybrid Systems [4]. As time passed by, computer scientists and biologists began to develop new formalisms, more suited to biological systems.

At the same time, some computational models inspired by biological properties were proposed: the most notable examples are P Systems [53] and their variants. They are based upon the structure of biological cells, abstracting from the way in which molecules interact and cross cell membranes. P systems and their variants offer the possibility of solving NP-complete problems in less-than exponential time, or even in linear time, by trading space for time [31]: for example, it was proven that P systems with minimal parallelism can solve the SAT (boolean satisfiability) problem in linear time [30]. This computational power is still theoretical, because there is no biological implementation of a P system.

In the remaining of the Chapter we put the thesis contribution in the context of the current literature. We start by presenting some interesting formalisms, and there we introduce stochastic frameworks built over them. Section 1.3 presents several formalisms specialized for describing the spatial properties of biological systems, in particular the ones based on *membranes*. Section 1.4 is dedicated to the main topic of this essay, the application of Type Theory in System Biology: the Section proposes some motivations and a series of formalisms and extensions proposed in this field. Finally, Section 1.5 presents the structure of the thesis.

## 1.1 Formalisms

Computer Science's formalisms designed for Systems Biology can be divided into three categories: automata-based models [4, 49], rewrite systems [37, 53], and process calculi [60, 61, 59]. Models based on automata have the great advantage of allowing the direct use of many verification tools such as model checkers. Models based on rewrite systems describe biological systems with a notation that can be easily understood by biologists. However, automata-like models and rewrite systems are not compositional. Studying the behavior of a system componentwise is in general ensured by process calculi, included those commonly used to describe biological systems. Let we present a brief overview on these formalisms.

*Bioambients* [59] is a calculus in which biological systems are modeled using a variant of the ambient calculus. In Bioambients both membranes and elements are modeled by ambients, and activities by capabilities (enter, exit, expel, etc.). An evolution of this calculus are *Brane Calculi* [27], the main differences consisting in the fact that they deal with membranes representing the sites of activity, and a computation can happen on the membrane surface.

# 1. INTRODUCTION

The $\pi$-*calculus* and new calculi based on it [57, 59] have been particularly successful in modeling biological systems, as they permit a compositional description. Interactions of biological components are modeled as communications on channels whose names can be passed; sharing names of private channels makes possible to model biological compartments.

*P Systems* [53] are a biologically inspired computational model. A P System is formed by a membrane structure: each membrane may contain molecules, represented by symbols of an alphabet, other membranes and rules. The rules contained into a membrane can be applied only to the symbols contained in the same membrane: these symbols can be modified or moved across membranes. The key feature of P Systems is the maximal parallelism, i.e., in a single evolution step all symbols in all membranes evolve in parallel, and every applicable rule is applied as many times as possible. Later on, through the introduction of ad-hoc features, they have been applied to describe and analyze biological systems [16, 56].

$\kappa$-*calculus* is a formalism proposed by Danos and Laneve [38] that idealizes protein-protein interactions using graphs and graph-rewriting operations. A protein is a node with a fixed number of sites, that may be bound or free. Proteins may be assembled into complexes by connecting two-by-two bound sites of proteins, thus building connected graphs. Collections of proteins and complexes evolve by means of reactions, which may create or remove proteins and bounds between proteins: $\kappa$-calculus essentially deals with complexations and decomplexations, where complexation is a combination of substances into a new substance called complex, and the decomplexation is the reaction inverse to complexation, when a complex is dissociated into smaller parts. These rules contain variables and are pattern-based, therefore may be applied in different contexts.

The *Calculus of Looping Sequences* (CLS) [12] has no explicit way to model protein domains (however they can be encoded, and a variant with explicit binding has been defined in [9]), but accounts for an explicit mechanism (the *looping sequences*) to deal with compartments and membranes. Thus, while the $\kappa$-calculus seems more suitable to model protein interactions, CLS permits a more natural description of membrane interactions. Another feature lacking in other formalisms is the capability to express ordered sequences of elements. To the best of our knowledge, CLS is the first formalism offering such a feature in an explicit way, thus allowing the modeler to naturally operate over proteins or DNA fragments which should be frequently defined as ordered sequences of elements. A variant of CLS without this capability, called *Calculus of Wrapped Compartments*, is presented in [33].

## 1.2 Stochastic Models

The usual mathematical model of a biological system consists of a collection of coupled ordinary differential equations, where each equation describes a number of system's chemical reactions. The variables are concentrations of participating molecules, and the parameters are reaction rate constants. Their results are accurate when reactions occur homogeneously throughout the compartmental volume, and the number of molecules is high, but without these constraints the models result less precise. Moreover, these models are deterministic: the future evolution of the system derives precisely from its starting conditions. When the system is operating near a critical point, noises can induce some new phenomena that cannot be detected deterministically. In such cases, biologists prefer to study the evolution of the system by means of stochastic models. In the mean case, the results of stochastic and deterministic models often coincide, but the former is more accurate near to critical points, or when the number of molecules is low. This accuracy is counterbalanced by the difficulty of solving the stochastic description either analytically or numerically. The solution is to resort to Monte Carlo type simulations, that produce a random walk through the possible states of the system. Various methods have been developed, but the one which is, *de facto*, the standard way to model quantitative aspects of biological systems is the Gillespie stochastic simulation algorithm [44]. The basic idea of Gillespie's algorithm is that a rate function is associated with each considered chemical reaction which is used as the parameter of an exponential distribution modeling the probability that the reaction takes place. In the standard approach this reaction rate is obtained by multiplying the kinetic constant of the reaction by the number of possible combinations of reactants that may occur in the region in which the reaction takes place, thus modeling the law of mass action. Gillespie's approach simulates the time evolution of a chemically reacting system by determining when the next reaction will occur and what kind of reaction it will be. Kind and time of the next reaction are computed on the basis of a stochastic reaction constant. Gillespie's stochastic simulation algorithm is defined for populations of a well-stirred mixture of $N$ molecular species $\{s_1, ..., s_N\}$ interacting through $M$ chemical reactions $\{R_1, ..., R_M\}$ under the conditions that the molecules are confined to a fixed volume and kept at constant temperature.

The Gillespie's approach is implemented in the stochastic versions of all the formalisms presented in the previous Section: Bioambients [59, 24], $\pi$-calculus [58], P Systems [29], $\kappa$-calculus [36], Calculus of Looping Sequences [11].

Gillespie's algorithm is not the unique stochastic approach to biological systems, and furthermore some assumptions can make the equations under the stochastic engine computationally easier (see [66]). For this reason, some formalism permits to specify the equations used to model the system: in [34], the reaction rate is defined in a more general way by associating to each reduction rule a function which can also define rates based on different principles as, for instance, the Michaelis-Menten nonlinear kinetics.

## 1.3 Spatial Formalisms

For the well-mixed chemical systems (even divided into nested compartments) often found in cellular biology, interaction and distribution analysis are sufficient to study the system's behavior. However, several complex biological phenomena include aspects in which space plays an essential role: key examples are the growth of tissues and organisms, embryogenesis and morphogenesis processes, or cell proliferation. This has encouraged, in recent years, the development of formal models for the description of biological systems in which spatial properties can be taken into account, as required by the emerging field of *spatial systems biology* [63] which aims at integrating the analysis of biological systems with spatial properties. This has brought to the extension of many formalisms developed for the analysis of biological systems with (even continuous) spatial features.

In [28], Cardelli and Gardner develop a calculus of processes located in a three-dimensional geometric space. The calculus introduces a single new geometric construct, called *frame shift*, which applies a three-dimensional space transformation to an evolving process. In such a work, standard notions of process equivalence give rise to geometric invariants.

BioShape [26] is a spatial, particle-based and multi-scale 3D simulator. It treats biological entities of different size as geometric 3D *shapes*. A shape is either basic (polyhedron, sphere, cone or cylinder) or composed (aggregation of shapes glued on common surfaces of contact). Every element involved in the simulation is a 3D process and has associated its physical motion law.

Adding too many features to the model (e.g., coordinates, position, extension, motion direction and speed, rotation, collision and overlap detection, communication range, etc.) could heavily rise the complexity of the analysis. To overcome this risk, a detailed study of the possible subsets of these features, chosen to meet the requirements of particular classes of biological phenomena, might be considered.

A different solution is to abstract all these features, by grouping all the molecules that can interact each others. These molecules are inserted into *ambients*, bounded by *membranes*, in which each molecule can interact with all the other molecules of the ambient, but they cannot have any relation with the molecules of other ambients. Ambients are organized in a tree structure, and molecules can move through adjacent ambients according to parent-child relationship. Sometimes, molecules can also be placed on the membrane surface. Among the formalisms presented in Section 1.1, Bioambients, Brane Calculi, P Systems and Calculus of Looping Sequences offer this expressivity.

In particular, the formal investigation of biological membranes has been initiated by Păun [53] with the definition of P systems, leading to the creation of a new research field, called *Membrane Computing* [54]. In this field, membrane systems are not only used for modeling biological systems, but they are also analyzed as computing devices, by studying their computational power in comparison with the classical notion of Turing computability and their efficiency in solving hard problems (e.g., NP-problems) in less-than exponential time; they are applied even in Linguistics, by understanding membranes as contexts of natural languages [31].

P systems allow to represent not only the mobility of elements between neighboring membranes, but also the mobility between the membranes themselves, as in *Simple*, *Enhanced* and *Mutual Mobile Membranes* [5]. *Simple Mobile Membranes* are a variant of P systems that permits to move membranes inside a neighboring membrane (endocytosis) or outside the membrane where it is placed (exocytosis). *Enhanced Mobile Membranes* represent a variant of Simple Mobile Membranes with more complex operations of endocytosis and exocytosis. Finally, *Mutual Mobile Membranes* represent a variant of Simple Mobile Membranes in which endocytosis and exocytosis are mediated by receptors. The same authors in [7] propose a variant of Mutual Mobile Membranes with objects on surface, and relate them to Brane Calculi. A deeper analysis of Mobile Membranes and their computability power, complexity and relations with Process Calculi with similar mobility features can be found in [6].

## 1.4 Types in Systems Biology

In the last few years there has been a growing interest in the use of type disciplines to enforce biological properties. The aim is to ensure the well-formedness of the model according to some biological property expressed through types, instead of syntax or evolution rules. As a result, the formalism is more general and flexible, and final models result easier to analyze.

The most obvious application of types in System Biology is their implementation in formalisms that permit to express modules. Modularity is the key idea to manage the complexity of biological processes, because it allows molecules or compartments to be specified and then combined. It is usually combined with abstraction, that allows generic properties to be specified independently of specific instances: the result are parametrized modules. An example is *P-Lingua* [41], a programming language for membrane computing which aims to be a standard to define P systems. A P-Lingua program consists of a set of parametrized programming modules composed by a sequence of sentences in P-lingua: these sentences are the membrane structure of the model or the rules and objects contained into these membranes. Modules are executed by using calls, that assign some values to their parameters.

In a formalism with modules, the task of a Type System is usually to check the correspondence between the types of the arguments and the types of the parameters in a module call operation.

*Biochemical Systems (LBS)* [55] combine rule-based approaches to modeling with modularity. Modules may be parametrized on compartments, rates, and species. Species are typed by the names of their component atomic species and of their modification site types: when a method is called, the Type System checks the correspondence between the types of the arguments and the types of the parameters.

A Type System is also implemented in *Little b* [48], a high-level programming language for modular model building. In Little b a modeler can define monomers, composed by a name and a sequence of bond sites: these can connect each other by labeling their bond sites, creating complexes; reactions are pairs of patterns that specify the transformation of complexes matching the first pattern to the second one, and may create or delete links between sites. Sites can be labeled with tags, that specify the kind of link of the site and the kind of links it accepts: this tag-based system serves as Type System, and in particular as a type checker.

A more complex Type System is implemented in the extension of *Kappa* *with agent hierarchies* [35]. A Kappa model consists of a collection of rules and agents; each agent has an associated set of sites. Modelers can define variants on an agent by adding or replacing its sites: the variance relation creates an agent hierarchy. A generic rule is then expanded into a set of concrete rules by replacing each agent in the rule with all appropriate agents below it in the hierarchy: so the hierarchy is used with the purpose to enable rapid development of large rule sets via the mechanism of generic rules. Moreover, the same hierarchical structure is used for a static analysis of the rule set: an analyzer navigates the space of variants of a model looking if, with the current rule set, a specific concrete rule can or cannot take place under a sequence of conditions. Even if this procedure can never prove that a rule is correct, it can be used to reject rules that lead to behaviors incompatible with experimental results.

Formalisms without modules are beginning to implement a type discipline too, even if in this case it is hard to find a general design pattern.

In [43] three type systems are defined for the *Biochemical Abstract Machine*, BIOCHAM, a software environment for modeling biochemical systems that provides tools and languages for describing protein networks (see [1]). The first one is used to infer the functions of proteins in a reaction model, the second one to infer activation and inhibition effects of proteins, and the last one to infer the topology of compartments.

In [8] a type system has been defined to ensure the well-formedness of links between protein sites within the *Linked Calculus of Looping Sequences*, an extension of CLS that permits to link two symbols by labeling them with the same label (see [9]). In particular, the authors use type inference for associating to each reduction rule the minimal set of conditions an instantiation must satisfy in order to assure that applying the rule to a correct system we get a correct system as well.

## 1.5 Motivations and Thesis Overview

Even if promising, the application of types in biological formalisms is not widespread. In order to present the benefits deriving from this approach, this thesis proposes several type disciplines, implementing different biological properties and other features useful to modeling. First of all, we start with Chapter 2 by formally recalling the Calculus of Looping Sequences, the formalism used as a basis for most of the extensions and type systems proposed in the thesis. In the same Chapter, we also recall the Calculus of Wrapped Compartments, and we briefly present its spatial extension proposed in [19], that uses a surface language for expressing spatial positions and transformations in a two-dimensional grid. In Chapter 3 we present the Calculus of Looping Sequences with Local Rules, proposed in [20], an extension of Calculus of Looping Sequences that permits to express biochemical transformations also by means of rules acting like molecules: they are valid only in the ambients where they are present, and can be moved, copied or deleted. This Chapter contains a section about the main topic of the thesis: Types in Systems Biology. In fact, Section 3.3 proposes a type system for checking that compartments must contain rules having certain features. This idea is resumed in the following two Chapters, both dealing with features associated to biological molecules instead of rules. Chapter 4 presents a type discipline for Required and Excluded Elements: it permits to specify which elements must be present and which ones are forbidden in presence of a specific element. It was originally presented in [8], improved in [40] and [17], and finally refined in [21], where we show that the approach of [40] subsumes the one of [8], and propose a semantics that uses both. The advantage of this semantics is that during reduction we first try to verify the (easier) constraints of [40], and in case they are not satisfied pass to verify the ones of [8]. Chapter 5 presents an improvement, proposed in [18], of this type discipline: for complex biological behaviors, the simple presence or exclusion is not sufficient, and we need more granular constraints. The improvement proposed permits to specify the number of elements required or forbidden by other elements. Both type disciplines are enriched with typed reductions, that guarantee the soundness of reduction rules. Also Chapter 6 presents a type system that counts the number of element, but for a completely different purpose: this information is used to compute the rate of a stochastic transition. This Chapter shows a stochastic version of the Calculus of Looping Sequences, proposed in [22], in which each rule for the evolution states explicitly the types of elements whose occurrences may speed-up or slow-down a reaction. Chapter 7 shows a different approach to types in Systems Biology, and a more general Calculus. It proposes a minimal Object Oriented Calculus for biological systems, presented in [18]: each molecule is associated with a class, having a sequence of parameterized rules, some of them peculiar and others derived from its superclass. The rules of the model must be created according to these classes, to ensure the correctness of their biological behavior. Finally, in Chapter 8 we draw our conclusions.

# Chapter 2

# The Calculus of Looping Sequences

## 2.1 Introduction

The *Calculus of Looping Sequences* (CLS for short) [12, 13, 50, 14, 15], is a formalism for describing biological systems and their evolution. CLS is based on term rewriting with a set of predefined rules modeling the activities one would like to describe. CLS terms are constructed by starting from a set of basic constituent elements which are composed with operators of sequencing, looping, containment and parallel composition. Sequences may represent DNA fragments and proteins, looping sequences may represent membranes, parallel composition may represent juxtaposition of elements and populations of chemical species. This permits to combine the notational simplicity of rewrite systems with the advantage of a form of compositionality.

## 2.2 Syntax and Semantics

The Calculus of Looping Sequences is essentially based on term rewriting, hence a CLS model consists of a term and a set of rewrite rules. The term is intended to represent the structure of the modeled system, and the rewrite rules to represent the events that may cause the system to evolve.

We start with defining the syntax of terms. We assume a possibly infinite alphabet $\mathcal{A}$ of symbols ranged over by $a, b, c, \ldots$.

**Definition 2.2.1 (Terms).** Terms $T$ *and* sequences $S$ *of CLS are given by the following grammar:*

$$\begin{array}{lcl} T & ::= & S \;\Big|\; (S)^{\circlearrowleft} \rfloor T \;\Big|\; T \mid T \\ S & ::= & \epsilon \;\Big|\; a \;\Big|\; S \cdot S \end{array}$$

*where $a$ is a generic element of $\mathcal{A}$, and $\epsilon$ represents the empty sequence. We denote by $\mathcal{T}$ the infinite set of terms, and by $\mathcal{S}$ the infinite set of sequences.*

In CLS we have a sequencing operator $\_\cdot\_$, a looping operator $(\_)^{\circlearrowleft}$, a parallel composition operator $\_ \mid \_$ and a containment operator $\_ \rfloor \_$. Sequencing can be used to concatenate elements of the alphabet $\mathcal{A}$. The empty sequence $\epsilon$ denotes the concatenation of zero symbols. A term can be either a sequence or a looping sequence (that is the application of the looping operator to a sequence) containing another term, or the parallel composition of two terms. By definition, looping and containment are always applied together, hence we can consider them as a single binary operator $(\_)^{\circlearrowleft} \rfloor \_$ which applies to one sequence and one term.

We call *compartment* any parallel composition of one or more terms. Given a term of the shape $(S)^{\circlearrowleft} \rfloor T$, its *looping sequence* is $S$ and its *inner compartment* is the term $T$[1].

The biological interpretation of the operators is the following: the main entities which occur in cells are DNA and RNA strands, proteins, membranes, and other macro-molecules. DNA strands (and similarly RNA strands) are sequences of nucleic acids, but they can be seen also at a higher level of abstraction as sequences of genes. Proteins are sequence of amino acids which usually have a very complex three-dimensional structure. In a protein there are usually (relatively) few subsequences, called domains, which actually are able to interact with other entities by means of chemical reactions. CLS sequences can model DNA/RNA strands and proteins by describing each gene or each domain with a symbol of the alphabet. Membranes are closed surfaces, often interspersed with proteins, which may contain something. A closed surface can be modeled by a looping sequence. The elements (or the subsequences) of the looping sequence may represent the proteins on the membrane, and by the containment operator it is possible to specify the content of the membrane. Other macro-molecules can be modeled as single

[1]Here we use this notation instead of the usual one, $(S)^{L} \rfloor T$, because later the letter $L$ will be used for another component, and the usual notation could lead to confusion.

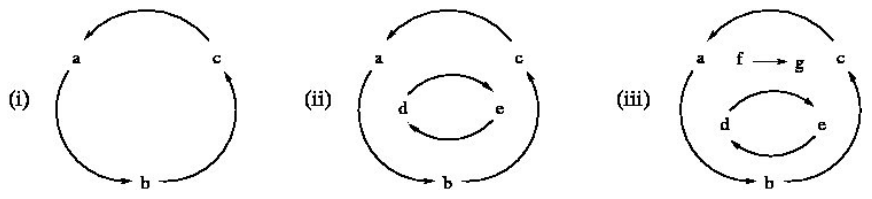

Figure 2.1: (i) represents $(a \cdot b \cdot c)^{\circlearrowleft}$; (ii) represents $(a \cdot b \cdot c)^{\circlearrowleft} \rfloor (d \cdot e)^{\circlearrowleft}$; (iii) represents $(a \cdot b \cdot c)^{\circlearrowleft} \rfloor ((d \cdot e)^{\circlearrowleft} \mid f \cdot g)$.

alphabet symbols, or as short sequences. Finally, juxtaposition of entities can be described by the parallel composition of their representations.

Brackets can be used to indicate the order of application of the operators, and we assume $(\_)^{\circlearrowleft} \rfloor \_$ to have precedence over $\_ \mid \_$. In Figure 2.1 we show some examples of CLS terms and their visual representation, using $(S)^{\circlearrowleft}$ as a short-cut for $(S)^{\circlearrowleft} \rfloor \epsilon$.

In CLS we may have syntactically different terms representing the same structure. We introduce a structural congruence relation to identify such terms.

**Definition 2.2.2 (Structural Congruence).** *The structural congruence relations $\equiv_S$ and $\equiv_T$ are the least congruence relations on sequences and on terms, respectively, satisfying the following rules:*

$$
\begin{gathered}
S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3 \qquad S \cdot \epsilon \equiv_S \epsilon \cdot S \equiv_S S \\
S_1 \equiv_S S_2 \;\; \textit{implies} \;\; S_1 \equiv_T S_2 \\
S_1 \equiv_T S_2 \;\; \textit{and} \;\; T_1 \equiv_T T_2 \;\; \textit{imply} \;\; (S_1)^{\circlearrowleft} \rfloor T_1 \equiv_T (S_2)^{\circlearrowleft} \rfloor T_2 \\
T_1 \mid T_2 \equiv_T T_2 \mid T_1 \qquad T_1 \mid (T_2 \mid T_3) \equiv_T (T_1 \mid T_2) \mid T_3 \qquad T \mid \epsilon \equiv_T T \\
(\epsilon)^{\circlearrowleft} \rfloor \epsilon \equiv_T \epsilon \qquad (S_1 \cdot S_2)^{\circlearrowleft} \rfloor T \equiv_T (S_2 \cdot S_1)^{\circlearrowleft} \rfloor T
\end{gathered}
$$

Rules of structural congruence state the associativity of $\cdot$ and $\mid$, the commutativity of the latter and the neutral role of $\epsilon$. Moreover, the axiom $(S_1 \cdot S_2)^{\circlearrowleft} \rfloor T \equiv_T (S_2 \cdot S_1)^{\circlearrowleft} \rfloor T$ says that looping sequences can rotate. In the following, for simplicity, we will use $\equiv$ in place of $\equiv_T$.

We say that an element $a$ *is present in a sequence* $S$ if $S \equiv S' \cdot a \cdot S''$ for some $S', S''$. An element $a$ *is present in a compartment* $T$ if $T \equiv T' \mid T''$ for some $T', T''$ and either $T' = S$ or $T' = (S)^{\circlearrowleft} \rfloor \_$ for some $S$ and in both cases $a$ is present in $S$.

Rewrite rules will be defined essentially as pairs of terms, with the first term describing the portion of the system in which the event modeled by the rule may occur, and the second term describing how that portion of the system changes when the event occurs. In the terms of a rewrite rule we allow the use of variables. As a consequence, a rule will be applicable to all terms which can be obtained by properly instantiating its variables. Variables can be of three kinds: two of these are associated with the two different syntactic categories of terms and sequences, and one is associated with single alphabet elements. We assume a set of term variables $\mathcal{TV}$ ranged over by $X, Y, Z, \ldots$, a set of sequence variables $\mathcal{SV}$ ranged over by $\widetilde{x}, \widetilde{y}, \widetilde{z}, \ldots$, and a set of element variables $\mathcal{X}$ ranged over by $x, y, z, \ldots$. All these sets are possibly infinite and pairwise disjoint. We denote by $\mathcal{V}$ the set of all variables, $\mathcal{V} = \mathcal{TV} \cup \mathcal{SV} \cup \mathcal{X}$, and by $\rho$ a generic variable of $\mathcal{V}$. Hence, a pattern is a term that may include variables.

**Definition 2.2.3 (Patterns).** Patterns $P$ *and* sequence patterns $SP$ *of CLS are given by the following grammar:*

$$\begin{array}{lcl} P & ::= & SP \;\big|\; (SP)^{\circlearrowleft} \rfloor P \;\big|\; P \mid P \;\big|\; X \\ SP & ::= & \epsilon \;\big|\; a \;\big|\; SP \cdot SP \;\big|\; \widetilde{x} \;\big|\; x \end{array}$$

*where $a$ is a generic element of $\mathcal{A}$, and $X, \widetilde{x}$ and $x$ are generic elements of $\mathcal{TV}, \mathcal{SV}$ and $\mathcal{X}$, respectively. We denote by $\mathcal{P}$ the infinite set of patterns.*

We assume the structural congruence relation to be trivially extended to patterns. An *instantiation* is a partial function $\sigma : \mathcal{V} \to \mathcal{T}$. An instantiation must preserve the kind of variables, thus for $X \in \mathcal{TV}, \widetilde{x} \in \mathcal{SV}$ and $x \in \mathcal{X}$ we have $\sigma(X) \in \mathcal{T}, \sigma(\widetilde{x}) \in \mathcal{S}$ and $\sigma(x) \in \mathcal{A}$, respectively. Given $P \in \mathcal{P}$, with $P\sigma$ we denote the term obtained by replacing each occurrence of each variable $\rho \in \mathcal{V}$ appearing in $P$ by the corresponding term $\sigma(\rho)$. By $\Sigma$ we denote the set of all the possible instantiations and, given $P \in \mathcal{P}$, by $Var(P)$ we denote the set of variables appearing in $P$. Now we define rewrite rules.

**Definition 2.2.4 (Rewrite Rules).** *A rewrite rule, $\Re$, is a pair of patterns, denoted by $P_1 \mapsto P_2$, where $P_1, P_2 \in \mathcal{P}$, $P_1 \not\equiv \epsilon$ and such that $Var(P_2) \subseteq Var(P_1)$.*

**Example 2.2.5.** *An example of rewrite rule is*

$$(a \cdot \widetilde{x})^{\circlearrowleft} \rfloor (b \mid Y) \mapsto b \mid (a \cdot \widetilde{x})^{\circlearrowleft} \rfloor Y.$$

*This rule says that the element $b$, alone, can exit from a looping sequence containing the element $a$.*

A rewrite rule $P_1 \mapsto P_2$ states that a term $P_1\sigma$, obtained by instantiating variables in $P_1$ by some instantiation function $\sigma$, can be transformed into the term $P_2\sigma$. We define the semantics of CLS as a transition system, in which states correspond to terms, and transitions correspond to rule applications.

We define the semantics of CLS by resorting to the notion of contexts.

**Definition 2.2.6 (Evaluation Contexts).** Evaluation Contexts $E$ *are defined as:*

$$E ::= \square \quad \big| \quad E \mid T \quad \big| \quad T \mid E \quad \big| \quad (S)^{\circlearrowleft} \rfloor E$$

*where* $T \in \mathcal{T}$ *and* $S \in \mathcal{S}$*. The context* $\square$ *is called the* empty context*. We denote by* $\mathcal{E}$ *the infinite set of evaluation contexts.*

By definition, every context contains a single hole $\square$. Let us assume $E, E' \in \mathcal{E}$. By $E[T]$ we denote the term obtained by replacing $\square$ with $T$ in $E$; By $E[E']$ we denote context composition, whose result is the context obtained by replacing $\square$ with $E'$ in $E$. The structural equivalence is extended to contexts in the natural way (i.e. by considering $\square$ as a special and unique term).

Rewrite rules can be applied to terms only if they occur in a legal context. Note that the general form of rewrite rules does not allow to have sequences as contexts. A rewrite rule introducing a parallel composition on the right hand side (as $a \mapsto b \mid c$) applied to an element of a sequence (e.g., $m \cdot a \cdot m$) would result into a syntactically incorrect term (in this case $m \cdot (b \mid c) \cdot m$). To modify a sequence, a pattern representing the whole sequence must appear in the rule. For example, rule $a \cdot \widetilde{x} \mapsto a \mid \widetilde{x}$ can be applied to any sequence starting with element $a$, and, hence, the term $a \cdot b$ can be rewritten as $a \mid b$, and the term $a \cdot b \cdot c$ can be rewritten as $a \mid b \cdot c$.

The semantics of CLS is defined as follows.

**Definition 2.2.7 (Semantics).** *Given a finite set of rewrite rules* $\mathcal{R}$*, the* semantics *of CLS is the least relation closed with respect to* $\equiv$ *and satisfying the following (set of) rules:*

$$\frac{\Re = P_1 \mapsto P_2 \in \mathcal{R} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma \quad E \in \mathcal{E}}{E[P_1\sigma] \to E[P_2\sigma]}$$

As usual we denote by $\to^*$ the reflexive and transitive closure of $\to$.

| | $\widetilde{x}$ | $Y$ | $E$ |
|---|---|---|---|
| (1) | $b \cdot c$ | $\epsilon$ | $d \mid (a)^{\circlearrowleft} \rfloor ((a \cdot c)^{\circlearrowleft} \rfloor \Box)$ |
| (2) | $c$ | $(a \cdot b \cdot c)^{\circlearrowleft} \rfloor \epsilon$ | $d \mid (a)^{\circlearrowleft} \rfloor \Box$ |
| (3) | $\epsilon$ | $(a \cdot c)^{\circlearrowleft} \rfloor ((a \cdot b \cdot c)^{\circlearrowleft} \rfloor \epsilon))$ | $d \mid \Box$ |

Figure 2.2: Instantiations and Contexts of Example 2.2.8

Given a set of rewrite rules $\mathcal{R}$, the behavior of a term $T$ is the tree of terms to which $T$ may reduce. Thus, a *model* in CLS is given by a term describing the initial state of the system and by a set of rewrite rules describing all the events that may occur.

**Example 2.2.8.** *Starting from the term*

$$d \mid (a)^{\circlearrowleft} \rfloor ((a \cdot c)^{\circlearrowleft} \rfloor ((a \cdot b \cdot c)^{\circlearrowleft} \rfloor (b)))$$

*we can apply the rule in Example 2.2.5 three times, using the instantiations and evaluation contexts in Fig. 2.2, obtaining the behavior*

$$\begin{array}{ll} & d \mid (a)^{\circlearrowleft} \rfloor ((a \cdot c)^{\circlearrowleft} \rfloor ((a \cdot b \cdot c)^{\circlearrowleft} \rfloor (b))) \\ \to & d \mid (a)^{\circlearrowleft} \rfloor ((a \cdot c)^{\circlearrowleft} \rfloor (b \mid (a \cdot b \cdot c)^{\circlearrowleft} \rfloor \epsilon)) \quad (*) \\ \to & d \mid (a)^{\circlearrowleft} \rfloor (b \mid (a \cdot c)^{\circlearrowleft} \rfloor ((a \cdot b \cdot c)^{\circlearrowleft} \rfloor \epsilon)) \quad (**) \\ \to & b \mid d \mid (a)^{\circlearrowleft} \rfloor ((a \cdot c)^{\circlearrowleft} \rfloor ((a \cdot b \cdot c)^{\circlearrowleft} \rfloor \epsilon)) \quad (***) \end{array}$$

## 2.2.1 Modeling Guidelines

CLS can be used to model biomolecular systems analogously to what is done, e.g, by Regev and Shapiro in [61] for the $\pi$-calculus. An abstraction is a mapping from a real-world domain to a mathematical domain, which may allow highlighting some essential properties of a system while ignoring other, complicated, ones. In [61], Regev and Shapiro show how to abstract biomolecular systems as concurrent computations by identifying the biomolecular entities and events of interest and by associating them with concepts of concurrent computations such as concurrent processes and communications.

The use of rewrite systems, such as CLS, to describe biological systems is founded on a different abstraction. Usually, entities (and their structures) are abstracted by terms of the rewrite system, and events by rewrite rules.

In order to describe cells, it is quite natural to consider molecular populations and membranes. Molecular populations are groups of molecules that are in the same compartment of the cell. As we have said before, molecules can be of many types: they could be classified as DNA and RNA strands, proteins, and other molecules.

DNA and RNA strands and proteins can be seen as non-elementary objects. DNA strands are composed by genes, RNA strands are composed by parts corresponding to the transcription of individual genes, and proteins are composed by parts having the role of interaction sites (or domains). Other molecules are considered as elementary objects, even if they are complexes.

Membranes are considered as elementary objects, in the sense that we do not describe them at the level of the lipids they are made of. The only interesting properties of a membrane are that it may have a content (hence, create a compartment) and that it may have molecules on its surface.

CLS is a very scalable formalism. On the one hand, depending on the level of detail one is interested in the analysis, an atomic element could range from the quark level (in a very low level analysis) to a species individual (in the study of population dynamics).[2] On the other hand, a looping sequence can be used to model cell compartmentalization, or, from a macroscopic point of view, ecoregions bounded by geographical frontiers (expressing the possible environments for a migrant population).

We give now some examples of biomolecular events of interest and their description in CLS. The simplest kind of event is the change of state of an elementary object. Then, we consider interactions between molecules: in particular complexation, decomplexation and catalysis. These interactions may involve single elements of non-elementary molecules (DNA and RNA strands, and proteins). Moreover, there can be interactions between membranes and molecules: in particular a molecule may cross or join a membrane.

Table 2.1 lists some guidelines (taken from [15]) for the abstraction into CLS rules of biomolecular events. Entities are associated with CLS terms: elementary objects are modeled as alphabet symbols, non-elementary objects as CLS sequences and membranes as looping sequences. Biomolecular events are associated with CLS rewrite rules.

[2]Atoms, chemicals, molecules, protein domains, proteins, cells, etc. are other possible elements one could model, at different levels of abstraction, as CLS simple alphabet symbols.

| **Biomolecular Event** | **Examples of CLS Rewrite Rule** |
|---|---|
| State change | $a \mapsto b$<br>$\widetilde{x} \cdot a \cdot \widetilde{y} \mapsto \widetilde{x} \cdot b \cdot \widetilde{y}$ |
| Complexation | $a \mid b \mapsto c$<br>$\widetilde{x} \cdot a \cdot \widetilde{y} \mid b \mapsto \widetilde{x} \cdot c \cdot \widetilde{y}$ |
| Decomplexation | $c \mapsto a \mid b$<br>$\widetilde{x} \cdot c \cdot \widetilde{y} \mapsto \widetilde{x} \cdot a \cdot \widetilde{y} \mid b$ |
| Catalysis | $c \mid P_1 \mapsto c \mid P_2$<br>(where $P_1 \mapsto P_2$ is the catalyzed event) |
| Membrane crossing | $a \mid (\widetilde{x})^{\circlearrowleft} \rfloor X \mapsto (\widetilde{x})^{\circlearrowleft} \rfloor (a \mid X)$<br>$(\widetilde{x})^{\circlearrowleft} \rfloor (a \mid X) \mapsto a \mid (\widetilde{x})^{\circlearrowleft} \rfloor X$<br>$\widetilde{x} \cdot a \cdot \widetilde{y} \mid (\widetilde{z})^{\circlearrowleft} \rfloor X \mapsto (\widetilde{z})^{\circlearrowleft} \rfloor (\widetilde{x} \cdot a \cdot \widetilde{y} \mid X)$<br>$(\widetilde{z})^{\circlearrowleft} \rfloor (\widetilde{x} \cdot a \cdot \widetilde{y} \mid X) \mapsto \widetilde{x} \cdot a \cdot \widetilde{y} \mid (\widetilde{z})^{\circlearrowleft} \rfloor X$ |
| Catalyzed<br>membrane crossing | $a \mid (b \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \mapsto (b \cdot \widetilde{x})^{\circlearrowleft} \rfloor (a \mid X)$<br>$(b \cdot \widetilde{x})^{\circlearrowleft} \rfloor (a \mid X) \mapsto a \mid (b \cdot \widetilde{x})^{\circlearrowleft} \rfloor X$<br>$\widetilde{x} \cdot a \cdot \widetilde{y} \mid (b \cdot \widetilde{z})^{\circlearrowleft} \rfloor X \mapsto (b \cdot \widetilde{z})^{\circlearrowleft} \rfloor (\widetilde{x} \cdot a \cdot \widetilde{y} \mid X)$<br>$(b \cdot \widetilde{z})^{\circlearrowleft} \rfloor (\widetilde{x} \cdot a \cdot \widetilde{y} \mid X) \mapsto \widetilde{x} \cdot a \cdot \widetilde{y} \mid (b \cdot \widetilde{z})^{\circlearrowleft} \rfloor X$ |
| Membrane joining | $(\widetilde{x})^{\circlearrowleft} \rfloor (a \mid X) \mapsto (a \cdot \widetilde{x})^{\circlearrowleft} \rfloor X$<br>$(\widetilde{x})^{\circlearrowleft} \rfloor (\widetilde{y} \cdot a \cdot \widetilde{z} \mid X) \mapsto (\widetilde{y} \cdot a \cdot \widetilde{z} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X$ |
| Catalyzed | $(b \cdot \widetilde{x})^{\circlearrowleft} \rfloor (a \mid X) \mapsto (a \cdot b \cdot \widetilde{x})^{\circlearrowleft} \rfloor X$ |

Table 2.1: Guidelines for the abstraction of biomolecular events into CLS.

## 2.3 Stochastic CLS

A stochastic version of CLS, called *Stochastic Calculus of Looping Sequences* (SCLS), has been defined in [11]. As seen in Section 1.2, the standard way of extending a formalism to model quantitative aspects of biological systems is by incorporating the collision-based stochastic framework presented by Gillespie in [44]. In Gillespies algorithm, a kinetic constant is associated with each considered chemical reaction. Since in CLS chemical reactions are translated into rewrite rules, the first step is surely to add rates to rewrite rules:

**Definition 2.3.1 (Stochastic Rewrite Rules).** *A stochastic rewrite rule is a triple of patterns, denoted by $P_1 \overset{k}{\mapsto} P_2$, where $P_1, P_2 \in \mathcal{P}$, $P_1 \not\equiv \epsilon$ and such that $Var(P_2) \subseteq Var(P_1)$; $k \in \mathbb{R}^{\geq 0}$ is the rate constant.*

The kinetic constant is used to calculate the rate constant of the chemical reaction. Such a constant is obtained by multiplying the kinetic constant of the reaction by the number of possible combinations of reactants that may occur in the system. The resulting rate is then used as the parameter of an exponential distribution modeling the time spent between two occurrences of the considered chemical reaction.

**Remark 2.3.2.** *As we have variables, we have to take into account how they can be instantiated in order to count the number of reactants. The rate of a transition is computed by resorting to a complete counting mechanism to detect all the possible occurrences of patterns within a term that, once the rule is applied, produce the same term. For example, consider the rule $(a \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \mapsto (b \cdot \widetilde{x})^{\circlearrowleft} \rfloor X$ with rate k:*

1. *if the rule is applied to the term $(a \cdot c \cdot a \cdot c)^{\circlearrowleft} \rfloor \epsilon$ the kinetic constant of the rule should be $2 \times k$ since the 2 matches of the pattern in the left-hand-side of the rule with the term are such that the corresponding reductions produce terms congruent to $(b \cdot c \cdot a \cdot c)^{\circlearrowleft} \rfloor \epsilon$, instead*

2. *if the rule is applied to the term $(a \cdot c \cdot a \cdot d)^{\circlearrowleft} \rfloor \epsilon$ the kinetic constant of the rule should be $k$, since in this case the two reductions produce the two terms $(b \cdot c \cdot a \cdot d)^{\circlearrowleft} \rfloor \epsilon$ and $(a \cdot c \cdot b \cdot d)^{\circlearrowleft} \rfloor \epsilon$ that are not congruent, and therefore do not express the same reaction.*

In order to overcome this problem, the authors consider *abstract* and *concrete* patterns. The CLS terms and patterns defined as in Definitions 2.2.1 and 2.2.3 are considered abstract, and they are denoted using a tilde, as in $\widetilde{P}$ and $\widetilde{T}$.

**Definition 2.3.3 (Concrete patterns and terms).** *If $\widetilde{P}$ is an abstract pattern, then a concrete pattern $P$, called a concretion of $\widetilde{P}$, is obtained by assigning to each alphabet symbol syntactically appearing in $\widetilde{P}$ a unique identifier $v \in Id$, where $Id$ is a finite set of identifiers.*

Intuitively, each symbol of the alphabet $\mathcal{A}$ appearing in patterns and terms becomes unique in the concretion by labeling it with a fresh identifier. Another useful notion is the *support*.

**Definition 2.3.4 (Support).** *Given a concrete pattern $P$, we call support the set of identifiers used to label its alphabet symbols, and we denote it by $Supp(P)$. Two concrete patterns $P$ and $P'$ are support-equivalent, written $P \doteqdot P'$, if they differ only by a bijection between their supports which preserves structure. Namely, $P \doteqdot P'$ if and only if $\widetilde{P} \equiv \widetilde{P}'$ and there exists a bijection between $Supp(P)$ and $Supp(P')$. We denote the $\doteqdot$-equivalence class of $P$ by $[[P]]$.*

The definitions of contexts and stochastic rewrite rules are extended to deal with concrete terms in the natural way. Without loss of generality, we assume instantiations to return abstract or concrete terms when applied to abstract or concrete patterns respectively.

We define a notion of occurrence of abstract patterns within a term, in order to define rewrite rules in an abstract way.

**Definition 2.3.5 (Occurrences).** *If $\widetilde{P}$ is an abstract pattern and $T$ a concrete term, an occurrence of $\widetilde{P}$ in $T$ is a pair $(E, P)$, where $P$ is a concretion of $\widetilde{P}$ and $E$ is an evaluation context such that $T \equiv E[P\sigma]$ for some instantiation $\sigma$.*

*An occurrence of a rule $R = (\widetilde{P}_1, \widetilde{P}_2, k)$ in a concrete term $T$ is a pair $(E, P_1)$, where $(P_1, P_2, k)$ is a concretion of $R$ and $T \equiv E[P_1\sigma]$ for some instantiation $\sigma$. If also $T' \doteqdot E[P_2\sigma]$ we say that the occurrence of rule $R$ in $T$ results into a term support-equivalent to $T'$. With $\mathcal{O}(R, T, [[T']])$ we define the set of occurrences of rule $R$ in the term $T$ resulting in a concrete term support-equivalent to $T'$.*

The use of support-equivalence in the definition of $\mathcal{O}(R, T, [[T']])$ allows to consider as a single occurrence the occurrences which differ only for the support in $P_2$ (thus producing different, but support-equivalent, $T'$). The rate of a transition driven by the rule $R = (\widetilde{P}_1, \widetilde{P}_2, k)$ is obtained as the product of the rate $k$ of the stochastic rewrite rule and the number of distinct occurrences of the rule within the term $T$ resulting in $T'$, denoted by $|\mathcal{O}(R, T, [[T']])|$. As a consequence, the stochastic transition system for abstract terms is defined as follows.

**Definition 2.3.6 (Semantics of SCLS).** *Given a finite set $\mathcal{R}$ of stochastic rewrite rules, the* semantics *of SCLS is the least labeled transition satisfying the following rule:*

$$\frac{R = \widetilde{P}_1 \overset{k}{\mapsto} \widetilde{P}_2 \in \mathcal{R} \quad (E, P_1) \in \mathcal{O}(R, T, [[T']]) \quad T \equiv E[P_1\sigma] \quad T' \doteqdot E[P_2\sigma]}{\widetilde{T} \xrightarrow{R, k \cdot |\mathcal{O}(R,T,[[T']])|} \widetilde{T}'}$$

## 2.4 Calculus of Wrapped Compartments

The *Calculus of Wrapped Compartments* (CWC) (see [32, 33]) is a simplified version of CLS, in which the sequences are not allowed by syntax. Moreover, it allows to associate specific proprieties to membranes. These simplifications were proposed in order to write a simulator for CLS. More specifically, it is a calculus for the description of biochemical systems which is based on the notion of a compartment which represents, in some sense, the abstraction of a region with specific properties (characterized by a *label*, a *wrap* and a *content*). Biochemical transformations are described via a set of reduction rules which characterize the behavior of the represented system.

### 2.4.1 Term Syntax

Let $\mathcal{AT}$ be a set of *atomic elements* (*atoms* for short), ranged over by $a$, $b$, ..., and $\mathcal{L}$ a set of *compartment types* represented as *labels* ranged over by $\ell, \ell', \ell_1, \ldots$ A *term* of CWC is a multiset $\overline{t}$ of *simple terms* where a simple term is either an atom $a$ or a compartment $(\overline{a} \rfloor \overline{t'})^{\ell}$ consisting of a *wrap* (represented by the multiset of atoms $\overline{a}$), a *content* (represented by the term $\overline{t'}$) and a *type* (represented by the label $\ell$).

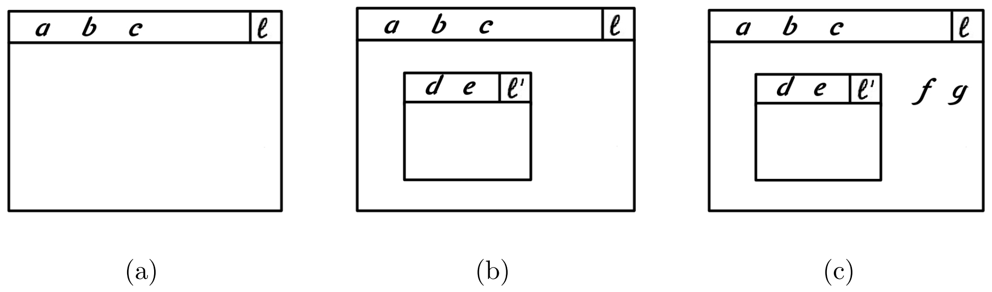

Figure 2.3: **(a)** represents $(a\ \ b\ \ c \rfloor \bullet)^{\ell}$; **(b)** represents $(a\ \ b\ \ c \rfloor (d\ \ e \rfloor \bullet)^{\ell'})^{\ell}$; **(c)** represents $(a\ \ b\ \ c \rfloor (d\ \ e \rfloor \bullet)^{\ell'}\ \ f\ \ g)^{\ell}$

The notation $n * t$ denotes $n$ occurrences of the simple term $t$. We denote an empty term by $\bullet$. An example of CWC term is $2{*}a\ \ b\ \ (c\ \ d \rfloor e\ \ f)^{\ell}$ representing a multiset (multisets are denoted by listing the elements separated by a space) consisting of two occurrences of $a$, one occurrence of $b$ (e.g. three molecules) and an $\ell$-type compartment $(c\ \ d \rfloor e\ \ f)^{\ell}$ which, in turn, consists of a wrap (a membrane) with two atoms $c$ and $d$ (e.g. two proteins) on its surface, and containing the atoms $e$ (e.g. a molecule) and $f$ (e.g. a DNA strand). See Figure 2.3 for some other examples with a simple graphical representation.

### 2.4.2 Rewriting Rules

System transformations are defined by rewriting rules, defined by resorting to CWC terms that may contain variables. We call *pattern* the l.h.s. component $\overline{p}$ of a rewrite rule and *open term* the r.h.s. component $\overline{o}$ of a rewrite rule, defined as multiset of *simple patterns* $p$ and *simple open terms* $o$ given by the following syntax:

$$\begin{array}{lcl|l}
p & ::= & a & (\overline{a}\ \ x \rfloor \overline{p}\ \ X)^{\ell} \\
o & ::= & a & (\overline{q} \rfloor \overline{o})^{\ell} \quad \big| \quad X \\
q & ::= & a & x
\end{array}$$

where $\overline{a}$ is a multiset of atoms, $\overline{p}$ is a pattern (a, possibly empty, multiset of simple patterns), $x$ is a *wrap variable* (can be instantiated by a multiset of atoms), $X$ is a *content variable* (can be instantiated by a CWC term), $\overline{q}$ is a multiset of atoms and wrap variables, and $\overline{o}$ is an open term (a, possibly empty, multiset of simple open terms). Patterns are intended to match, via substitution of variables with ground terms (containing no variables), with compartments (or atoms) occurring as subterms of the term representing the whole system. Note that we force *exactly* one variable to occur in each compartment content and wrap of our patterns and simple patterns. This prevents ambiguities in the instantiations needed to match a given compartment.

A *rewrite rule* is a triple $(\ell, \overline{p}, \overline{o})$, denoted by $\ell : \overline{p} \longmapsto \overline{o}$, where $\overline{p}$ and $\overline{o}$ are such that the variables occurring in $\overline{o}$ are a subset of the variables occurring in $\overline{p}$. The application of a rule $\ell : \overline{p} \longmapsto \overline{o}$ to a term $\overline{t}$ is performed in the following way: 1) Find in $\overline{t}$ (if it exists) a compartment of type $\ell$ with content $u$ and a substitution $\sigma$ of variables by ground terms such that $u = \sigma(\overline{p}\ \ X)$[3] and 2) Replace in $\overline{t}$ the subterm $u$ with $\sigma(\overline{o}\ \ X)$. We write $\overline{t} \longmapsto \overline{t'}$ if $\overline{t'}$ is obtained by applying a rewrite rule to $\overline{t}$. The rewrite rule $\ell : \overline{p} \longmapsto \overline{o}$ can be applied to any compartment of type $\ell$ with $\overline{p}$ in its content (that will be rewritten with $\overline{o}$). For instance, the rewrite rule $\ell : a\ \ b \longmapsto c$ means that in all compartments of type $\ell$ an occurrence of $a\ \ b$ can be replaced by $c$.

While the rule does not change the label $\ell$ of the compartment where the rule is applied, it may change all the labels of the compartments occurring in its content. For instance, the rewrite rule $\ell : (a\ \ x \rfloor X)^{\ell_1} \longmapsto (a\ \ x \rfloor X)^{\ell_2}$ means that, if contained in a compartment of type $\ell$, all compartments of type $\ell_1$ and containing an $a$ in their wrap can change their type to $\ell_2$.

For uniformity reasons we assume that the whole system is always represented by a term consisting of a single compartment with distinguished label $\top$ and empty wrap, i.e., any system is represented by a term of the shape $(\bullet \rfloor \overline{t})^{\top}$, which will be also written as $\overline{t}$, for simplicity.

### 2.4.3 Stochastic CWC

In the stochastic version of CWC [34], each reduction rule is then enriched by the kinetic constant $k$ of the reaction that it represents (notation $\ell : \overline{p} \overset{k}{\mapsto} \overline{o}$). For instance in evaluating the application rate of the stochastic rewrite rule $R = \ell : a\ \ b \overset{k}{\mapsto} c$ (written in the simplified form) to the term $\overline{t} = a\ \ a\ \ b\ \ b$ in a compartment of type $\ell$ we must consider the number of the possible

[3]The implicit (distinguished) variable $X$ matches with all the remaining part of the compartment content.

combinations of reactants of the form $a\ \ b$ in $\bar{t}$. Since each occurrence of $a$ can react with each occurrence of $b$, this number is 4. So the application rate of $R$ is $k \cdot 4$.

### 2.4.4 The CWC Simulator

The CWC simulator [2] is a tool under development at the Computer Science Department of the Turin University, based on the Stochastic version of CWC. It treats CWC models with different rating semantics (law of mass action, Michaelis-Menten kinetics, Hill equation) and it can run independent stochastic simulations over CWC models, featuring deep parallel optimizations for multi-core platforms on the top of FastFlow [3]. It also performs online analysis by a modular statistical framework.

### 2.4.5 Spatial CWC

In this Section we recall [19], where we introduce a surface language for CWC that defines a framework in which the notion of space is included as an essential component of the system. The space is structured as a square grid, whose dimension must be declared as part of the system specification. The surface language provides basic constructs for modeling spatial interactions on the grid. These constructs can be compiled away to obtain a standard CWC model, thus exploiting the existing CWC simulation tool.

We distinguish between two kinds of compartments: *Standard* compartments (corresponding to the usual CWC compartments), used to represent entities (like bacteria or cells) that can move through space, and *Spatial* compartments, used to represent portions of space. Each spatial compartment defines a location in a two dimensional grid through a special atom, called *coordinate*, that occurs on its wrap. A coordinate is denoted by `row.column`, where `row` and `column` are integers. Spatial compartments have distinguished labels, called *spatial labels*, that can be used to provide a specific characterization of a portion of space. For example, the spatial compartment $(\texttt{1.2} \rfloor 2 * b)^{soil}$ represents the cell of the grid located in the first row and the second column, and has type *soil*, the spatial compartment $(\texttt{2.3} \rfloor 3 * b\ \ c)^{water}$ represents a *water*-type spatial compartment in position `2.3`.

### Surface Terms

The initial state of the system under analysis is defined as a set of compartments modeling the two-dimensional grid containing the biological entities of interest.

Let $\Theta$ denote a set of coordinates and $\ell_s$ a spatial label. The notation:

$$\Theta, \ell_s \boxplus\ \bar{t}$$

defines a set of cells of the grid. Namely $\Theta, \ell_s \boxplus\ \bar{t}$ denotes the top level CWC term:

$$(\bullet \rfloor (\mathtt{r_1.c_1} \rfloor \bar{t})^{\ell_s} \ \ \ldots \ \ (\mathtt{r_n.c_n} \rfloor \bar{t})^{\ell_s})^{\top}$$

where $\mathtt{r_i.c_j}$ range over all elements of $\Theta$. To increase the expressivity of the language, few structures to denote portions (i.e. sets of cells) of the grid have been defined: rectangles by $\mathtt{rect}[\mathtt{r.c},\mathtt{r'.c'}]$, where $\mathtt{r.c},\mathtt{r'.c'}$ represent the edges of the rectangle; rows and columns with the constructions $\mathtt{row}[i]$ and $\mathtt{col}[j]$ respectively.

A spatial CWC term is thus defined by the set of grid cells covering the entire grid.

### Surface Rewrite Rules

Rules can model three kinds of events.

*Non-Spatial Events:* are described by standard CWC rules, i.e. by rules of the shape:

$$\ell : \overline{p} \overset{k}{\mapsto} \overline{o}$$

Non-spatial rules can be applied to any compartment of type $\ell$ occurring in any portion of the grid and do not depend on a particular location.

*Spatial Events:* are described by rules that can be applied to specific spatial compartments. These rules allow to change the spatial label of the considered compartment. Spatial events are described by rules of the following shape:

$$\Theta \triangleright \ell_s : \overline{p} \overset{k}{\mapsto} \ell'_s : \overline{o}$$

Spatial rules can be applied only within the spatial compartments with coordinates contained in the set $\Theta$ and with the spatial label $\ell_s$. The application of the rule may also change the label of the spatial compartments $\ell_s$ to $\ell'_s$. This rule is translated into the CWC set of rules:

$$\top : (\mathtt{r_i.c_i}\ x \rfloor \overline{p}\ X)^{\ell_s} \overset{k}{\mapsto} (\mathtt{r_i.c_i}\ x \rfloor \overline{o}\ X)^{\ell'_s} \quad \forall \mathtt{r_i.c_i} \in \Theta.$$

Note that spatial rules are analogous to non spatial ones. The only difference is the explicit indication of the set $\Theta$ which allows to write a single rule instead of a set of rules (one for each element of $\Theta$).

*Spatial Movement Events:* are described by rules considering the content of two adjacent spatial compartments and are described by rules of the following shape:

$$\Theta \lhd \Delta \rhd \ell_{s_1}, \ell_{s_2} : \overline{p_1}, \overline{p_2} \overset{k}{\mapsto} \ell'_{s_1}, \ell'_{s_2} : \overline{o_1}, \overline{o_2}$$

This rule changes the content of two adjacent (according to the possible directions contained in the set of directions $\Delta$[4]) spatial compartments and thus allows to define the movement of objects. The pattern matching is performed by checking the content of a spatial compartment of type $\ell_{s_1}$ located in a portion of the grid defined by $\Theta$ and the content of the adjacent spatial compartment of type $\ell_{s_2}$. Such a rule could also change the labels of the spatial compartments. This rule is translated into the CWC set of rules:

$$\begin{aligned}\top : (\mathtt{r_i.c_i}\ x \rfloor \overline{p_1}\ X)^{\ell_{s1}} (dir(\mathtt{r_i.c_i})\ y \rfloor \overline{p_2}\ Y)^{\ell_{s2}} \overset{k}{\mapsto}\\ (\mathtt{r_i.c_i}\ x \rfloor \overline{o_1}\ X)^{\ell'_{s1}} (dir(\mathtt{r_i.c_i})\ y \rfloor \overline{o_2}\ Y)^{\ell'_{s2}}\end{aligned}$$

for all $\mathtt{r_i.c_i} \in \Theta$ and for all $dir \in \Delta$.

## Implementation

A software module, written in Java by means of the ANTLR parser generator [52], implements the translation of a surface language model into the corresponding standard CWC model that can be executed by the CWC simulator. Notably, this framework could also be applied to other calculi which are able to express compartmentalization and ambients, like the ones presented in Section 1.3. A similar solution was proposed for P systems in [10], where membranes and objects are placed in a two-dimensional discrete space,

[4]The four *direction* operators can be `N, W, S, E` that applied to a range of cells shift them, respectively, up, left, down and right. For instance `E(1.1)` = `1.2`. In the intuitive way, the four diagonal movements (namely, `NW`, `SW`, `NE`, `SE`) are also defined, and the special symbol $\diamond$ denotes the set containing all eight possible directions.

and rules take into account the positions of objects; in addition, the authors permit to specify mutually exclusive objects (i.e. objects that cannot occupy the same position at the same time), that ensure the universality of the formalism.

# Chapter 3

# A Calculus of Looping Sequences with Local Rules

## 3.1 Introduction

In this chapter it is presented the variant of the Calculus of Looping Sequences with global and local rewrite rules (CLSLR, for short) proposed in [20]. Global rules are applied anywhere in a given term wherever their patterns match the portion of the system under investigation, while local rules can only be applied in the compartment in which they are defined. Terms written in CLSLR are thus syntactically extended to contain explicit local rules within the term, on different compartments. Local rules can be created, moved between different compartments and deleted. We feel that having a calculus in which we can model the dynamic evolution of the rules describing the system results in a more natural and direct way to study emerging properties of complex systems. As it happens in nature, where *data* and *programs* are encoded in the same kind of molecular structures, we insert rewrite rules within the terms modeling the system under investigation.

In CLSLR we also enrich CLS with a parallel semantics in which we define a reduction step lead by any number of global and local rules that could be performed in parallel.

Since in this framework the focus is put on local rules, we define a set of *features* that can be associated to each local rule. Features may define general properties of rewrite rules or properties which are strictly related to the model under investigation. We define a *membrane type* for the compartments of our model and develop a type systems enforcing the property that a compartment must contain only local rules with specific features.

Thus, the main characteristics of CLSLR are:

- different compartments can evolve according to different *local* rules;
- the set of *global* rules is fixed;
- local rules are *dynamic*: they can be added, moved and erased according to both global and local rules;
- a *parallel* reduction step allows the application of several global and local rules;
- compartments are enforced to contain only rules with specific *features*.

As a running case study, emphasizing the peculiarities of the calculus, we consider some mitochondrial activity underlining the form of symbiosis between a cell and its mitochondria (see [42]). Mitochondria are membrane-enclosed organelle found in eukaryotic cells that generate most of the cell's energy supply in the form of adenosine triphosphate (ATP). A mitochondrion is formed by two membranes, the outer and the inner membrane, having different properties and proteins on their surfaces. Both membranes have receptors to mediate the entrance of molecules. In Figure 3.1 we show the expression of a gene (encoded in the DNA within the nucleus of the cell) destined to be translated into a protein that will be catch by mitochondria and will then catalyze the production of ATP. In particular, we will model the following steps: (1) genes within the nucleus' DNA are transcribed into mRNA, (2) mRNA moves from the nucleus to the cell's cytoplasm, (3) where it is translated into the protein.

The vast majority of proteins destined for the mitochondria are encoded in the nucleus of the cell and synthesized in the cytoplasm. These are tagged by a signal sequence, which is recognized by a receptor protein in the Transporter Outer Membrane complex (TOM). The signal sequence and adjacent portions of the polypeptide chain are inserted in the intermembranous space through the TOM complex, then in the mitochondrion internal space through a Transporter Inner Membrane complex (TIM). According to this description, we model the following final steps: (4) protein is detected by TOM and brought within the intermembranous space, (5) then, through TIM, in the mitochondrion's inside, (6) where it catalyzes the production of ATP, (7)

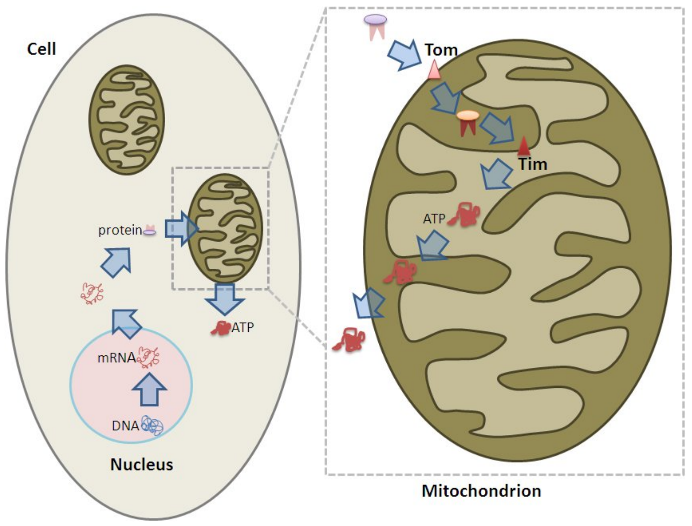

Figure 3.1: From the cell nucleus to mitochondria.

that exits the inner, (8) and the outer mitochondrial membranes towards the cell's cytoplasm.

## 3.2 The calculus

In this Section we present the Calculus of Looping Sequences with Local Rules (CLSLR).

### 3.2.1 Syntax of CLSLR

We assume a possibly infinite alphabet $\mathcal{A}$ of symbols ranged over by $a, b, c, \ldots$, a set of element variables $\mathcal{X}$ ranged over by $x, y, z, \ldots$, a set of sequence variables $\mathcal{SV}$ ranged over by $\widetilde{x}, \widetilde{y}, \widetilde{z}, \ldots$, and a set of term variables $\mathcal{TV}$ ranged over by $X, Y, Z, \ldots$. All these sets are possibly infinite and pairwise disjoint. We denote by $\mathcal{V}$ the set of all variables, $\mathcal{V} = \mathcal{X} \cup \mathcal{SV} \cup \mathcal{TV}$, and by $\chi$ a generic variable of $\mathcal{V}$. Hence, a pattern is a term that may include variables.

**Definition 3.2.1 (Patterns).** Patterns $P$, sequence patterns $SP$ *and* local rules $R$ *of* CLSLR *are given by the following grammar:*

$$\begin{array}{lcl} P & ::= & (SP)^{\circlearrowleft} \rfloor P \;\big|\; P \mid P \;\big|\; L \\ SP & ::= & \epsilon \;\big|\; a \;\big|\; x \;\big|\; SP \cdot SP \;\big|\; \widetilde{x} \\ L & ::= & X \;\big|\; SP \;\big|\; L \mid L \;\big|\; R \\ R & ::= & L \mapsto L \;\big|\; L^{\uparrow SP} \mapsto L^{\uparrow SP} \;\big|\; L^{\downarrow SP} \mapsto L^{\downarrow SP} \end{array}$$

*where $a$ is a generic element of $\mathcal{A}$, and $X, \widetilde{x}$ and $x$ are generic elements of $\mathcal{TV}, \mathcal{SV}$ and $\mathcal{X}$, respectively. Sequence patterns $SP$ defines patterns for sequences of elements, $(SP)^{\circlearrowleft}$ denotes a closed (looping) sequence which may contain other patterns through the $\rfloor$ operator, $\mid$ is used to denote the parallel composition (juxtaposition) of patterns, and $R$ denotes the syntax of local rules that may either exit ($L^{\uparrow SP}$) or enter ($L^{\downarrow SP}$) a closed sequence $SP$. We denote by $\mathcal{P}$ the infinite set of patterns.*
*A local rule $L_1 \mapsto L_2$* is well formed *if $L_1 \not\equiv \epsilon$ and $Var(L_2) \subseteq Var(L_1)$, where $Var(P)$ denotes set of variables appearing in $P$.*
*A local rule $L_1^{\uparrow SP_1} \mapsto L_2^{\uparrow SP_2}$* or *$L_1^{\downarrow SP_1} \mapsto L_2^{\downarrow SP_2}$* is well formed *if $L_1 \not\equiv \epsilon$, $Var(L_2) \subseteq Var(L_1)$ and $Var(SP_2) \subseteq Var(SP_1)$.*

Terms are patterns containing variables only inside local rules. Sequences are closed sequence patterns. We denote by $\mathcal{T}$ the infinite set of terms, ranged over by $T$, and by $\mathcal{S}$ the infinite set of sequences, ranged over by $S$.

An *instantiation* is a partial function $\sigma : (\mathcal{TV} \to \mathcal{T}) \cup (\mathcal{SV} \to \mathcal{S}) \cup (\mathcal{X} \to \mathcal{A})$. Given $P \in \mathcal{P}$, with $P\sigma$ we denote the term obtained by replacing each occurrence of each variable $\chi \in \mathcal{V}$ appearing in $P$ by the corresponding term $\sigma(\chi)$, but for local rules, which are left unchanged by instantiations, i.e., $R\sigma = R$ for all $R$ and $\sigma$. By $\Sigma$ we denote the set of all the possible instantiations.

**Definition 3.2.2 (Structural Congruence).** *The structural congruence relations $\equiv_S$ and $\equiv_R$ and $\equiv_P$ are the least congruence relations on sequence patterns, local rules and on patterns, respectively, satisfying the rules shown in Figure 3.2.*

Structural congruence rules state the associativity of $\cdot$ and $\mid$, the commutativity of the latter and the neutral role of $\epsilon$. Moreover, the axiom $(SP_1 \cdot SP_2)^{\circlearrowleft} \rfloor P \equiv_P (SP_2 \cdot SP_1)^{\circlearrowleft} \rfloor P$ says that looping sequences can rotate. In the following, for simplicity, we will use $\equiv$ in place of $\equiv_P$.

$$
\begin{gathered}
SP_1 \cdot (SP_2 \cdot SP_3) \equiv_S (SP_1 \cdot SP_2) \cdot SP_3 \qquad SP \cdot \epsilon \equiv_S \epsilon \cdot SP \equiv_S SP \\
SP_1 \equiv_S SP_2 \ \text{ implies } \ SP_1 \equiv_P SP_2 \\
SP_1 \equiv_P SP_2 \ \text{ and } \ P_1 \equiv_P P_2 \ \text{ imply } \ (SP_1)^{\circlearrowleft} \rfloor P_1 \equiv_P (SP_2)^{\circlearrowleft} \rfloor P_2 \\
L_1 \equiv_P L_1' \ \text{ and } L_2 \equiv_P L_2' \ \text{ imply } L_1 \mapsto L_2 \equiv_R L_1' \mapsto L_2' \\
L_1 \equiv_P L'_1 \ \text{ and } L_2 \equiv_P L'_2 \ \text{ and } \ SP_1 \equiv_P SP'_1 \ \text{ and } \ SP_2 \equiv_P SP'_2 \text{ imply} \\
L_1^{\uparrow SP_1} \mapsto L_2^{\uparrow SP_2} \equiv_R {L'}_1^{\uparrow SP'_1} \mapsto {L'}_2^{\uparrow SP'_2} \ \text{ and } \ L_1^{\downarrow SP_1} \mapsto L_2^{\downarrow SP_2} \equiv_R {L'}_1^{\downarrow SP'_1} \mapsto {L'}_2^{\downarrow SP'_2} \\
R_1 \equiv_R R_2 \ \text{ implies } \ R_1 \equiv_P R_2 \\
P_1 \mid P_2 \equiv_P P_2 \mid P_1 \qquad P_1 \mid (P_2 \mid P_3) \equiv_P (P_1 \mid P_2) \mid P_3 \qquad P \mid \epsilon \equiv_P P \\
(\epsilon)^{\circlearrowleft} \rfloor \epsilon \equiv_P \epsilon \qquad (SP_1 \cdot SP_2)^{\circlearrowleft} \rfloor P \equiv_P (SP_2 \cdot SP_1)^{\circlearrowleft} \rfloor P
\end{gathered}
$$

Figure 3.2: Structural Congruence

### 3.2.2 (Parallel) Operational Semantics

In order to define a reduction step in which (possibly) more than one rule is applied, following [25], we first define the application of a single rule, either global or local, using the notion of evaluation context given in Definition 2.2.6.

To enforce the fact that local and global rule applications can be done in parallel, we underline those subterms that are produced by the application of the rule involved. Terms matching the left-hand-side of a (local or global) rule must not have any underlined subterm. Underlined terms are only used for bookkeeping in the definition of rule application.

Let $\underline{\mathcal{T}}$ denote the set of terms in which some subterms can be underlined. The erasing mapping $\eta : \underline{\mathcal{T}} \mapsto \mathcal{T}$ erases all underlining obtaining a term generated by the grammar of Definition 3.2.1.

We first define the application of global or local rules to terms in $\underline{\mathcal{T}}$ which produces terms in $\underline{\mathcal{T}}$.

*Global rules* are of the shape $P_1 \mapsto P_2$. They can be applied to terms only if they occur in a legal evaluation context.

*Local rules* are inside compartments, and can be applied only if a term matching the left-hand-side of the rule occurs in the same compartment.

Notice that global rules have patterns $P$ on both the left and the right-hand side of the rules, whereas local rules have the less general $L$ patterns. In particular, $L$ patterns do not contain compartments, and therefore cannot change the nesting structure of the compartments of a term.

**Definition 3.2.3 (Rule Application).** *Given a finite set of global rules $\mathcal{R}$,* the rule application $\to$ *is the least relation closed with respect to* $\equiv$ *defined by:*

$$\frac{P_1 \mapsto P_2 \in \mathcal{R} \quad \sigma \in \Sigma \quad P_1\sigma \not\equiv \epsilon}{E[P_1\sigma] \to E[\underline{P_2\sigma}]} \text{ (GRT)}$$

$$\frac{\sigma \in \Sigma \quad L_1\sigma \not\equiv \epsilon}{E[L_1 \mapsto L_2 \mid L_1\sigma \mid T] \to E[L_1 \mapsto L_2 \mid \underline{L_2\sigma} \mid T]} \text{ (LR)}$$

$$\frac{\sigma \in \Sigma \quad L_1\sigma \not\equiv \epsilon}{E[(S_1\sigma)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid L_1\sigma \mid T)] \to E[\underline{L_2\sigma} \mid \big(\underline{S_2\sigma}\big)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid T)]} \text{ (LR-Out)}$$

$$\frac{\sigma \in \Sigma \quad L_1\sigma \not\equiv \epsilon}{E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid L_1\sigma \mid (S_1\sigma)^{\circlearrowleft} \rfloor T] \to E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid \big(\underline{S_2\sigma}\big)^{\circlearrowleft} \rfloor (T \mid \underline{L_2\sigma})]} \text{ (LR-In)}$$

With rule (LR-Out) a term or a rule $L_1\sigma$ exits from the membrane in which it is contained only if the membrane has as loop the sequence $S_1\sigma$: when outside, $L_1\sigma$ is transformed into $L_2\sigma$, and the sequence $S_1\sigma$ into $S_2\sigma$. (LR-In) is similar, but it moves terms or rules into local membranes. Local rules do not allow to move, create or delete membranes: only global rules can do that.

Observe that local rules can be dynamically added and deleted both by global and local rules. A global rule which adds the local rule $R$ is of the shape $P \mapsto R$ and a global rule which erases the same local rule is of the shape $R \mapsto P$. A local rule which adds the local rule $R$ is of the shape $L \mapsto R$ and a local rule which erases the same local rule is of the shape $R \mapsto L$. Moreover local rules can add local rules in compartments separated by just one membrane, since they can be of the shapes $L^{\uparrow S} \mapsto R^{\uparrow S'}$ or $L^{\downarrow S} \mapsto R^{\downarrow S'}$. Finally, note that we can derive a sequential version of these rules just by removing the underline from their right-hand side.

A reduction step of the parallel semantics $\Longrightarrow$, starting from a term in $\mathcal{T}$ applies any number of global or local rules that could be performed in parallel, producing a final term in $\mathcal{T}$ (with no underlined subterms).

**Definition 3.2.4 (Parallel Reduction).** *The reduction* $\Longrightarrow$ *between terms in* $\mathcal{T}$ *is defined by:*

$$\frac{T = T_0 \to T_1 \to \cdots \to T_{n+1} \quad n \geq 0 \quad T' = \eta(T_{n+1})}{T \Longrightarrow T'}$$

Note that there is no stopping condition for rule application: we can apply only one rule, and then stop; either we can apply only two rules, and then stop. We can apply any number of local or global rules, until no rule is applicable (i.e it does not exist $T_{n+2}$ such that $T_{n+1} \to T_{n+2}$).

To justify the definition of the reduction $\Longrightarrow$ we have to show that the order in which the local or global rules are applied is not important. The notion of multi-hole context, i.e., of term where some disjoint subterms are replaced by holes, is handy.

**Definition 3.2.5 (Multi-Hole Contexts).** Multi-Hole Contexts $C$ *are defined as:*

$$C ::= \square \quad \big| \quad T \quad \big| \quad C \mid C \quad \big| \quad (\square)^{\circlearrowleft} \rfloor C \quad \big| \quad (S)^{\circlearrowleft} \rfloor C$$

*where* $T \in \mathcal{T}$ *and* $S \in \mathcal{S}$. *We denote by* $\mathcal{C}$ *the infinite set of multi-hole contexts.*

Given a multi-hole context $C \in \mathcal{C}$ containing $n$ holes, with $C[T_1]\dots[T_n]$ we denote the term obtained by replacing each $i$-th hole with the term $T_i$ in $C$. We can show that in a parallel reduction only disjoint subterms can change. Rules (GRT) and (LR) modify just one subterm. Rules (LR-Out) and (LR-In) modify three subterms, i.e., a membrane and a term exiting or entering the membrane, using $\epsilon$ for the missing term.

**Theorem 3.2.6.** *If* $T \Longrightarrow T'$*, then there is a multi-hole context* $C$ *and terms* $T_1, \dots, T_n$*,* $T'_1, \dots, T'_n$ *such that* $T \equiv C[T_1]\dots[T_n]$*,* $T' \equiv C[T'_1]\dots[T'_n]$ *and for all* $1 \leq i \leq n$*:*

- *either* $C[T^*_1]\dots[T_i]\dots[T^*_n] \to C[T^*_1]\dots[T'_i]\dots[T^*_n]$
- *or there are* $i_1, i_2, i_3$ *such that* $i \in \{i_1, i_2, i_3\}$ *and* $C[T^*_1]\dots[T_{i_1}] \mid ([T_{i_2}])^{\circlearrowleft} \rfloor [T_{i_3}]\dots[T^*_n] \to C[T^*_1]\dots[T'_{i_1}] \mid \big([T'_{i_2}]\big)^{\circlearrowleft} \rfloor [T'_{i_3}]\dots[T^*_n]$

*where* $T^*_j$ *can be either* $T_j$ *(subterm of* $T$*) or* $T'_j$ *(subterm of* $T'$*).*

*Proof.* If $T \Longrightarrow T'$, then for some $U_0, \dots, U_{m+1}$ we get $T = U_0 \to \dots \to U_{m+1}$ where $m \geq 0$ and $T' = \eta(U_{m+1})$. We show by induction on $h \leq m$ and by cases on the last applied reduction rule that

- *either* $U_h = C[T^*_1]\dots[T_i]\dots[T^*_n]$ *and* $U_{h+1} = C[T^*_1]\dots[T'_i]\dots[T^*_n]$

- *or there are* $i_1, i_2, i_3$ *such that* $i \in \{i_1, i_2, i_3\}$ *and*
  $U_h = C[T_1^*] \ldots [T_{i_1}] \mid ([T_{i_2}])^{\circlearrowleft} \rfloor [T_{i_3}] \ldots [T_n^*]$,
  $U_{h+1} = C[T_1^*] \ldots [T'_{i_1}] \mid \left([T'_{i_2}]\right)^{\circlearrowleft} \rfloor [T'_{i_3}] \ldots [T_n^*]$

*where "$*$" can be either " " or "$'$" and all terms with $'$ are either underlined or* $\epsilon$.

If the last applied rule is

$$\frac{\sigma \in \Sigma \quad L_1\sigma \not\equiv \epsilon}{E[L_1 \mapsto L_2 \mid L_1\sigma \mid V] \to E[L_1 \mapsto L_2 \mid \underline{L_2\sigma} \mid V]}$$

then $U_h = E[L_1 \mapsto L_2 \mid L_1\sigma \mid V]$ and $U_{h+1} = E[L_1 \mapsto L_2 \mid \underline{L_2\sigma} \mid V]$. By induction $U_h = C[T_1^*] \ldots [T_n^*]$. Since $L_1\sigma$ is a subterm of $T$ and $\underline{L_2\sigma}$ is a subterm of $U_{m+1}$ there must be an index $i$ such that $T_i = L_1\sigma$ and $T'_i = \underline{L_2\sigma}$.

If the last applied rule is

$$\frac{\sigma \in \Sigma \quad L\sigma \not\equiv \epsilon \quad L_1\sigma \in \mathcal{T} \quad S_1\sigma \in \mathcal{T}}{E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid L_1\sigma \mid (S_1\sigma)^{\circlearrowleft} \rfloor V] \to E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid \left(\underline{S_2\sigma}\right)^{\circlearrowleft} \rfloor (V \mid \underline{L_2\sigma})]}$$

then $U_h = E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid L_1\sigma \mid (S_1\sigma)^{\circlearrowleft} \rfloor V]$ and $U_{h+1} = E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid \left(\underline{S_2\sigma}\right)^{\circlearrowleft} \rfloor (V \mid \underline{L_2\sigma})]$. By induction $U_h = C[T_1^*] \ldots [T_n^*]$. Notice that $L_1\sigma, S_1\sigma$ are subterms of $T$ and $\underline{L_2\sigma}, \underline{S_2\sigma}$ are subterm of $U_{m+1}$. Moreover $L_1\sigma$, $S_1\sigma$ and $\epsilon$ in $T$ are replaced by $\epsilon$, $\underline{S_2\sigma}$ and $\underline{L_2\sigma}$ in $U_{m+1}$, respectively. Therefore there must be indexes $i_1, i_2, i_3$ such that $T_{i_1} = L_1\sigma$, $T'_{i_1} = \epsilon$, $T_{i_2} = S_1\sigma$, $T'_{i_2} = \underline{S_2\sigma}$, $T_{i_3} = \epsilon$, $T'_{i_3} = \underline{L_2\sigma}$.

□

**Example 3.2.7 (Mitochondria Running Example: Syntax and Reductions).** *A CLSLR term representing the mitochondria evolution inside the cell's activity discussed in the introduction could be:*

$$\begin{aligned} \mathtt{CELL} = (cell)^{\circlearrowleft} \rfloor ( \quad & \mathtt{NUCLEUS} \mid \mathtt{MITOCH} \mid \ \ldots \ \mid \mathtt{MITOCH} \mid \\ & mRNA \mapsto protein \mid \\ & protein^{\downarrow Tom} \mapsto protein^{\downarrow Tom} \quad ) \end{aligned}$$

*A cell is composed by its membrane (here just represented by the element cell) and its content (in this case, the nucleus, a certain number of mitochondria and a few rules modeling the activity of interest). In particular,* *the two rules above model the steps (3) and (4), respectively, of the example schematised in Section 3.1.*

*Assuming that* DNA *is the sequence of genes representing the cell's DNA, and g is the particular gene (contained in* DNA*) codifying the protein, we define the nucleus of the cell with the CLSLR term:*

$$\begin{aligned} \mathtt{NUCLEUS} = (nucleus)^{\circlearrowleft} \rfloor (\quad & \mathsf{DNA} \mid \\ & \widetilde{x} \cdot g \cdot \widetilde{y} \mapsto (\widetilde{x} \cdot g \cdot \widetilde{y} \mid mRNA) \mid \\ & mRNA^{\uparrow nucleus} \mapsto mRNA^{\uparrow nucleus} \quad ) \end{aligned}$$

*Note that the first of the two rules above models step (1) of our example (DNA transcription into mRNA), while the second one (mRNA exits the nucleus) models step (2).*

*The mitochondria of our model are composed of a membrane, on which we point out the Tom complex, containing an inner membrane (INN_MITOCH) and a couple of rules:*

$$\begin{aligned} \mathtt{MITOCH} = (Tom)^{\circlearrowleft} \rfloor (\quad & \mathtt{INN\_MITOCH} \mid \\ & protein^{\downarrow Tim} \mapsto (Mit_A \mapsto (Mit_A \mid ATP))^{\downarrow Tim} \mid \\ & ATP^{\uparrow \widetilde{x}} \mapsto ATP^{\uparrow \widetilde{x}} \quad ) \end{aligned}$$

*where we denote by the element $Mit_A$ a mitochondrial factor inside the inner membrane (activated by our protein), necessary to produce ATP. In particular, the protein, when in the intermembranous space, is moved through Tim inside the inner mitochondrial space (step (5) of our example) and then transformed into the newly generated rule $Mit_A \mapsto (Mit_A \mid ATP)$ which will lead the production of ATP (step (6) of our example).*

*Finally, in INN_MITOCH we point out the Tim complex:*

$$\begin{aligned} \mathtt{INN\_MITOCH} = (Tim)^{\circlearrowleft} \rfloor (\quad & Mit_A \mid \\ & ATP^{\uparrow \widetilde{x}} \mapsto ATP^{\uparrow \widetilde{x}} \quad ) \end{aligned}$$

*Both in MITOCH and INN_MITOCH we have the rules needed to transport the ATP towards the cell's cytoplasm (steps (7) and (8) of the example).*

*A possible (parallel) reduction of this term, when g initially produces a certain number of mRNA is shown (by focusing only on the more interesting changes) in Figure 3.3. The ATP produced in the last but two reductions in INN_MITOCH moves to MITOCH in the last but one reduction and new ATP is produced in INN_MITOCH. In the last reduction, the firstly generated ATP moves to the cell, the secondly generated ATP moves to MITOCH and new ATP is produced in INN_MITOCH.*

$$\begin{array}{ll}
\Longrightarrow^{+} & (cell)^{\circlearrowleft} \rfloor ((nucleus)^{\circlearrowleft} \rfloor (mRNA \mid \ldots \mid mRNA \mid \ldots) \mid \ldots) \\
\Longrightarrow^{+} & (cell)^{\circlearrowleft} \rfloor (mRNA \mid \ldots \mid mRNA \mid \ldots) \\
\Longrightarrow & (cell)^{\circlearrowleft} \rfloor (protein \mid \ldots \mid protein \mid \ldots) \\
\Longrightarrow & (cell)^{\circlearrowleft} \rfloor ((Tom)^{\circlearrowleft} \rfloor (protein \mid \ldots) \mid \ldots \mid \\
 & \qquad (Tom)^{\circlearrowleft} \rfloor (protein \mid \ldots) \mid \ldots) \\
\Longrightarrow & (cell)^{\circlearrowleft} \rfloor ((Tom)^{\circlearrowleft} \rfloor ((Tim)^{\circlearrowleft} \rfloor (Mit_A \mapsto (Mit_A \mid ATP) \mid \ldots) \mid \ldots) \mid \\
 & \qquad (Tom)^{\circlearrowleft} \rfloor ((Tim)^{\circlearrowleft} \rfloor (Mit_A \mapsto (Mit_A \mid ATP) \mid \ldots) \mid \ldots) \mid \\
 & \qquad \ldots) \\
\Longrightarrow & (cell)^{\circlearrowleft} \rfloor ((Tom)^{\circlearrowleft} \rfloor ((Tim)^{\circlearrowleft} \rfloor (ATP \mid \ldots) \mid \ldots) \mid \\
 & \qquad (Tom)^{\circlearrowleft} \rfloor ((Tim)^{\circlearrowleft} \rfloor (ATP \mid \ldots) \mid \ldots) \mid \ldots) \\
\Longrightarrow & (cell)^{\circlearrowleft} \rfloor ((Tom)^{\circlearrowleft} \rfloor (ATP \mid (Tim)^{\circlearrowleft} \rfloor (ATP \mid \ldots) \mid \ldots) \mid \\
 & \qquad (Tom)^{\circlearrowleft} \rfloor (ATP \mid (Tim)^{\circlearrowleft} \rfloor (ATP \mid \ldots) \mid \ldots) \mid \ldots) \\
\Longrightarrow & (cell)^{\circlearrowleft} \rfloor (ATP \mid \ldots \mid ATP \mid \\
 & \qquad (Tom)^{\circlearrowleft} \rfloor (ATP \mid (Tim)^{\circlearrowleft} \rfloor (ATP \mid \ldots) \mid \ldots) \mid \\
 & \qquad (Tom)^{\circlearrowleft} \rfloor (ATP \mid (Tim)^{\circlearrowleft} \rfloor (ATP \mid \ldots) \mid \ldots) \mid \ldots)
\end{array}$$

Figure 3.3: Mitochondria evolution

## 3.3 Types

In this section we introduce a type system that enforces the fact that compartments must contain rules having specific features. E.g., in [51] the following *rule features* for $L_1 \mapsto L_2$ are suggested:

- the rule is **deleting** if $Vars(L_1) \supset Vars(L_2)$ (denoted by $\mathsf{d}$);
- the rule is **replicating** if some variable in $L_2$ occurs twice (denoted by $\mathsf{r}$);
- the rule is **splitting** if $L_1$ has a subterm containing two different variables (denoted by $\mathsf{s}$);
- the rule is **equating** if some variable in $L_1$ occurs twice (denoted by $\mathsf{e}$).

This kind of features reflects a structure of rewrite features which could be common for rewrite systems in general. Other, model-dependent, features could be defined to reflect peculiarities and properties of the particular model

$$\Delta \vdash_s \epsilon : \emptyset \quad \text{(TSEPS)} \qquad\qquad \Delta, \chi : \varphi \vdash_s \chi : \varphi \quad \text{(TSVAR)}$$

$$\frac{a : \varphi \in \Psi}{\Delta \vdash_s a : \varphi}\ \text{(TSELM)} \qquad\qquad \frac{\Delta \vdash_s SP : \varphi \quad \Delta \vdash_s SP' : \varphi'}{\Delta \vdash_s SP \cdot SP' : \varphi \cup \varphi'}\ \text{(TSSEQ)}$$

Figure 3.4: Typing Rules for Membranes

under investigation. The features of the rules allowed in a compartment are controlled by the wrapping sequence of the compartment. Our *typing system* and the consequent *typed parallel reduction* ensure that, in spite of the facts that reducing a term may move rules in and out of compartments, compartments always contain rules allowed by their wrapping sequence. In addition to the previous features of rules we say that:

- the feature of rule $L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2}$ is that it is an **out rule** (denoted by $\mathsf{o}$);
- the feature of rule $L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2}$ is that it is an **in rule** (denoted by $\mathsf{i}$).

To express the control of the wrapping sequence over the content of the compartment, we associate a subset $\varphi$ of $\{\mathsf{d}, \mathsf{r}, \mathsf{s}, \mathsf{e}, \mathsf{o}, \mathsf{i}\}$ to every element in $\mathcal{A}$. This is called a *membrane type*. We use $\Psi$ to denote a classification of elements. The type assignment in Figure 3.4, where a basis $\Delta$, defined by

$$\Delta \ ::= \ \emptyset \ \ \big| \ \ \Delta, \chi : \varphi$$

assigns membrane types to element and sequence variables, and defines the type of a sequence as the union of the membrane types of its elements.

To define the type of patterns, that may contain parallel (composition) of rules, we consider:

1. the features of the rules, contained in the pattern, and
2. in case there are output rules the type of the rules that are emitted by these output rules.

Therefore a *pattern type*, denoted by $\tau$, is a sequence of membrane types, i.e., $\tau \in \{\varphi\}^*$. By $\emptyset$ we denote the empty sequence. If the parallel composition of local rules $R_1 \mid \cdots \mid R_n$, $n \geq 0$, has type $\varphi \cdot \tau$, then $\varphi$ is the union of the features of the rules $R_i$ $(1 \le i \le n)$, and $\tau$ is the type of the parallel composition of rules in $L'$ for those $L'$ such that $R_i = L^{\uparrow SP} \mapsto L'^{\uparrow SP'}$ for some $i$, $1 \le i \le n$ (the type of the parallel composition of the rules that are emitted). If no rule is emitted, then $\tau = \emptyset$, and $\varphi \cdot \emptyset = \varphi$.

*Union* of pattern types, $\sqcup$, is defined inductively by:

- $\emptyset \sqcup \tau = \tau \sqcup \emptyset = \tau$, and
- $\varphi_1 \cdot \tau_1 \sqcup \varphi_2 \cdot \tau_2 = (\varphi_1 \cup \varphi_2) \cdot (\tau_1 \sqcup \tau_2)$.

and *containment*, $\sqsubseteq$, is defined by:

- $\emptyset \sqsubseteq \tau$
- $\varphi \cdot \tau \sqsubseteq \varphi' \cdot \tau'$ if $\varphi \subseteq \varphi'$ and $\tau \sqsubseteq \tau'$.

The judgment $\Delta \vdash_p P : \tau$, defined in Figure 3.5, asserts that the pattern $P$ is *well formed* and has pattern type $\tau$, assuming the basis $\Delta$, which assigns membrane types to element and sequence variables and pattern types to term variables. The judgment $\Delta \vdash_{gr} P_1 \mapsto P_2 : \mathtt{ok}$ in last rule defines well-formedness of global rules.

It is easy to verify that the typing rules in Figures 3.4 and 3.5 enjoy weakening, i.e., if $\Delta \subseteq \Delta'$ then $\Delta \vdash_s SP : \varphi$ implies $\Delta' \vdash_s SP : \varphi$, $\Delta \vdash_p P : \tau$ implies $\Delta' \vdash_p P : \tau$, and $\Delta \vdash_{gr} P_1 \mapsto P_2 : \mathtt{ok}$ implies $\Delta' \vdash_{gr} P_1 \mapsto P_2 : \mathtt{ok}$.

Rule (Tvar) asserts that a term variable is well typed when its pattern type is found in the basis. Rule (Tseq) asserts that, since a sequence does not contain rules, its pattern type is empty. Rule (TRloc) asserts that the type of a local rule $R = L_1 \mapsto L_2$ is the union of the set of features of the rule $R$, denoted by $\mathsf{features}(R)$, and the pattern type of its right-hand-side $L_2$. This is because once the rule is applied, an instance of the pattern $L_2$ will substitute the instance of its left-hand-side $L_1$. Rule (TRlocOut) checks that the features of rules allowed by the membrane $S_2$ include the one allowed by $S_1$, so that if the compartment was well formed before applying $L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2}$, it will be well formed afterwards (when $S_2$ replaces $S_1$). Moreover, the pattern type of the rule is $\{\mathtt{o}\}$, concatenated with the pattern type of $L_2$, since $L_2$ is the pattern sent outside the compartment. Rule (TRlocIn) checks rule $L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2}$. Since the pattern $L_2$ will get into a compartment with membrane $S_1$, the membrane $S_2$, that replaces $S_1$, must allow all the features of rules that were allowed before, and moreover, it allows the features of the rules in $L_2$. The type is $\{\mathtt{i}\}$ union the type of the patterns that are emitted by the out rules contained

$$\Delta, X:\tau \vdash_p X:\tau \quad \text{(Tvar)} \qquad\qquad \Delta \vdash_p SP:\emptyset \quad \text{(Tseq)}$$

$$\frac{\Delta \vdash_p L_2:\tau}{\Delta \vdash_p L_1 \mapsto L_2 : \mathsf{features}(L_1 \mapsto L_2) \sqcup \tau} \text{ (TRloc)}$$

$$\frac{\Delta \vdash_p L_2:\tau \quad \Delta \vdash_s S_1:\varphi_1 \quad \Delta \vdash_s S_2:\varphi_2 \quad \varphi_1 \subseteq \varphi_2}{\Delta \vdash_p L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} : \{\mathsf{o}\} \cdot \tau} \text{ (TRlocOut)}$$

$$\frac{\Delta \vdash_p L_2:\varphi\ \cdot\tau \quad \Delta \vdash_s S_1:\varphi_1 \quad \Delta \vdash_s S_2:\varphi_2 \quad \varphi \cup \varphi_1 \subseteq \varphi_2}{\Delta \vdash_p L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} : \{\mathsf{i}\} \sqcup \tau} \text{ (TRlocIn)}$$

$$\frac{\Delta \vdash_p P:\tau \quad \Delta \vdash_p P':\tau'}{\Delta \vdash_p P \mid P' : \tau \sqcup \tau'} \text{ (Tpar)}$$

$$\frac{\Delta \vdash_s SP:\varphi \quad \Delta \vdash_p P:\varphi' \cdot \tau' \quad \varphi' \subseteq \varphi}{\Delta \vdash_p (SP)^{\circlearrowleft} \rfloor P : \tau'} \text{ (Tcomp)}$$

$$\frac{\Delta \vdash_p P_1:\tau_1 \quad \Delta \vdash_p P_2:\tau_2 \quad \tau_2 \sqsubseteq \tau_1}{\Delta \vdash_{gr} P_1 \mapsto P_2 : \mathtt{ok}} \text{ (TRglob)}$$

Figure 3.5: Typing Rules for Patterns and Global Rules

in $L_2$. Rule (Tpar) enforces the fact that the patterns in parallel are both well formed and the final pattern type is the union of the two pattern types. Rule (Tcomp) checks that a compartment contain only rules whose features are allowed by its wrapping sequence. The pattern type of the compartment is the type of the rules that are emitted. Finally, rule (TRGlob) says that the global rule $P_1 \mapsto P_2$ is well formed in case the pattern $P_2$ that will replace $P_1$ has less features, so that it is allowed by all the compartments in which $P_1$ is allowed.

As we can see from rule (TRLoc) the type system is independent from the specific set of features considered. Any syntactic characterization of rules could be considered for a feature.

**Definition 3.3.1.** *An instantiation* $\sigma$ agrees *with a basis* $\Delta$ *(notation* $\sigma \in \Sigma_\Delta$*) if* $x:\varphi \in \Delta$ *implies* $\vdash_s \sigma(x):\varphi$*, and* $\widetilde{x}:\varphi \in \Delta$ *implies* $\vdash_s \sigma(\widetilde{x}):\varphi$*, and*

$X : \tau \in \Delta$ *implies* $\vdash_p \sigma(X) : \tau$.

This is sound since the judgments $\vdash_s$ use assumptions on element and sequence variables, while the judgments $\vdash_p$ use assumptions on term variables. Based on the previous typing system we define a *typed semantics*, that preserves well-formedness of terms.

**Definition 3.3.2 (Typed Rule Application).** *Given a finite set of global rules $\mathcal{R}$, the* typed rule application $\rightarrow_{\mathsf{T}}$ *is the least relation closed with respect to $\equiv$ and satisfying the following rules:*

$$\frac{\mathcal{R}_\Delta = \{P_1 \mapsto P_2 \in \mathcal{R} \mid \Delta \vdash_{gr} P_1 \mapsto P_2 : \mathtt{ok}\} \quad P_1 \mapsto P_2 \in \mathcal{R}_\Delta \quad \sigma \in \Sigma_\Delta \quad P_1\sigma \not\equiv \epsilon}{E[P_1\sigma] \rightarrow_{\mathsf{T}} E[\underline{P_2\sigma}]} \text{(T-GRT)}$$

$$\frac{\sigma \in \Sigma_\Delta \quad L_1\sigma \not\equiv \epsilon}{E[L_1 \mapsto L_2 \mid L_1\sigma] \rightarrow_{\mathsf{T}} E[L_1 \mapsto L_2 \mid \underline{L_2\sigma}]} \text{(T-LR)}$$

$$\frac{\sigma \in \Sigma_\Delta \quad L_1\sigma \not\equiv \epsilon}{E[(S_1\sigma)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid L_1\sigma \mid T)] \rightarrow_{\mathsf{T}} E[\underline{L_2\sigma} \mid \left(\underline{S_2\sigma}\right)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid T)]} \text{(T-LR-Out)}$$

$$\frac{\sigma \in \Sigma_\Delta \quad L_1\sigma \not\equiv \epsilon}{E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid L_1\sigma \mid (S_1\sigma)^{\circlearrowleft} \rfloor T] \rightarrow_{\mathsf{T}} E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid \left(\underline{S_2\sigma}\right)^{\circlearrowleft} \rfloor (T \mid \underline{L_2\sigma})]} \text{(T-LR-In)}$$

**Definition 3.3.3 (Typed Parallel Reduction).** *The reduction $\Longrightarrow_{\mathsf{T}}$ between term in $\mathcal{T}$ is defined by:*

$$\frac{T = T_0 \rightarrow_{\mathsf{T}} T_1 \rightarrow_{\mathsf{T}} \cdots \rightarrow_{\mathsf{T}} T_{n+1} \quad n \geq 0 \quad T' = \eta(T_{n+1})}{T \Longrightarrow_{\mathsf{T}} T'}$$

The property enforced by the type system is that well-typed terms reduce to well-typed terms: the proof follows a series of preparatory lemmas.

**Lemma 3.3.4 (Inversion Lemma).** *Inversion of the typing relations:*

1. *If $\Delta \vdash_s \epsilon : \varphi$, then $\varphi = \emptyset$.*

2. *If $\Delta \vdash_s \chi : \varphi$, then $\chi : \varphi \in \Delta$.*

3. *If $\Delta \vdash_s a : \varphi$, then $a : \varphi \in \Psi$.*

4. *If $\Delta \vdash_s SP \cdot SP' : \varphi$, then $\Delta \vdash_s SP : \varphi_1$, $\Delta \vdash_s SP' : \varphi_2$ and $\varphi = \varphi_1 \sqcup \varphi_2$.*

5. *If* $\Delta \vdash_p X : \tau$, *then* $X : \tau \in \Delta$.

6. *If* $\Delta \vdash_p SP : \tau$, *then* $\tau = \emptyset$.

7. *If* $\Delta \vdash_p L_1 \mapsto L_2 : \tau$, *then* $\tau = \mathsf{features}(L_1 \mapsto L_2) \sqcup \tau'$ *and* $\Delta \vdash_p L_2 : \tau'$.

8. *If* $\Delta \vdash_p L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} : \tau$, *then* $\tau = \{\mathsf{o}\} \cdot \tau'$, $\Delta \vdash_p L_2 : \tau'$, $\Delta \vdash_s S_1 : \varphi_1$, $\Delta \vdash_s S_2 : \varphi_2$ *and* $\varphi_1 \subseteq \varphi_2$.

9. *If* $\Delta \vdash_p L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} : \tau$, *then* $\tau = \{\mathsf{i}\} \sqcup \tau'$, $\Delta \vdash_p L_2 : \varphi \cdot \tau'$, $\Delta \vdash_s S_1 : \varphi_1$, $\Delta \vdash_s S_2 : \varphi_2$ *and* $\varphi \cup \varphi_1 \subseteq \varphi_2$.

10. *If* $\Delta \vdash_p P \mid P' : \tau$, *then* $\tau = \tau_1 \sqcup \tau_2$, $\Delta \vdash_p P : \tau_1$ *and* $\Delta \vdash_p P' : \tau_2$.

11. *If* $\Delta \vdash_p (SP)^{\circlearrowleft} \rfloor P : \tau$, *then* $\Delta \vdash_s SP : \varphi$, $\Delta \vdash_p P : \varphi' \cdot \tau$ *and* $\varphi' \subseteq \varphi$.

12. *If* $\Delta \vdash_{gr} P_1 \mapsto P_2 : \mathtt{ok}$, *then* $\Delta \vdash_p P_1 : \tau_1$, $\Delta \vdash_p P_2 : \tau_2$ *and* $\tau_2 \sqsubseteq \tau_1$.

*Proof.* Immediate from the typing rules in Figures 3.4 and 3.5. □

**Lemma 3.3.5.** *If* $\Delta \vdash_p E[P] : \tau$ *then*

1. $\Delta \vdash_p P : \tau_0$ *for some* $\tau_0$, *and*

2. *if* $P'$ *is such that* $\Delta \vdash_p P' : \tau'$ *with* $\tau' \sqsubseteq \tau_0$, *then* $\Delta \vdash_p E[P'] : \tau''$ *with* $\tau'' \sqsubseteq \tau$.

*Proof.* By induction on the definition of contexts.

- If $E = \square$, then $E[P] = P$, and so $\Delta \vdash_p P : \tau$. Since in this case $E[P'] = P'$, and $\Delta \vdash_p P' : \tau'$ with $\tau' \sqsubseteq \tau$ by hypothesis, then $\Delta \vdash_p E[P'] : \tau'$.

- If $E = E' \mid T$, then $E[P] = E'[P] \mid T$. From Lemma 3.3.4(10) we derive $\Delta \vdash_p E'[P] : \tau_1$ and $\Delta \vdash_p T : \tau_2$, with $\tau_1 \sqcup \tau_2 = \tau$. By induction hypothesis on $E'[P]$ we get $\Delta \vdash_p P : \tau_0$ and $\Delta \vdash_p E'[P'] : \tau_1'$ with $\tau_1' \sqsubseteq \tau_1$. Applying rule (Tpar) we conclude $\Delta \vdash_p E[P'] : \tau''$ with $\tau'' = \tau_1' \sqcup \tau_2$, and then $\tau'' \sqsubseteq \tau$.

- If $E = (S)^{\circlearrowleft} \rfloor E'$, then $E[P] = (S)^{\circlearrowleft} \rfloor E'[P]$. From Lemma 3.3.4(11) we derive $\Delta \vdash_p S : \varphi_0$, and $\Delta \vdash_p E'[P] : \varphi \cdot \tau$, with $\varphi \subseteq \varphi_0$. By induction hypothesis on $E'[P]$ we get $\Delta \vdash_p P : \tau_0$, and $\Delta \vdash_p E'[P'] : \varphi' \cdot \tau'$ with $\varphi' \cdot \tau' \sqsubseteq \varphi \cdot \tau$. Applying rule (Tcomp) we conclude $\Delta \vdash_p E[P'] : \tau'$, with $\tau' \sqsubseteq \tau$.

□

**Lemma 3.3.6.** *If $\sigma \in \Sigma_\Delta$, then $\vdash_s SP\sigma : \varphi$ if and only if $\Delta \vdash_s SP : \varphi$.*

*Proof.* ($\Leftarrow$) By induction on $\Delta \vdash_s SP : \varphi$. Consider the last applied rule.

- If the rule is (TSvar), the proof follows from $\sigma \in \Sigma_\Delta$. For rules (TSeps) and (TSelm), the fact that $SP$ is a term implies that $SP\sigma = SP$, and, moreover, it is typable from the empty environment.
- Rule (TSseq). In this case $SP = SP_1 \cdot SP_2$, and from Lemma 3.3.4(4) we derive $\Delta \vdash_s SP_1 : \varphi_1$, $\Delta \vdash_s SP_2 : \varphi_2$, and $\varphi = \varphi_1 \cup \varphi_2$. By induction hypotheses on $SP_1$ and $SP_2$ we get $\vdash_s SP_1\sigma : \varphi_1$ and $\vdash_s SP_2\sigma : \varphi_2$. Therefore, since $SP_1\sigma \cdot SP_2\sigma = (SP_1 \cdot SP_2)\sigma$, applying the rule (TSseq) we conclude $\vdash_s (SP_1 \cdot SP_2)\sigma : \varphi$.

($\Rightarrow$) By induction on $SP$.

- If $SP = \chi$, the proof follows from $\sigma \in \Sigma_\Delta$. If $SP = \epsilon$ or $SP = a$ we use weakening.
- Let $SP$ be $SP_1 \cdot SP_2$. Since $(SP_1 \cdot SP_2)\sigma = SP_1\sigma \cdot SP_2\sigma$, from Lemma 3.3.4(4) we derive $\varphi = \varphi_1 \cup \varphi_2$, $\vdash_s SP_1\sigma : \varphi_1$, and $\vdash_s SP_2\sigma : \varphi_2$. By induction hypotheses we get $\Delta \vdash_s SP_1 : \varphi_1$, and $\Delta \vdash_s SP_2 : \varphi_2$. Applying rule $(TSseq)$ we conclude $\Delta \vdash_s SP_1 \cdot SP_2 : \varphi$.

□

**Lemma 3.3.7.** *If $\sigma \in \Sigma_\Delta$, then $\vdash_p P\sigma : \tau$ if and only if $\Delta \vdash_p P : \tau$.*

*Proof.* ($\Leftarrow$) By induction on $\Delta \vdash_p P : \tau$. Consider the last applied rule.

- If the rule is (Tvar), the proof follows from $\sigma \in \Sigma_\Delta$. For rules (TRloc), (TRlocOut), (TRlocIn) the fact that $P$ is a rule implies that $P\sigma = P$ and, moreover, it is typable from the empty environment. For rule (Tseq) if $P$ is a sequence pattern then also $P\sigma$ is a sequence pattern, and then we can apply rule (Tseq) with the empty environment.
- If the rule is (Tpar), then $P = P_1 \mid P_2$, and from Lemma 3.3.4(10) we derive $\Delta \vdash_p P_1 : \tau_1$, $\Delta \vdash_p P_2 : \tau_2$, and $\tau = \tau_1 \sqcup \tau_2$. By induction hypotheses on $P_1$ and $P_2$ we get $\vdash_p P_1\sigma : \tau_1$, and $\vdash_p P_2\sigma : \tau_2$. Therefore, since $P_1\sigma \mid P_2\sigma = (P_1 \mid P_2)\sigma$, applying the rule (Tpar) we conclude $\vdash_p (P_1 \mid P_2)\sigma : \tau$.

- If the rule is (Tcomp), then the proof is similar using Lemmas 3.3.4(11) and 3.3.6 for the first premise.

($\Rightarrow$) By induction on $P$.

- If $P = X$, the proof follows from $\sigma \in \Sigma_\Delta$. If $P$ is a sequence pattern, then also $P\sigma$ is a sequence pattern, and we can apply the rule (Tseq). If $P$ is a rule, then $P = P\sigma$.
- Let $P$ be $P = P_1 \mid P_2$. Since $(P_1 \mid P_2)\sigma = P_1\sigma \mid P_2\sigma$, and the fact that $\vdash_p (P_1 \mid P_2)\sigma : \tau$, from Lemma 3.3.4(10) we derive $\vdash_p P_1\sigma : \tau_1$, $\vdash_p P_2\sigma : \tau_2$, and $\tau = \tau_1 \sqcup \tau_2$. By induction hypotheses on $P_1$ and $P_2$ we get $\Delta \vdash_p P_1 : \tau_1$ and $\Delta \vdash_p P_2 : \tau_2$. Applying rule (Tpar) we conclude $\Delta \vdash_p (P_1 \mid P_2) : \tau$.
- If $P = (SP)^{\circlearrowleft} \rfloor P'$ the proof is similar using Lemmas 3.3.4(11) and 3.3.6 for the first premise.

$\square$

**Theorem 3.3.8 (Subject Reduction).** *If $\vdash_p T : \tau$ and $T \Longrightarrow_{\mathsf{T}} T'$, then $\vdash_p T' : \tau'$ for some $\tau' \sqsubseteq \tau$.*

*Proof.* By cases on the reduction rules.

**Rule (TGR)**

From Definition 3.3.2, $T = E[P_1\sigma]$, $T' = E[P_2\sigma]$, and $\sigma \in \Sigma_\Delta$. By hypothesis $\vdash_p T : \tau$ and $\Delta \vdash_{gr} P_1 \mapsto P_2 : \mathtt{ok}$. Therefore, Lemma 3.3.5(1) implies $\vdash_p P_1\sigma : \tau_1$ for some $\tau_1$, and from Lemma 3.3.7 we derive $\Delta \vdash_p P_1 : \tau_1$. From $\Delta \vdash_{gr} P_1 \mapsto P_2 : \mathtt{ok}$, Lemma 3.3.4(12) implies $\Delta \vdash_p P_2 : \tau_2$ with $\tau_2 \sqsubseteq \tau_1$. We can apply Lemma 3.3.7 obtaining $\vdash_p P_2\sigma : \tau_2$. From Lemma 3.3.5(2) we conclude $\Delta \vdash_p E[P_2\sigma] : \tau'$ for some $\tau' \sqsubseteq \tau$.

**Rule (TLR)**

From Definition 3.3.2, $T = E[L_1 \mapsto L_2 \mid L_1\sigma]$, $T' = E[L_1 \mapsto L_2 \mid L_2\sigma]$, and $\sigma \in \Sigma_\Delta$. By hypothesis $\vdash_p T : \tau$. Therefore, Lemma 3.3.5(1) implies $\vdash_p L_1 \mapsto L_2 \mid L_1\sigma : \tau_0$ for some $\tau_0$. Since $\sigma \in \Sigma_\Delta$, Lemma 3.3.7 implies $\Delta \vdash_p L_1 \mapsto L_2 \mid L_1 : \tau_0$. From Lemma 3.3.4(10) we derive $\Delta \vdash_p L_1 \mapsto L_2 : \tau_0'$, and $\Delta \vdash_p L_1 : \tau_1$ with $\tau_0' \sqcup \tau_1 = \tau_0$. By Lemma 3.3.4(7) we derive $\Delta \vdash_p L_2 : \tau_2$ with $\tau_2 \sqsubseteq \tau_1$. Lemma 3.3.7 implies $\vdash_p L_2\sigma : \tau_2$, then from (Tpar) we derive $\vdash_p L_1 \mapsto L_2 \mid L_2\sigma : \tau_3$ with $\tau_3 = \tau_0' \sqcup \tau_2$. Since $\tau_3 \sqsubseteq \tau_0$ we can apply Lemma 3.3.5(2) obtaining $\vdash_p E[L_1 \mapsto L_2 \mid L_2\sigma] : \tau'$ with $\tau' \sqsubseteq \tau$.

**Rule** (LR-Out)

From Definition 3.3.2, $T = E[(S_1\sigma)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid L_1\sigma \mid T_0)]$, $T' = E[L_2\sigma \mid (S_2\sigma)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid T_0)]$, and $\sigma \in \Sigma_\Delta$. By hypothesis $\vdash_p T : \tau$. Lemma 3.3.5(1) implies $\vdash_p (S_1\sigma)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid L_1\sigma \mid T_0) : \tau_0$, and, since $\sigma \in \Sigma_\Delta$, Lemma 3.3.7 implies $\Delta \vdash_p (S_1)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid L_1 \mid T_0) : \tau_0$. By Lemma 3.3.4(11) we get $\Delta \vdash_s S_1 : \varphi_1$, and $\Delta \vdash_p L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid L_1 \mid T_0 : \varphi_0 \cdot \tau_0$ where $\varphi_0 \subseteq \varphi_1$. By Lemma 3.3.4(10) we have $\Delta \vdash_p T_0 : \varphi \cdot \tau_1$, and $\Delta \vdash_p L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} : \varphi' \cdot \tau_2$ for some $\varphi \cdot \tau_1 \sqcup \varphi' \cdot \tau_2 \sqsubseteq \varphi_0 \cdot \tau_0$. Lemma 3.3.4(8) implies $\varphi' = \{\mathsf{o}\}$, $\Delta \vdash_p L_2 : \tau_2$ and $\Delta \vdash_s S_2 : \varphi_2$ where $\varphi_1 \subseteq \varphi_2$. Since $\sigma \in \Sigma_\Delta$, Lemma 3.3.7 implies $\vdash_p L_2\sigma : \tau_2$, $\vdash_s S_2\sigma : \varphi_2$, $\Delta \vdash_p L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} : \{\mathsf{o}\}\cdot\tau_2$, and $\vdash_p T_0 : \varphi \cdot \tau_1$. Using these premises, we apply rule (Tpar) deriving $\vdash_p L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid T_0 : \{\mathsf{o}\} \cdot \tau_2 \sqcup \varphi \cdot \tau_1$, and then $\vdash_p (S_2\sigma)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid T_0) : \tau_1 \sqcup \tau_2$ by rule (Tcomp). Finally $\vdash_p L_2\sigma \mid (S_2\sigma)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid T_0) : \tau_1 \sqcup \tau_2$ by rule (Tpar): since $\tau_1 \sqcup \tau_2 \sqsubseteq \tau_0$, we can apply the Lemma 3.3.5(2), obtaining $\vdash_p E[L_2\sigma \mid (S_2\sigma)^{\circlearrowleft} \rfloor (L_1^{\uparrow S_1} \mapsto L_2^{\uparrow S_2} \mid T_0)] : \tau'$ with $\tau' \sqsubseteq \tau$.

**Rule** (LR-In)

From Definition 3.3.2, $T = E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid L_1\sigma \mid (S_1\sigma)^{\circlearrowleft} \rfloor T_0]$, $T' = E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid (S_2\sigma)^{\circlearrowleft} \rfloor (T_0 \mid L_2\sigma)]$, and $\sigma \in \Sigma_\Delta$. By hypothesis $\vdash_p T : \tau$. Lemma 3.3.5(1) implies $\vdash_p L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid L_1\sigma \mid (S_1\sigma)^{\circlearrowleft} \rfloor T_0 : \tau_0$, and, since $\sigma \in \Sigma_\Delta$, Lemma 3.3.7 implies $\Delta \vdash_p L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid L_1 \mid (S_1)^{\circlearrowleft} \rfloor T_0 : \tau_0$. By Lemma 3.3.4(10) we have $\Delta \vdash_p L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} : \tau_1$ and $\Delta \vdash_p (S_1)^{\circlearrowleft} \rfloor T_0 : \tau_2$ for some $\tau_1 \sqcup \tau_2 \sqsubseteq \tau_0$. Lemma 3.3.4(9) implies $\tau_1 = \{\mathsf{i}\} \sqcup \tau_3$, $\Delta \vdash_s S_1 : \varphi_1$, $\Delta \vdash_s S_2 : \varphi_2$, and $\Delta \vdash_p L_2 : \varphi \cdot \tau_3$, where $\varphi \cup \varphi_1 \subseteq \varphi_2$. By Lemma 3.3.4(11) $\Delta \vdash_p T_0 : \varphi' \cdot \tau_2$ for some $\varphi' \subseteq \varphi_1$. Since $\sigma \in \Sigma_\Delta$, Lemma 3.3.7 implies $\vdash_s S_1\sigma : \varphi_1$, $\vdash_s S_2\sigma : \varphi_2$, $\vdash_p L_2\sigma : \varphi \cdot \tau_3$, $\vdash_p L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} : \{\mathsf{i}\} \sqcup \tau_3$, and $\vdash_p T_0 : \varphi' \cdot \tau_2$. Using these premises, we apply rule (Tpar) deriving $\vdash_p L_2\sigma \mid T_0 : \varphi \cdot \tau_3 \sqcup \varphi' \cdot \tau_2$, and then $\vdash_p (S_2\sigma)^{\circlearrowleft} \rfloor L_2\sigma \mid T_0 : \tau_3 \sqcup \tau_2$ by rule (Tcomp). Finally $\vdash_p L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid (S_2\sigma)^{\circlearrowleft} \rfloor L_2\sigma \mid T_0 : \tau_1 \sqcup \tau_2$, because $\tau_1 = \{\mathsf{i}\} \sqcup \tau_3$, by rule (Tpar): since $\tau_1 \sqcup \tau_2 \sqsubseteq \tau_0$, we can apply the Lemma 3.3.5(2) obtaining $\vdash_p E[L_1^{\downarrow S_1} \mapsto L_2^{\downarrow S_2} \mid (S_2\sigma)^{\circlearrowleft} \rfloor L_2\sigma \mid T_0] : \tau'$ for some $\tau' \sqsubseteq \tau$. □

**Example 3.3.9 (Mitochondria Running Example: Typing).** *Let the rules used in Example 3.2.7 be labeled by:*

- $R_g = \widetilde{x} \cdot g \cdot \widetilde{y} \mapsto (\widetilde{x} \cdot g \cdot \widetilde{y} \mid mRNA)$,

- $R_m = mRNA \mapsto protein$,
- $R_{o\downarrow} = protein^{\downarrow Tom} \mapsto protein^{\downarrow Tom}$,
- $R_{m\uparrow} = mRNA^{\uparrow nucleus} \mapsto mRNA^{\uparrow nucleus}$,
- $R_{i\downarrow} = protein^{\downarrow Tim} \mapsto {R_a}^{\downarrow Tim}$,
- $R_a = Mit_A \mapsto (Mit_A \mid ATP)$
- $R_{a\uparrow} = ATP^{\uparrow\widetilde{x}} \mapsto ATP^{\uparrow\widetilde{x}}$.

*Let* $\varphi_g = \mathsf{features}(R_g)$, $\varphi_m = \mathsf{features}(R_m)$, $\varphi_a = \mathsf{features}(R_a)$. *The term representing our model can be typed if* $\Psi$ *contains appropriate membrane types for the elements which occur in the membranes, i.e.:*

$$\{cell : \varphi_{cell}, nucleus : \varphi_{nucleus}, Tom : \varphi_{Tom}, Tim : \varphi_{Tim}\} \subseteq \Psi$$

*where* $\{\mathsf{i}\} \cup \varphi_m \subseteq \varphi_{cell}$, $\{\mathsf{o}\} \cup \varphi_g \subseteq \varphi_{nucleus}$, $\{\mathsf{o}, \mathsf{i}\} \subseteq \varphi_{Tom}$, *and* $\{\mathsf{o}\} \cup \varphi_a \in \varphi_{Tim}$. *In this case the given parallel reduction is also a typed parallel reduction for this term.*

*We can type the* ***MITOCH*** *and* ***INN_MITOCH*** *with the derivations in Figure3.6.*

## 3.4 Conclusions

In rewrite system models, the term (describing the systems under consideration) and the list of rules (describing the system's evolution) could be considered as separate. Here, a calculus with global (separate from the system) and local (dynamic and system intrinsic) rewrite rules is presented. While global rules can, as usual, be applied anywhere in a given term, local rules can only be applied in the compartment on which they are defined. Local rules are equipped with dynamic features: they can be created, moved and erased.

Some of the ideas used in CLSLR are not completely new, but can be found in other formalisms. For example, the use of variables and contexts for rule application is a key feature of $\kappa$-calculus, even if it does not deal with membranes. Moreover, both approaches emphasize the key rule of the surface components, in proteins ($\kappa$-calculus) or membranes (CLSLR), for biological modeling.

$$\dfrac{\dfrac{\dfrac{\Delta \vdash_p Mit_A \mid ATP : \emptyset}{\Delta \vdash_p R_a : \varphi_a}\,(\textsc{TRLoc}) \quad \Delta \vdash_s Tim : \varphi_{Tim} \quad \varphi_a \subseteq \varphi_{Tim}}{\Delta \vdash_p R_{i\downarrow} : \{\mathsf{i}\}}\,(\textsc{TRLocIn}) \qquad \Delta \vdash_p R_{a\uparrow} : \{\mathsf{o}\}}{\Delta \vdash_p R_{i\downarrow} \mid R_{a\uparrow} : \{\mathsf{i},\mathsf{o}\}}\,(\textsc{TPar})$$

$$\dfrac{\Delta \vdash_s Tom : \varphi_{Tom} \qquad \dfrac{\Delta \vdash_p \texttt{INN\_MITOCH} : \emptyset \quad \Delta \vdash_p R_{i\downarrow} \mid R_{a\uparrow} : \{\mathsf{i},\mathsf{o}\}}{\Delta \vdash_p \texttt{INN\_MITOCH} \mid R_{i\downarrow} \mid R_{a\uparrow} : \{\mathsf{i},\mathsf{o}\}}\,(\textsc{TPar}) \qquad \{\mathsf{i},\mathsf{o}\} \subseteq \varphi_{Tom}}{\Delta \vdash_p (Tom)^{\circlearrowleft} \rfloor (\texttt{INN\_MITOCH} \mid R_{i\downarrow} \mid R_{a\uparrow}) : \emptyset}\,(\textsc{TComp})$$

$$\dfrac{\Delta \vdash_s Tim : \varphi_{Tim} \qquad \dfrac{\Delta \vdash_p Mit_A : \emptyset \qquad \dfrac{\Delta \vdash_p ATP : \emptyset \quad \Delta \vdash_s \widetilde{x} : \varphi_{Tom} \quad \varphi_{Tom} \subseteq \varphi_{Tom}}{\Delta \vdash_p R_{a\uparrow} : \{\mathsf{o}\}}\,(\textsc{TRLocOut})}{\Delta \vdash_p Mit_A \mid R_{a\uparrow} : \{\mathsf{o}\}}\,(\textsc{TPar}) \qquad \{\mathsf{o}\} \subseteq \varphi_{Tim}}{\Delta \vdash_p (Tim)^{\circlearrowleft} \rfloor (Mit_A \mid R_{a\uparrow}) : \emptyset}\,(\textsc{TComp})$$

Figure 3.6: Type derivations for `MITOCH` and `INN_MITOCH`.

More similarities can be found with P Systems. Locality and intrinsic parallelism of rules are also present in our approach, but in CLSLR the level of parallelism is not necessarily maximal, and moreover not only molecules but also rules can be created, deleted or moved across membranes. In both approaches the local rules cannot describe some possible biological behaviors such as fusion, deletion or creation of membranes. As it happens for P Systems, local rules are intrinsically parallel. Indeed, expressing rules that are local to well delimited compartments, and with the possibility to define systems with multiple, parallel, compartments, naturally leads to the definition of a parallel semantics. In addition, CLSLR allows to describe activities involving membranes, such as joining, division or fusion, and non-standard activities trough global rules.

As a future work, in the lines of [22, 39, 18], we plan to investigate how to adapt this model with a quantitative semantics, also studying the limits and constraints imposed by parallel semantics.

# Chapter 4

# A Type Discipline for Required and Excluded Elements

## 4.1 Introduction

In chemistry, hydrophobicity is the physical property of a molecule (known as a hydrophobe) that is repelled from a mass of water. Hydrophobic molecules tend to be non-polar and thus prefer other neutral molecules and non-polar solvents. Hydrophobic molecules in water often cluster together forming micelles. From the other perspective, water on hydrophobic surfaces will exhibit a high contact angle (thus causing, for example, the familiar dew drops on a hydrophobic leaf surface). Examples of hydrophobic molecules include the alkanes, oils, fats, and greasy substances in general. Hydrophobic materials are used for oil removal from water, the management of oil spills, and chemical separation processes to remove non-polar from polar compounds. Hydrophobicity is just an example of repellency in Biochemistry. Other well-known examples may be found in the behavior of anions and cations, or at a different level of abstraction, in the behavior of the rh antigen for the different blood types. As a counterpart, there may be elements, in nature, which always require the presence of other elements: for example, it is difficult to find a lonely atom of oxygen, they always appear in the pair $O_2$.

In this chapter, based on [21], we bring these aspects at their maximum limit, and, by abstracting away all the phenomena which give rise/arise to/from repellency (and its counterpart), we assume that for each kind of element of our reality we are able to fix a set of elements which are required by the element for its existence, and a set of elements whose presence is forbidden by the element. We enrich the basic CLS presented in Chapter 2 with a type discipline which guarantees the soundness of reduction rules with respect to some relevant properties of biological systems deriving from the required and excluded kinds of elements. The key technical tool we use is to associate to each reduction rule the minimal set of conditions an instantiation must satisfy in order to assure that applying this rule to a "correct" system we get a "correct" system as well. We also propose a type inference algorithm and show its soundness and completeness.

The required/excluded elements properties modeled here assure, through type inference, that only compatible elements are combined together in the different environments of the biological system took in consideration. Thus the type system intrinsically yields a notion of correct (well-behaving) system according to the expressed requirements.

## 4.2 Type Discipline

We classify elements in $\mathcal{A}$ with *basic types*. Intuitively, given a molecule represented by an element in $\mathcal{A}$, we associate to it a type $\mathtt{t}$ which specifies the kind of the molecule. We assume a fixed typing $\Psi$ for the elements in $\mathcal{A}$.

For each basic type $\mathtt{t}$ we assume to have a pair of sets of basic types $(\mathtt{R_t}, \mathtt{E_t})$, where $\mathtt{t} \notin \mathtt{R_t} \cup \mathtt{E_t}$ and $\mathtt{R_t} \cap \mathtt{E_t} = \emptyset$, saying that the presence of elements of basic type $\mathtt{t}$ requires the presence of elements whose basic type is in $\mathtt{R_t}$, and forbids the presence of elements whose basic type is in $\mathtt{E_t}$. We consider only *local* properties: elements influence each other if they are either present in the same compartment or one is present in the looping sequence and the other one is present in the inner compartment of a containment operator.

The type system derives the set of types of a pattern (and therefore also a term), checking that the constraints imposed by the required and excluded sets are satisfied. Types are pairs $(\mathtt{P}, \mathtt{R})$: $\mathtt{P}$ is the set of basic types of elements that are *present* in the top-level compartment of the term, and $\mathtt{R}$ is the set of basic types of elements that are *required* to fulfill the requirements of the elements present in the top-level compartment of the term. The set of *excluded* elements for a given set $\mathtt{P}$ of present elements is given by $\mathtt{E_P} = \bigcup_{\mathtt{t} \in \mathtt{P}} \mathtt{E_t}$. Given a basic type $\mathtt{t}$, the type $(\{\mathtt{t}\}, \mathtt{R_t})$ is the type of a compartment containing only elements of type $\mathtt{t}$, and it is well formed.

A type $(\mathtt{P},\mathtt{R})$ is *well formed* if the required types $\mathtt{R}$ are required by the present elements $\mathtt{P}$, and the constraints on required and excluded elements are not contradictory [1].

**Definition 4.2.1 (Well-Formed Types).** *A type* $(\mathtt{P},\mathtt{R})$ *is* well formed *if* $\mathtt{R} \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}} \mathtt{R}_{\mathtt{t}}$*, and* $\mathtt{P}\cap\mathtt{E}_{\mathtt{P}} = \mathtt{P}\cap\mathtt{R} = \mathtt{R}\cap\mathtt{E}_{\mathtt{P}} = \emptyset$*.*

Pairs of well-formed types can be combined, to obtain a new type.

**Definition 4.2.2 (Join of Types).** *Given two well-formed types* $(\mathtt{P},\mathtt{R})$ *and* $(\mathtt{P}',\mathtt{R}')$ *we define their* join $(\mathtt{P},\mathtt{R})\sqcup(\mathtt{P}',\mathtt{R}')$ *by*

$$(\mathtt{P},\mathtt{R})\sqcup(\mathtt{P}',\mathtt{R}') = (\mathtt{P}\cup\mathtt{P}', (\mathtt{R}\cup\mathtt{R}')\setminus(\mathtt{P}\cup\mathtt{P}')).$$

The type obtained through join could be not well formed. In order to ensure the well-formedness of the result, we ask the compatibility of the joined types. A pair of types is *compatible* if required and excluded types are compatible with the union of their present types.

**Definition 4.2.3 (Type Compatibility).** *Well-formed types* $(\mathtt{P},\mathtt{R})$ *and* $(\mathtt{P}',\mathtt{R}')$ *are* compatible *(written* $(\mathtt{P},\mathtt{R}) \bowtie (\mathtt{P}',\mathtt{R}')$*) if* $\mathtt{E}_{\mathtt{P}}\cap\mathtt{P}' = \mathtt{E}_{\mathtt{P}}\cap\mathtt{R}' = \emptyset$*, and* $\mathtt{E}_{\mathtt{P}'}\cap\mathtt{P} = \mathtt{E}_{\mathtt{P}'}\cap\mathtt{R} = \emptyset$*.*

Bases are defined by:

$$\Delta \ ::= \ \emptyset \quad \Big| \quad \Delta, x : (\{\mathtt{t}\},\mathtt{R}_{\mathtt{t}}) \quad \Big| \quad \Delta, \rho : (\mathtt{P},\mathtt{R})$$

where $\rho$ denotes a sequence or term variable, i.e. $\rho \in \mathcal{TV}\cup\mathcal{SV}$. A basis $\Delta$ is *well formed* if all types in the basis are well formed. We check the safety of terms, sequences and more generally patterns using the typing rules of Figure 4.1. It is easy to verify that, if we start from well-formed bases, then in a derivation we produce only well-formed bases and well-formed types. Note that terms and sequences are typable from the empty basis.

All the rules are trivial except for the last one, $(\mathsf{Tcomp})$. According to this rule, we can put a pattern $P$ inside a containment operator with looping sequence $SP$ only if all the types required from $P$ are provided by $SP$. This is because the elements present in the inner compartment can not interact with the elements present outside the looping sequence.

It is easy to verify that the type system of Figure 4.1 enjoys weakening, i.e. that $\Delta \vdash P : (\mathtt{P},\mathtt{R})$ and $\Delta \subseteq \Delta'$ imply $\Delta' \vdash P : (\mathtt{P},\mathtt{R})$.

The following substitution properties will be handy.

[1]A requirement we could have asked is the symmetry of repellency, that is: for all $\mathtt{t},\mathtt{t}'$, if $\mathtt{t}\in\mathtt{E}_{\mathtt{t}'}$, then $\mathtt{t}'\in\mathtt{E}_{\mathtt{t}}$. However, such a requirement would not change the type system, since compatibility of types encompasses this property.

$$\Delta \vdash \epsilon : (\emptyset, \emptyset) \quad (\mathsf{Teps}) \qquad \frac{a : \mathtt{t} \in \Psi}{\Delta \vdash a : (\{\mathtt{t}\}, \mathtt{R}_{\mathtt{t}})} \ (\mathsf{Tel})$$

$$\Delta, \chi : (\mathtt{P}, \mathtt{R}) \vdash \chi : (\mathtt{P}, \mathtt{R}) \quad (\mathsf{Tvar})$$

$$\frac{\Delta \vdash SP : (\mathtt{P}, \mathtt{R}) \quad \Delta \vdash SP' : (\mathtt{P}', \mathtt{R}') \quad (\mathtt{P}, \mathtt{R}) \bowtie (\mathtt{P}', \mathtt{R}')}{\Delta \vdash SP \cdot SP' : (\mathtt{P}, \mathtt{R}) \sqcup (\mathtt{P}', \mathtt{R}')} \ (\mathsf{Tseq})$$

$$\frac{\Delta \vdash P : (\mathtt{P}, \mathtt{R}) \quad \Delta \vdash P' : (\mathtt{P}', \mathtt{R}') \quad (\mathtt{P}, \mathtt{R}) \bowtie (\mathtt{P}', \mathtt{R}')}{\Delta \vdash P \mid P' : (\mathtt{P}, \mathtt{R}) \sqcup (\mathtt{P}', \mathtt{R}')} \ (\mathsf{Tpar})$$

$$\frac{\Delta \vdash SP : (\mathtt{P}, \mathtt{R}) \quad \Delta \vdash P : (\mathtt{P}', \mathtt{R}') \quad (\mathtt{P}, \mathtt{R}) \bowtie (\mathtt{P}', \mathtt{R}') \text{ and } \mathtt{R}' \subseteq \mathtt{P}}{\Delta \vdash (SP)^{\circlearrowleft} \rfloor P : (\mathtt{P}, \mathtt{R} \setminus \mathtt{P}')} \ (\mathsf{Tcomp})$$

Figure 4.1: Typing Rules

**Lemma 4.2.4.** *If* $\Delta \vdash E[P] : (\mathtt{P}, \mathtt{R})$ *then*

1. $\Delta \vdash P : (\mathtt{P}', \mathtt{R}')$ *for some* $(\mathtt{P}', \mathtt{R}')$*, and*
2. $\Delta, X : (\mathtt{P}', \mathtt{R}') \vdash E[X] : (\mathtt{P}, \mathtt{R})$*, and*
3. *if* $P'$ *is such that* $\Delta \vdash P' : (\mathtt{P}', \mathtt{R}')$*, then* $\Delta \vdash E[P'] : (\mathtt{P}, \mathtt{R})$.

*Proof.* Easy by induction on the definition of evaluation contexts. □

We are interested in applying reduction rules only to correct terms, whose type is well formed and whose requirements are completely satisfied. More formally:

**Definition 4.2.5 (Correct Terms).** *A term* $T$ *is* correct *if* $\vdash T : (\mathtt{P}, \emptyset)$ *for some* $\mathtt{P}$.

**Example 4.2.6.** *Assuming* $\mathcal{A} = \{a, b, c, d\}$ $\quad \Psi = \{a : \mathtt{t}_a, b : \mathtt{t}_b, c : \mathtt{t}_c, d : \mathtt{t}_d\}$ $\mathtt{R}_b = \{\mathtt{t}_c\}$ $\quad \mathtt{R}_c = \{\mathtt{t}_a\}$ $\quad \mathtt{E}_d = \{\mathtt{t}_b, \mathtt{t}_c\}$ $\quad \mathtt{R}_a = \mathtt{R}_d = \mathtt{E}_a = \mathtt{E}_b = \mathtt{E}_c = \emptyset$ *and using the rules in Figure 4.1, the terms in lines* $(*)$ *and* $(**)$ *of Example 2.2.8 have type* $(\{\mathtt{t}_a, \mathtt{t}_d\}, \emptyset)$*, so they are correct terms. However, the term in line* $(***)$*, does not have a type, since the element* $b$ *is in the same compartment of the element* $d$*, but the basic type of* $b$ *is in the set of the elements excluded by the presence of the basic type of* $d$.

Rules such that the left-hand-side and the right-hand-side patterns have the same type do not change the type of terms to which they are applied.

**Definition 4.2.7 ($\Delta$-safe rules).** *A rewrite rule $P_1 \mapsto P_2$ is* $\Delta$-safe *if $\Delta \vdash P_1 : (\mathtt{P},\mathtt{R})$ and $\Delta \vdash P_2 : (\mathtt{P},\mathtt{R})$ for some $(\mathtt{P},\mathtt{R})$.*

When we apply a $\Delta$-safe rule to a term we need to choose an instantiation which agrees with $\Delta$, i.e. $\sigma$ must replace the variables in the domain of $\Delta$ as prescribed by $\Delta$, as seen in Definition 3.3.1. More formally:

**Definition 4.2.8.** *An instantiation $\sigma$* agrees *with a basis $\Delta$ (notation $\sigma \in \Sigma_\Delta$) if $\chi : (\mathtt{P},\mathtt{R}) \in \Delta$ implies $\vdash \sigma(\chi) : (\mathtt{P},\mathtt{R})$.*

Agreement between substitutions and basis assures type preservation, as proved in the following lemma.

**Lemma 4.2.9.** *If $\sigma \in \Sigma_\Delta$, then $\vdash P\sigma : (\mathtt{P},\mathtt{R})$ if and only if $\Delta \vdash P : (\mathtt{P},\mathtt{R})$.*

*Proof.* ($\Leftarrow$) By induction on $\Delta \vdash P : (\mathtt{P},\mathtt{R})$. Consider the last applied rule.

- If the rule is (Tvar), the proof follows from $\sigma \in \Sigma_\Delta$. For rules (Teps), (Tel) the fact that $P$ is a term implies that $P\sigma = P$ and, moreover, it is typable from the empty environment.
- Rule (Tseq). In this case $P = SP{\cdot}SP'$, $(\mathtt{P},\mathtt{R}) = (\mathtt{P}'',\mathtt{R}'') \sqcup (\mathtt{P}',\mathtt{R}')$, $\Delta \vdash SP : (\mathtt{P}'',\mathtt{R}'')$, $\Delta \vdash SP' : (\mathtt{P}',\mathtt{R}')$ and $(\mathtt{P}'',\mathtt{R}'') \bowtie (\mathtt{P}',\mathtt{R}')$. By induction hypotheses, $\vdash SP\,\sigma : (\mathtt{P}'',\mathtt{R}'')$ and $\vdash SP'\,\sigma : (\mathtt{P}',\mathtt{R}')$. Therefore, since $SP\,\sigma{\cdot}SP'\,\sigma = (SP{\cdot}SP')\,\sigma$, applying rule (Tseq) we conclude $\vdash (SP{\cdot}SP')\,\sigma : (\mathtt{P},\mathtt{R})$.
- For rules (Tpar), (Tcomp) the proof is similar.

($\Rightarrow$) By induction on $P$.

- If $P = \chi$, the proof follows from $\sigma \in \Sigma_\Delta$. If $P = \epsilon$, or $P = a$, by weakening.
- Let $P$ be $SP \cdot SP'$. Since $(SP \cdot SP')\,\sigma = SP\,\sigma \cdot SP'\,\sigma$, the fact that $\vdash (SP{\cdot}SP')\,\sigma : (\mathtt{P},\mathtt{R})$ implies that the last applied rule must be (Tseq). Therefore, $(\mathtt{P},\mathtt{R}) = (\mathtt{P}'',\mathtt{R}'') \sqcup (\mathtt{P}',\mathtt{R}')$, $(\mathtt{P}'',\mathtt{R}'') \bowtie (\mathtt{P}',\mathtt{R}')$, $\vdash SP\,\sigma : (\mathtt{P}'',\mathtt{R}'')$, and $\vdash SP'\,\sigma : (\mathtt{P}',\mathtt{R}')$. By induction hypothesis on $SP$ and $SP'$ we get $\Delta \vdash SP : (\mathtt{P}'',\mathtt{R}'')$, and $\Delta \vdash SP' : (\mathtt{P}',\mathtt{R}')$. Applying rule (Tseq) we conclude $\Delta \vdash SP{\cdot}SP' : (\mathtt{P},\mathtt{R}) \sqcup (\mathtt{P}',\mathtt{R}')$.

- If $P = P' \mid P''$ or $P = (SP)^{\circlearrowleft} \rfloor P'$ the proof is similar.

$\square$

Since $\Delta$-safe rules do not modify the type of a term, they cannot create or delete elements in the term. Moreover, also movements of elements between membranes are very limited.

**Example 4.2.10.** *Assuming the sets in Example 4.2.6 and the basis*

$$\Delta = \{\widetilde{x} : (\{\mathtt{t}_b, \mathtt{t}_c\}, \emptyset), Y : (\emptyset, \emptyset)\}$$

*the rule in Example 2.2.5 is a $\Delta$-safe rule, because the left and the right side of the rule have the same type:*

$$\Delta \vdash (a \cdot \widetilde{x})^{\circlearrowleft} \rfloor (b \mid Y) : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset) \quad \Delta \vdash b \mid (a \cdot \widetilde{x})^{\circlearrowleft} \rfloor Y : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$$

*On the other side, using the basis*

$$\Delta' = \{\widetilde{x} : (\emptyset, \emptyset), Y : (\{\mathtt{t}_a\}, \emptyset)\}$$

*the same rule is not a $\Delta'$-safe rule, because left-hand-side and right-hand-side of the rule do not have the same type:*

$$\Delta' \vdash (a \cdot \widetilde{x})^{\circlearrowleft} \rfloor (b \mid Y) : (\{\mathtt{t}_a\}, \emptyset) \quad \Delta' \vdash b \mid (a \cdot \widetilde{x})^{\circlearrowleft} \rfloor Y : (\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$$

In order to allow the application of rules that may introduce/remove/-move elements preserving safety, we introduce a restriction on rules based on the context of application rather than, as for $\Delta$-safety, the type of patterns involved in the rule. To do this, we first characterize evaluation contexts by the types of terms that may fill their hole, and then rules by the types of terms that their application produces.

**Definition 4.2.11 (Typed Holes).** *Given an evaluation context $E$, and a well-formed type* $(\mathtt{P}, \mathtt{R})$*, the type* $(\mathtt{P}, \mathtt{R})$ *is* OK *for the context $E$ if $X : (\mathtt{P}, \mathtt{R}) \vdash E[X] : (\mathtt{P}', \emptyset)$ for some $\mathtt{P}'$.*

The above notion guarantees that filling an evaluation context with a term whose type is OK for the context we obtain a correct term: note that there may be more than one type $(\mathtt{P}, \mathtt{R})$ such that $(\mathtt{P}, \mathtt{R})$ is OK for the context $E$.

We can classify reduction rules according to the types we can derive for the right hand sides of the rules.

**Definition 4.2.12 ($\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rules).** *A rewrite rule $P_1 \mapsto P_2$ is* $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe *if $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$.*

To ensure correctness, a rule can be applied to a typed term only if the instance of the pattern on its right-hand-side has a type that is OK for the context. Note that, contrarily to $\Delta$-safe rules, this kind of rules can create or delete elements. On the other hand, at every application of this kind of rules we must check if the type of its right-hand-side is OK for the context.

**Example 4.2.13.** *Assuming the sets in Example 4.2.6 and the basis*

$$\Delta = \{\widetilde{x} : (\emptyset, \emptyset), Y : (\emptyset, \emptyset)\}$$

*the rule in Example 2.2.5 is* $\Delta$-$(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$*-safe, because its right side has type* $(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$*.*
*Let* $E_1$ *be the evaluation context* $(a \cdot c) \mid \Box$*, the type* $(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$ *is OK for* $E_1$*, since*

$$X : (\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\}) \vdash E_1[X] : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$$

*and so we can apply the rule in Example 2.2.5.*

*Instead, the type* $(\{\mathtt{t}_a, \mathtt{t}_b\}, \emptyset)$ *is not OK for the context* $E_2 = a \mid \Box$*, since*

$$X : (\{\mathtt{t}_a, \mathtt{t}_b\}, \emptyset) \vdash E_2[X] : (\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$$

*and so we cannot apply the rule in Example 2.2.5.*

Since both $\Delta$-safe and $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rules preserve correctness, our semantics uses both.

**Definition 4.2.14 (Typed Semantics).** *Given a finite set of rewrite rules* $\mathcal{R}$*, the* typed semantics *of CLS is the least relation closed with respect to* $\equiv$ *satisfying the following (sets of) rules:*

$$\frac{\mathfrak{R} = P_1 \mapsto P_2 \in \mathcal{R} \text{ is a } \Delta\text{-safe rule} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma_\Delta \quad E \in \mathcal{E}}{E[P_1\sigma] \Longrightarrow E[P_2\sigma]} \;(\mathfrak{R}\text{-}\Delta)$$

$$\frac{\begin{array}{c}\mathfrak{R} = P_1 \mapsto P_2 \in \mathcal{R} \text{ is a } \Delta\text{-}(\mathtt{P}, \mathtt{R})\text{-safe rule} \qquad P_1\sigma \not\equiv \epsilon \\ \sigma \in \Sigma_\Delta \qquad E \in \mathcal{E} \qquad (\mathtt{P}, \mathtt{R}) \text{ is OK for } E\end{array}}{E[P_1\sigma] \Longrightarrow E[P_2\sigma]} \;(\mathfrak{R}\text{-}\Delta\text{-}(\mathtt{P}, \mathtt{R}))$$

Reduction preserves correctness, as proved in the following theorem.

**Theorem 4.2.15 (Subject Reduction).** *If* $\vdash T : (\mathtt{P}, \emptyset)$ *and* $T \Longrightarrow T'$*, then* $\vdash T' : (\mathtt{P}', \emptyset)$ *for some* $\mathtt{P}'$*.*

*Proof.* We analyze the two sets of rules of the semantics separately. Let $\mathfrak{R} = P_1 \mapsto P_2$.

| $\Delta_{(i)}$ | type of $\mathtt{P}_1\sigma$ | type of $\mathtt{P}_2\sigma$ | type of $E[\mathtt{P}_2\sigma]$ |
|---|---|---|---|
| $\widetilde{x} : (\{\mathtt{t}_b, \mathtt{t}_c\}, \{\mathtt{t}_a\})\ Y : (\emptyset, \emptyset)$ | $(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$ | $(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$ | $(\{\mathtt{t}_a, \mathtt{t}_d\}, \emptyset)$ |
| $\widetilde{x} : (\{\mathtt{t}_c\}, \{\mathtt{t}_a\})\ Y : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$ | $(\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset)$ | $(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$ | $(\{\mathtt{t}_a, \mathtt{t}_d\}, \emptyset)$ |
| $\widetilde{x} : (\emptyset, \emptyset)\ Y : (\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset)$ | $(\{\mathtt{t}_a\}, \emptyset)$ | $(\{\mathtt{t}_a, \mathtt{t}_b\}, \emptyset)$ | — |

Figure 4.2: Basis and Typings of Example 4.2.16

**Rules** ($\Re$-$\Delta$) From Definition 4.2.14, $T = E[P_1\sigma]$ and $T' = E[P_2\sigma]$ and $\sigma \in \Sigma_\Delta$. By hypothesis $\vdash E[P_1\sigma] : (\mathtt{P}, \emptyset)$. Therefore, Lemma 4.2.4.(1) implies $\vdash P_1\sigma : (\mathtt{P}', \mathtt{R}')$ for some $(\mathtt{P}', \mathtt{R}')$, and from Lemma 4.2.9 we derive $\Delta \vdash P_1 : (\mathtt{P}', \mathtt{R}')$. By Definition 4.2.7, we get that $\Delta \vdash P_2 : (\mathtt{P}', \mathtt{R}')$. Applying Lemma 4.2.9, we derive that $\vdash P_2\sigma : (\mathtt{P}', \mathtt{R}')$. Finally, from Lemma 4.2.4.(3) we obtain $\vdash E[P_2\sigma] : (\mathtt{P}, \emptyset)$.

**Rules** ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$) From Definition 4.2.12, we have that $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$. Lemma 4.2.9 and $\sigma \in \Sigma_\Delta$ imply that $\vdash P_2\sigma : (\mathtt{P}, \mathtt{R})$. Since, from Definition 4.2.14, $(\mathtt{P}, \mathtt{R})$ is OK for $E$, we get that $X : (\mathtt{P}, \mathtt{R}) \vdash E[X] : (\mathtt{P}', \emptyset)$ for some $\mathtt{P}'$. Therefore, by Lemma 4.2.4.(3) we conclude that $\vdash E[P_2\sigma] : (\mathtt{P}', \emptyset)$.

$\square$

**Example 4.2.16.** *Using the sets in Example 4.2.6 we can study the behavior of the term in Example 2.2.8. That is the evolution of the initial term due to the application of the* ($\Re$-$\Delta$) *and* ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$) *rules where* $\Re = P_1 \mapsto P_2$, *with*

- $P_1 = (a \cdot \widetilde{x})^{\circlearrowleft} \rfloor (b \mid Y)$, *and*
- $P_2 = b \mid (a \cdot \widetilde{x})^{\circlearrowleft} \rfloor Y$

*and* $\Delta_{(1)}$ *is* $\Delta$ *of the first line in Fig. 4.2, etc. Rule* $P_1 \mapsto P_2$ *is a* $\Delta_{(1)}$-*safe rule, since* $\Delta_{(1)} \vdash P_1 : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$, *and* $\Delta_{(1)} \vdash P_2 : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$. *Therefore, applying rule* ($\Re$-$\Delta$) *we get*

$$d \mid (a)^{\circlearrowleft} \rfloor ((a \cdot c)^{\circlearrowleft} \rfloor ((a \cdot b \cdot c)^{\circlearrowleft} \rfloor (b))) \Longrightarrow$$
$$d \mid (a)^{\circlearrowleft} \rfloor ((a \cdot c)^{\circlearrowleft} \rfloor (b \mid (a \cdot b \cdot c)^{\circlearrowleft} \rfloor \epsilon))$$

*the reduction in line* $(*)$ *of Example 4.2.6.*

*The rule* $P_1 \mapsto P_2$ *is not a* $\Delta_{(2)}$-*safe rule, since* $\Delta_{(2)} \vdash P_1 : (\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset)$, *and* $\Delta_{(2)} \vdash P_2 : (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$. *However,* $P_1 \mapsto P_2$ *is a* $\Delta_{(2)}$-$(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$-*safe rule and the evaluation context of the reduction* $E_{(2)}$, *in the second line of Fig. 2.2, is OK for* $(\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset)$. *So, applying rule* ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$) *we get*

$$d \mid (a)^{\circlearrowleft} \rfloor ((a \cdot c)^{\circlearrowleft} \rfloor (b \mid (a \cdot b \cdot c)^{\circlearrowleft} \rfloor \epsilon)) \Longrightarrow$$
$$d \mid (a)^{\circlearrowleft} \rfloor (b \mid (a \cdot c)^{\circlearrowleft} \rfloor ((a \cdot b \cdot c)^{\circlearrowleft} \rfloor \epsilon))$$

*the reduction in line* $(**)$ *of Example 4.2.6.*
*Finally, for the third reduction neither of the conditions holds.*

- *Firstly,* $P_1 \mapsto P_2$ *is not* $\Delta_{(3)}$*-safe, since* $\Delta_{(3)} \vdash P_1 : (\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset)$ *and* $\Delta_{(3)} \vdash P_2 : (\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$.
- *Secondly, even though* $P_1 \mapsto P_2$ *is* $\Delta_{(3)}$*-*$(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$*-safe, the context* $E_{(3)} = d \mid \Box$ *is not OK for* $(\{\mathtt{t}_a, \mathtt{t}_b\}, \{\mathtt{t}_c\})$.

*Indeed the term in line* $(***)$ *of Example 4.2.6 cannot be typed.*

It is possible to prove that given $\Re = P_1 \mapsto P_2$ if the requirements for applying rule ($\Re$-$\Delta$) are satisfied, then also the requirements for applying rule ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$) are satisfied.

**Theorem 4.2.17.** *If* $P_1 \mapsto P_2$ *is a* $\Delta$*-safe rule and* $P_1\sigma \not\equiv \epsilon$ *and* $\sigma \in \Sigma_\Delta$ *and* $E \in \mathcal{E}$ *and* $\vdash E[P_1\sigma] : (\mathtt{P}', \emptyset)$*, then there is a type* $(\mathtt{P}, \mathtt{R})$ *OK for* $E$ *such that* $P_1 \mapsto P_2$ *is a* $\Delta$*-*$(\mathtt{P}, \mathtt{R})$*-safe rule.*

*Proof.* From the hypothesis that $P_1 \mapsto P_2$ is a $\Delta$-safe rule, and Definition 4.2.7 we have that $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$, and $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$. Therefore, from Definition 4.2.12, $P_1 \mapsto P_2$ is a $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rule. From $\sigma \in \Sigma_\Delta$, $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$ and Lemma 4.2.9 we derive that $\vdash P_1\sigma : (\mathtt{P}, \mathtt{R})$. The hypothesis $\vdash E[P_1\sigma] : (\mathtt{P}', \emptyset)$ and Lemma 4.2.4.(2) imply that $X : (\mathtt{P}, \mathtt{R}) \vdash E[X] : (\mathtt{P}', \emptyset)$, and so $(\mathtt{P}, \mathtt{R})$ is OK for $E$. $\Box$

Given a set of rewrite rules $\mathcal{R}$, the previous theorem proves that if a term is reducible with the typed semantics whose reductions use only ($\Re$-$\Delta$) rules ($\Re \in \mathcal{R}$), then the term is also reducible with the typed semantics whose reductions use only ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$) rules ($\Re \in \mathcal{R}$). Example 4.2.16 shows that the vice versa is not true. Moreover, a term reducible with the typed semantics whose reductions use both sets of rules ($\Re$-$\Delta$) and ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$) is also reducible with the typed semantics whose reductions use only rules ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$). The advantage to have both sets of rules is that checking the applicability of a rule $\Re = P_1 \mapsto P_2$ to a well-typed term using rules ($\Re$-$\Delta$) is more efficient than using ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$) rules. In fact, for both kinds of rules, after having derived the substitution $\sigma \in \Sigma_\Delta$ from the matching of the pattern $P_1$ with the term, we need to derive $\Delta \vdash P_1\sigma : (\mathtt{P}_1, \mathtt{R}_1)$, and

$\Delta \vdash P_2\sigma : (\mathtt{P}_2, \mathtt{R}_2)$. Afterwards, for rules $(\Re\text{-}\Delta)$ we have simply to check if $P_2\,\sigma$ has the same type as $P_1\,\sigma$, whereas for rules $(\Re\text{-}\Delta\text{-}(\mathtt{P},\mathtt{R}))$, in addition to find the type of $P_2\,\sigma$, we have to check whether this type is OK for the evolution context, and so we have to derive the type of the context. Taking advantage from these considerations, in the following section we will show how to use type inference to provide an algorithm for the typed semantics of Definition 4.2.14.

## 4.3 Type Inference

The definition of typed semantics (Definition 4.2.14) is not effective, since we do not know how to choose $\Delta$ for $(\Re\text{-}\Delta)$ rules and $\Delta, \mathtt{P}, \mathtt{R}$ for $(\Re\text{-}\Delta\text{-}(\mathtt{P},\mathtt{R}))$ rules. In the present section we define inference rules for principal typing [67] in order to derive which rules are $\Delta$-safe and which ones are $\Delta$-$(\mathtt{P},\mathtt{R})$-safe, where the choices of $\Delta, \mathtt{P}, \mathtt{R}$ are guided by the term we want to reduce. This will allow us to get an algorithm for checking the applicability of reduction rules to typed terms preserving typing.

We convene that for each variable $x \in \mathcal{X}$ there is an *e-type variable* $\eta_x$ ranging over basic types, and for each variable $\rho \in \mathcal{TV} \cup \mathcal{SV}$ there are two variables $\pi_\rho$, $\phi_\rho$ (called *p-type variable* and *r-type variable*) ranging over sets of basic types. Moreover we convene that $\Pi$ ranges over formal unions and differences of sets of basic types, e-type variables and p-type variables, and $\Phi$ ranges over formal unions and differences of sets of basic types and r-type variables. We denote by $\mu$ a generic p-type, r-type or e-type variable.

A *basis scheme* $\Theta$ is a mapping from atomic variables to their e-type variables, and from sequence and term variables to pairs of their p-type variables and r-type variables:

$$\Theta \ ::= \ \emptyset \quad \Big| \quad \Theta, x : \eta_x \quad \Big| \quad \Theta, \rho : (\pi_\rho, \phi_\rho).$$

The rules for inferring principal typings use judgments of the shape:

$$\vdash P : \Theta; (\Pi, \Phi); \Xi$$

where $\Theta$ is the *principal basis* in which $P$ is well formed, $(\Pi, \Phi)$ is the *principal type* of $P$, and $\Xi$ is the set of constraints that should be satisfied. Figure 4.3 gives these inference rules.

$$\vdash \epsilon : \emptyset; (\emptyset, \emptyset); \emptyset \quad \text{(Reps)} \qquad \frac{a : \mathtt{t} \in \Psi}{\vdash a : \emptyset; (\{\mathtt{t}\}, \mathtt{R}_{\mathtt{t}}); \emptyset} \text{(Rel)}$$

$$\vdash x : \{x : (\{\eta_x\}, \mathtt{R}_{\eta_x})\}; (\{\eta_x\}, \mathtt{R}_{\eta_x}); \emptyset \quad (\mathsf{Rvar}_1)$$

$$\vdash \rho : \{\rho : (\pi_\rho, \phi_\rho)\}; (\pi_\rho, \phi_\rho); \emptyset \quad (\mathsf{Rvar}_2)$$

$$\frac{\vdash SP : \Theta; (\Pi, \Phi); \Xi \qquad \vdash SP' : \Theta'; (\Pi', \Phi'); \Xi'}{\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Pi, \Phi) \sqcup (\Pi', \Phi'); \Xi \cup \Xi' \cup \{(\Pi, \Phi) \bowtie (\Pi', \Phi')\}} \text{(Rseq)}$$

$$\frac{\vdash P : \Theta; (\Pi, \Phi); \Xi \qquad \vdash P' : \Theta'; (\Pi', \Phi'); \Xi'}{\vdash P \mid P' : \Theta \cup \Theta'; (\Pi, \Phi) \sqcup (\Pi', \Phi'); \Xi \cup \Xi' \cup \{(\Pi, \Phi) \bowtie (\Pi', \Phi')\}} \text{(Rpar)}$$

$$\frac{\vdash SP : \Theta; (\Pi, \Phi); \Xi \qquad \vdash P : \Theta'; (\Pi', \Phi'); \Xi'}{\vdash (SP)^{\circlearrowleft} \rfloor P : \Theta \cup \Theta'; (\Pi, \Phi \setminus \Pi'); \Xi \cup \Xi' \cup \{(\Pi, \Phi) \bowtie (\Pi', \Phi'), \Phi' \subseteq \Pi\}} \text{(Rcomp)}$$

Figure 4.3: Inference Rules

**Example 4.3.1.** *We can use the inference rules in Figure 4.3 to infer the types of the patterns of the rule in Example 2.2.5, where, again, we assume the basic types of Example 4.2.6, obtaining*

$$\begin{array}{l} \vdash P_1 : \Theta; (\{\mathtt{t}_a\} \cup \pi_{\widetilde{x}}, \phi_{\widetilde{x}} \setminus (\{\mathtt{t}_b\} \cup \pi_Y)); \Xi_1 \\ \vdash P_2 : \Theta; (\{\mathtt{t}_a, \mathtt{t}_b\} \cup \pi_{\widetilde{x}}, \{\mathtt{t}_c\} \cup (\phi_{\widetilde{x}} \setminus \pi_Y)); \Xi_2 \end{array}$$

*where*

$$\begin{array}{lcl} \Theta & = & \{\ \widetilde{x} : (\pi_{\widetilde{x}}, \phi_{\widetilde{x}}), \quad X : (\pi_Y, \phi_Y)\ \} \\ \Xi_1 & = & \{\ (\{\mathtt{t}_a\}, \emptyset) \bowtie (\pi_{\widetilde{x}}, \phi_{\widetilde{x}}), \quad (\{\mathtt{t}_a\} \cup \pi_{\widetilde{x}}, \phi_{\widetilde{x}}) \bowtie (\{\mathtt{t}_b\} \cup \pi_Y, \{\mathtt{t}_c\} \cup \phi_Y), \\ & & (\{\mathtt{t}_b\}, \{\mathtt{t}_c\}) \bowtie (\pi_Y, \phi_Y), \quad \{\mathtt{t}_c\} \cup \phi_Y \subseteq \{\mathtt{t}_a\} \cup \pi_{\widetilde{x}}\ \} \\ \Xi_2 & = & \{\ (\{\mathtt{t}_a\}, \emptyset) \bowtie (\pi_{\widetilde{x}}, \phi_{\widetilde{x}}), \quad (\{\mathtt{t}_a\} \cup \pi_{\widetilde{x}}, \phi_{\widetilde{x}}) \bowtie (\pi_Y, \phi_Y), \\ & & \phi_Y \subseteq \{\mathtt{t}_a\} \cup \pi_{\widetilde{x}}, \quad (\{\mathtt{t}_b\}, \{\mathtt{t}_c\}) \bowtie (\{\mathtt{t}_a\} \cup \pi_{\widetilde{x}}, \phi_{\widetilde{x}} \setminus \pi_Y)\ \} \end{array}$$

Soundness and completeness of our inference rules can be stated as usual. A *type mapping* maps e-type variables to basic types, p-type variables and r-type variables to sets of basic types. A type mapping $m$ *satisfies* a set of constraints $\Xi$ if all constraints in $m(\Xi)$ are satisfied.

**Theorem 4.3.2 (Soundness of Type Inference).** *If* $\vdash P : \Theta; (\Pi, \Phi); \Xi$ *and* $m$ *is a type mapping which satisfies* $\Xi$*, then* $m(\Theta) \vdash P : (m(\Pi), m(\Phi))$.

*Proof.* By induction on derivations, and by cases on the last applied rule.

- For rules (Reps), (Rel), (Rvar$_1$), and (Rvar$_2$) the result is trivial.
- Rule (Rseq). In this case the conclusion of the rule is $\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Pi, \Phi) \sqcup (\Pi', \Phi'); \Xi \cup \Xi' \cup \{(\Pi, \Phi) \bowtie (\Pi', \Phi')\}$, and the assumptions are $\vdash SP : \Theta; (\Pi, \Phi); \Xi$ and $\vdash SP' : \Theta'; (\Pi', \Phi'); \Xi'$. Since $m$ satisfies $\Xi$ and $\Xi'$, by induction hypothesis, and weakening, we derive that $m(\Theta \cup \Theta') \vdash SP : (m(\Pi), m(\Phi))$ and $m(\Theta \cup \Theta') \vdash SP' : (m(\Pi'), m(\Phi'))$. Moreover, since $m$ satisfies $\{(\Pi, \Phi) \bowtie (\Pi', \Phi')\}$, we have that $(m(\Pi), m(\Phi)) \bowtie (m(\Pi'), m(\Phi'))$. So rule (Tseq) can be applied, and $m(\Theta \cup \Theta') \vdash SP \cdot SP' : (m(\mathtt{P}), m(\mathtt{R})) \sqcup (m(\mathtt{P}'), m(\mathtt{R}'))$.
- For rules (Rpar), and (Rcomp) the result can be proved like for rule (Rseq).

$\square$

**Theorem 4.3.3 (Completeness of Type Inference).** *If* $\Delta \vdash P : (\mathtt{P}, \mathtt{R})$, *then* $\vdash P : \Theta; (\Pi, \Phi); \Xi$ *for some* $\Theta$, $(\Pi, \Phi)$, $\Xi$ *and there is a type mapping* $m$ *that satisfies* $\Xi$ *and such that* $\Delta \supseteq m(\Theta)$, $\mathtt{P} = m(\Pi)$, $\mathtt{R} = m(\Phi)$.

*Proof.* By induction on the derivation of $\Delta \vdash P : (\mathtt{P}, \mathtt{R})$.

- If the last rule of the derivation is (Teps), (Tel), or (Tvar) the result is obvious. Note that, for (Tvar) in the inference we distinguish the case of element variables (from sequence or term variables).
- Rule (Tseq). The conclusion of the rule is $\Delta \vdash SP{\cdot}SP' : (\mathtt{P}, \mathtt{R}) \sqcup (\mathtt{P}', \mathtt{R}')$, and the assumptions are $\Delta \vdash SP : (\mathtt{P}, \mathtt{R})$, $\Delta \vdash SP' : (\mathtt{P}', \mathtt{R}')$ and the condition $(\mathtt{P}, \mathtt{R}) \bowtie (\mathtt{P}', \mathtt{R}')$. By induction hypothesis, there are $\Theta$, $\Pi$, $\Phi$, $\Xi$, $\Theta'$, $\Pi'$, $\Phi'$, $\Xi'$ such that $\vdash SP : \Theta; (\Pi, \Phi); \Xi$ and $\vdash SP' : \Theta'; (\Pi', \Phi'); \Xi'$. These are the assumptions of rule (Rseq), whose conclusion is $\vdash SP \cdot SP' : \Theta \cup \Theta'; (\Pi, \Phi) \sqcup (\Pi', \Phi'); \Xi \cup \Xi' \cup \{(\Pi, \Phi) \bowtie (\Pi', \Phi')\}$. Moreover, by induction there is a type mapping $m'$ satisfying $\Xi$ such that $\Delta \supseteq m'(\Theta)$, $\mathtt{P} = m'(\Pi)$ and $\mathtt{R} = m'(\Phi)$, and there is a type mapping $m''$ satisfying $\Xi'$ such that $\Delta \supseteq m''(\Theta')$, $\mathtt{P}' = m''(\Pi')$ and $\mathtt{R}' = m''(\Phi')$. Therefore, we derive $\Delta \supseteq m'(\Theta) \cup m''(\Theta')$ and $(\mathtt{P}, \mathtt{R}) \sqcup (\mathtt{P}', \mathtt{R}') = (m'(\Pi), m'(\Phi)) \sqcup (m''(\Pi'), m''(\Phi'))$. Since the basis $m'(\Theta)$ and $m''(\Theta')$ are both subsets of the same basis $\Delta$, then for all the (e-type, p-type or r-type) variables $\mu$ such that $\mu \in dom(m') \cap dom(m'')$

we get $m'(\mu) = m''(\mu)$. Therefore the mapping $m$

$$m(\mu) = \begin{cases} m'(\mu) & \text{if } \mu \in dom(m') \\ m''(\mu) & \text{if } \mu \in dom(m'') \end{cases}$$

is well defined.

Moreover, since $m$ satisfies $\Xi$, $\Xi'$ and $(\Pi, \Phi) \bowtie (\Pi', \Phi')$, then $m$ satisfies also all the constraints of the conclusion of the rule (Rseq).

- If the last rule is (Tpar) or (Tcomp) the proof is similar.

□

Now, we put our inference rules at work in order to decide the applicability of typed reduction rules, for both $\Delta$-safe and $\Delta$-($\mathtt{P}, \mathtt{R}$)-safe rules.

In order to decide applicability of $\Delta$-safe rules, we characterize them.

**Lemma 4.3.4 (Characterization of $\Delta$-safe rules).** *A rule $P_1 \mapsto P_2$ is a $\Delta$-safe rule if and only if the type mapping $m$ defined by*

1. $m(\eta_x) = \mathtt{t}$ if $\Delta(x) = (\{\mathtt{t}\}, \mathtt{R}_\mathtt{t})$
2. $m(\pi_\rho) = \mathtt{P}'$ if $\Delta(\rho) = (\mathtt{P}', \mathtt{R}')$
3. $m(\phi_\rho) = \mathtt{R}'$ if $\Delta(\rho) = (\mathtt{P}', \mathtt{R}')$

*satisfies the set of constraints $\Xi_1 \cup \Xi_2 \cup \{\Pi_1 = \Pi_2\} \cup \{\Phi_1 = \Phi_2\}$, where $\vdash P_1 : \Theta_1; (\Pi_1, \Phi_1); \Xi_1$ and $\vdash P_2 : \Theta_2; (\Pi_2, \Phi_2); \Xi_2$.*

*Proof.* ($\Leftarrow$) Since $\vdash P_1 : \Theta_1; (\Pi_1, \Phi_1); \Xi_1$, $\vdash P_2 : \Theta_2; (\Pi_2, \Phi_2); \Xi_2$ and $m$ satisfies $\Xi_1$ and $\Xi_2$, applying Theorem 4.3.2 we derive $m(\Theta_1) \vdash P_1 : (m(\Pi_1), m(\Phi_1))$, and $m(\Theta_2) \vdash P_2 : (m(\Pi_2), m(\Phi_2))$. From the definition of $m$, we have that $m(\Theta_2) \subseteq \Delta$ and $m(\Theta_2) \subseteq \Delta$, and by weakening we derive that $\Delta \vdash P_1 : (m(\Pi_1), m(\Phi_1))$ and $\Delta \vdash P_2 : (m(\Pi_2), m(\Phi_2))$. Moreover, from the fact that $m$ satisfies $\{\Pi_1 = \Pi_2\} \cup \{\Phi_1 = \Phi_2\}$, we have that $m(\Pi_1) = m(\Pi_2) = \mathtt{P}$ and $m(\Phi_1) = m(\Phi_2) = \mathtt{R}$. Therefore, $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$ and $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$, and $P_1 \mapsto P_2$ is a $\Delta$-safe rule.

($\Rightarrow$) Since $P_1 \mapsto P_2$ is a $\Delta$-safe rule, we have that $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$ and $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$. From Theorem 4.3.3, applied to $\Delta \vdash P_1 : (\mathtt{P}, \mathtt{R})$, we derive that $\vdash P_1 : \Theta_1; (\Pi_1, \Phi_1); \Xi_1$ and there is a type mapping $m'$ satisfying $\Xi_1$ such that $\Delta \supseteq m'(\Theta_1)$, $\mathtt{P} = m'(\Pi_1)$, $\mathtt{R} = m'(\Phi_1)$. Applying Theorem 4.3.3 to $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$ we derive that $\vdash P_2 : \Theta_2; (\Pi_2, \Phi_2); \Xi_2$

and there is a type mapping $m''$ satisfying $\Xi_2$ such that $\Delta \supseteq m''(\Theta_2)$, $\mathtt{P} = m''(\Pi_2)$, $\mathtt{R} = m''(\Phi_2)$. Since the basis $m'(\Theta_1)$ and $m''(\Theta_2)$ are both subsets of $\Delta$, then, the mapping $m$ defined by

$$m(\mu) = \begin{cases} m'(\mu) & \text{if } \mu \in dom(m') \\ m''(\mu) & \text{if } \mu \in dom(m'') \end{cases}$$

is well defined. Moreover, $m$ satisfies $\Xi_1 \cup \Xi_2$, and since $m'(\Pi_1) = \mathtt{P} = m''(\Pi_2)$, $m'(\Phi_1) = \mathtt{R} = m''(\Phi_2)$, then $m$ also satisfies $\{\Pi_1 = \Pi_2\} \cup \{\Phi_1 = \Phi_2\}$.

□

**Example 4.3.5.** *Using Lemma 4.3.4, we can see that the constraints making $\Delta$-safe the rule in Example 2.2.5 are*

$$\{\mathtt{t}_a\} \cup \pi_{\widetilde{x}} = \{\mathtt{t}_a, \mathtt{t}_b\} \cup \pi_{\widetilde{x}} \quad \phi_{\widetilde{x}} \setminus (\{\mathtt{t}_b\} \cup \pi_{\widetilde{x}}) = \{\mathtt{t}_c\} \cup (\phi_{\widetilde{x}} \setminus \pi_Y)$$

*plus the constraints in the sets $\Xi_1$ and $\Xi_2$ of Example 4.3.1.*

In order to decide applicability of $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rules, we characterize the $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rules and the OK relation.

**Lemma 4.3.6 (Characterization of $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rules).** *A rule $P_1 \mapsto P_2$ is $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe if and only if the type mapping $m$ defined by*

1. $m(\eta_x) = \mathtt{t}$ *if* $\Delta(x) = (\{\mathtt{t}\}, \mathtt{R}_\mathtt{t})$,
2. $m(\pi_\rho) = \mathtt{P}'$ *if* $\Delta(\rho) = (\mathtt{P}', \mathtt{R}')$,
3. $m(\phi_\rho) = \mathtt{R}'$ *if* $\Delta(\rho) = (\mathtt{P}', \mathtt{R}')$,

*satisfies the set of constraints* $\Xi_2 \cup \{\Pi_2 = \mathtt{P}, \Phi_2 = \mathtt{R}\}$, *where* $\vdash P_2 : \Theta_2; (\Pi_2, \Phi_2); \Xi_2$.

*Proof.* ($\Leftarrow$) Let $\vdash P_2 : \Theta_2; (\Pi_2, \Phi_2); \Xi_2$ and $m$ satisfies $\Xi_2 \cup \{\Pi_2 = \mathtt{P}, \Phi_2 = \mathtt{R}\}$. From Theorem 4.3.2 we derive that $m(\Theta_2) \vdash P_2 : (\mathtt{P}, \mathtt{R})$. By definition of $m$ we get $m(\Theta_2) = \Delta$. Therefore $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$, and $P_1 \mapsto P_2$ is a $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rule.

($\Rightarrow$) Let $P_1 \mapsto P_2$ be a $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rule, then $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{R})$. From Theorem 4.3.3, we have that $\vdash P_2 : \Theta_2; (\Pi_2, \Phi_2); \Xi_2$, for some $\Theta_2, \Pi_2, \Phi_2, \Xi_2$, and there is a type mapping $m'$ satisfying $\Xi$ such that $\Delta \supseteq m'(\Theta_2)$, $\mathtt{P} = m'(\Pi_2)$, and $\mathtt{R} = m'(\Phi_2)$. Therefore $m'$ satisfies $\Xi_2 \cup \{\Pi_2 = \mathtt{P}, \Phi_2 = \mathtt{R}\}$. From definition of $m$, we get $m(\Theta_2) = \Delta$, and since $\Delta \supseteq m'(\Theta_2)$, also $m$ satisfies $\Xi_2 \cup \{\Pi_2 = \mathtt{P}, \Phi_2 = \mathtt{R}\}$.

□

About the OK relation, it is not necessary to consider the whole context, but only the part of the context which influences the typing of the hole. The key observation is that the typing of a term inside two nested looping sequences does not depend on the typing of the terms outside the outermost looping sequence. We call *core of the evaluation context* the subterm of the context including the hole and the part of the context affecting the type of the hole. The following definition formalizes this notion.

**Definition 4.3.7.** *The* core of the evaluation context $E$ *(notation* $\mathsf{core}(E)$*) is defined by:*

- $\mathsf{core}(E) = E$ *if* $E \equiv \Box \mid T_1$ *or* $E \equiv (S_1)^{\circlearrowleft} \rfloor (\Box \mid T_1) \mid T_2$*;*
- $\mathsf{core}(E) = E_2$ *if* $E = E_1[E_2]$ *where* $E_2 \equiv (S_2)^{\circlearrowleft} \rfloor ((S_1)^{\circlearrowleft} \rfloor (\Box \mid T_1) \mid T_2)$.

Remark that $\mathsf{core}$ is unambiguously defined, since every evaluation context can be split in an unique way into one of the three shapes of the previous definition.

**Lemma 4.3.8 (Characterization of OK Relation).** *Let the evaluation context $E$ be such that $\vdash E[T] : (\mathtt{P}_0, \emptyset)$ for some $T, \mathtt{P}_0$. A type $(\mathtt{P}, \mathtt{R})$ is OK for $E$ if and only if the type mapping $m$ defined by*

*1.* $m(\pi_X) = \mathtt{P}$,

*2.* $m(\phi_X) = \mathtt{R}$,

*satisfies the set of constraints*

$$\Xi \cup \{\Phi = \emptyset \text{ if } \pi_X \text{ or } \phi_X \text{ occurs in } \Phi\},$$

*where* $\vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \phi_X)\}; (\Pi, \Phi); \Xi$.

*Proof.* ($\Leftarrow$) Lemma 4.2.4.(1) and $\vdash E[T] : (\mathtt{P}_0, \emptyset)$ imply that all subterms of $\mathsf{core}(E)[X]$ are typable, i.e. that there are $\mathtt{P}_1, \mathtt{R}_1, \mathtt{P}'_1, \mathtt{R}'_1, \mathtt{P}_2, \mathtt{R}_2, \mathtt{P}'_2, \mathtt{R}'_2$ such that $\vdash T_1 : (\mathtt{P}_1, \mathtt{R}_1)$, $\vdash S_1 : (\mathtt{P}'_1, \mathtt{R}'_1)$, $\vdash T_2 : (\mathtt{P}_2, \mathtt{R}_2)$, $\vdash S_2 : (\mathtt{P}'_2, \mathtt{R}'_2)$ in the last case of the definition of $\mathsf{core}(E)[X]$, and suitable subsets of these typing judgments in the other two cases.

By Definition 4.3.7 we have the following cases.

- $E = \mathsf{core}(E)$ and
  - either $\mathsf{core}(E) = \Box \mid T_1$ and $\Pi = \pi_X \cup \mathtt{P}_1$ and $\Phi = \phi_X \cup \mathtt{R}_1$,

  - or $\mathsf{core}(E) = (S_1)^{\circlearrowleft} \rfloor (\Box \mid T_1) \mid T_2$ and $\Pi = \mathtt{P}_1' \cup \mathtt{P}_2$ and $\Phi = (\mathtt{R}_1' \cup \mathtt{R}_2) \setminus (\pi_X \cup \mathtt{P}_1)$.

  Since $m$ satisfies $\{\Phi = \emptyset$ if $\pi_X$ or $\phi_X$ occurs in $\Phi\}$, then $m(\Phi) = \emptyset$. From Theorem 4.3.2, since $\vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \phi_X)\}; (\Pi, \Phi); \Xi$ and $m$ satisfies $\Xi$, we derive that $X : (\mathtt{P}, \mathtt{R}) \vdash \mathsf{core}(E)[X] : (m(\Pi), \emptyset)$. Moreover, since $E[X] = \mathsf{core}(E)[X]$, we have that $X : (\mathtt{P}, \mathtt{R}) \vdash E[X] : (m(\Pi), \emptyset)$. Therefore, the type $(\mathtt{P}, \mathtt{R})$ is OK for the context $E$.

- $\mathsf{core}(E) = (S_2)^{\circlearrowleft} \rfloor ((S_1)^{\circlearrowleft} \rfloor (\Box \mid T_1) \mid T_2)$ and $\Pi = \mathtt{P}_2'$ and $\Phi = \mathtt{R}_2' \setminus (\mathtt{P}_1' \cup \mathtt{P}_2)$.

  From $\vdash E[T] : (\mathtt{P}_0, \emptyset)$ by Lemma 4.2.4.(1) and .(2) we get $X : (\mathtt{P}', \mathtt{R}') \vdash \mathsf{core}(E)[X] : (\mathtt{P}'', \mathtt{R}'')$ for some $\mathtt{P}', \mathtt{R}', \mathtt{P}'', \mathtt{R}''$. This implies by the Completeness Theorem (Theorem 4.3.3) that there is a mapping $m'$ such that $m'(\mathtt{P}_2') = \mathtt{P}''$ and $m'(\mathtt{R}_2' \setminus (\mathtt{P}_1' \cup \mathtt{P}_2)) = \mathtt{R}''$. Since $\mathtt{P}_2'$ and $\mathtt{R}_2' \setminus (\mathtt{P}_1' \cup \mathtt{P}_2)$ do not contain variables, we get $\mathtt{P}_2' = \mathtt{P}''$ and $\mathtt{R}_2' \setminus (\mathtt{P}_1' \cup \mathtt{P}_2) = \mathtt{R}''$, independently from the types assumed for the variable $X$. This implies by Lemma 4.2.4.(3) and .(2) $X : (\mathtt{P}, \mathtt{R}) \vdash E[X] : (\mathtt{P}_0, \emptyset)$, so we conclude that $(\mathtt{P}, \mathtt{R})$ is OK for the context $E$.

($\Rightarrow$) By Definition 4.2.11, since $(\mathtt{P}, \mathtt{R})$ is OK for $E$, then $X : (\mathtt{P}, \mathtt{R}) \vdash E[X] : (\mathtt{P}', \emptyset)$ for some $\mathtt{P}'$. Theorem 4.3.3 implies that $\vdash E[X] : \Theta'; (\Pi', \Phi'); \Xi'$ and there is a type mapping $m$ that satisfies $\Xi'$ and such that $\{X : (\mathtt{P}, \mathtt{R})\} \supseteq m(\Theta')$, $m(\Pi') = \mathtt{P}$, $m(\Phi') = \emptyset$. By definition $\Theta' = \{X : (\pi_X, \phi_X)\}$, we get $m(\pi_X) = \mathtt{P}$ and $m(\phi_X) = \mathtt{R}$. Being $\mathsf{core}(E)[X]$ a subterm of $E[X]$, by Lemma 4.2.4.(1) we get $X : (\mathtt{P}, \mathtt{R}) \vdash \mathsf{core}(E)[X] : (\mathtt{P}'', \mathtt{R}')$ for some $\mathtt{P}'', \mathtt{R}'$. Theorem 4.3.3 implies that $\vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \phi_X)\}; (\Pi, \Phi); \Xi$ and by construction $\Xi \subseteq \Xi'$, so $m$ satisfies also $\Xi$. If $\mathsf{core}(E) = E$, then $\Phi = \Phi'$, which implies $m(\Phi) = \emptyset$. Otherwise neither $\pi_X$ nor $\phi_X$ occurs in $\Phi$.

□

It is easy to check that if $\mathsf{core}(E) \equiv (S_2)^{\circlearrowleft} \rfloor ((S_1)^{\circlearrowleft} \rfloor (\Box \mid T_1) \mid T_2)$, and $\vdash T_1 : (\mathtt{P}_1, \mathtt{R}_1)$, $\vdash S_1 : (\mathtt{P}_1', \mathtt{R}_1')$, $\vdash T_2 : (\mathtt{P}_2, \mathtt{R}_2)$, $\vdash S_2 : (\mathtt{P}_2', \mathtt{R}_2')$, then to prove that $E$ is OK we have to verify the following six constraints:

- $(\pi_X, \phi_X) \bowtie (\mathtt{P}_1, \mathtt{R}_1)$

- $(\mathtt{P}'_1, \mathtt{R}'_1) \bowtie ((\pi_X, \phi_X) \sqcup (\mathtt{P}_1, \mathtt{R}_1))$
- $((\phi_X \cup \mathtt{R}_1) \setminus (\pi_X \cup \mathtt{P}_1)) \subseteq \mathtt{P}'_1$
- $(\mathtt{P}'_1, \mathtt{R}'_1 \setminus (\pi_X \cup \mathtt{P}_1)) \bowtie (\mathtt{P}_2, \mathtt{R}_2)$
- $(\mathtt{P}'_2, \mathtt{R}'_2) \bowtie ((\mathtt{P}'_1, \mathtt{R}'_1 \setminus (\pi_X \cup \mathtt{P}_1)) \sqcup (\mathtt{P}_2, \mathtt{R}_2))$
- $(((\mathtt{R}'_1 \setminus (\pi_X \cup \mathtt{P}_1)) \cup \mathtt{R}_2) \setminus (\mathtt{P}'_1 \cup \mathtt{P}_2)) \subseteq \mathtt{P}'_2$.

The set of constraints is smaller when the core context has one of the simpler shapes.

**Example 4.3.9.** *Using Lemma 4.3.8, the constraints making the type associated with the p-type and r-type variable $X$ OK for the contexts in Example 2.2.8 are:*

$$\begin{array}{lll} (A) & (\{\mathtt{t}_a, \mathtt{t}_c\}, \emptyset) \bowtie (\pi_X, \phi_X) & \phi_X \subseteq \{\mathtt{t}_a, \mathtt{t}_c\} \\ (B) & (\{\mathtt{t}_a\}, \emptyset) \bowtie (\pi_X, \phi_X) & \phi_X \subseteq \{\mathtt{t}_a\} \\ (C) & (\{\mathtt{t}_d\}, \emptyset) \bowtie (\pi_X, \phi_X) & \phi_X = \emptyset. \end{array}$$

The previous lemmas imply the following theorem asserting the condition of applicability of the rewrite rules.

**Theorem 4.3.10 (Applicability of rewrite rules).** *Let*

$$\begin{array}{ll} \vdash P_1 : \Theta_1; (\Pi_1, \Phi_1); \Xi_1 \quad \textit{and} \quad \vdash P_2 : \Theta_2; (\Pi_2, \Phi_2); \Xi_2 \quad \textit{and} \\ \vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \phi_X)\}; (\Pi_E, \Phi_E); \Xi_E \quad \textit{and} \quad P_1\sigma \not\equiv \epsilon. \end{array}$$

*Then the rule $P_1 \mapsto P_2$ can be applied to the term $E[P_1\sigma]$ such that $\vdash E[P_1\sigma] : (\mathtt{P}_0, \emptyset)$ (for some $\mathtt{P}_0$) if and only if the type mapping $m$ defined by*

1. *$m(\eta_x) = \mathtt{t}$ if $\sigma(x) : \mathtt{t} \in \Psi$,*
2. *$m(\pi_\rho) = \mathtt{P}'$ if $\vdash \sigma(\rho) : (\mathtt{P}', \mathtt{R}')$,*
3. *$m(\phi_\rho) = \mathtt{R}'$ if $\vdash \sigma(\rho) : (\mathtt{P}', \mathtt{R}')$,*

*satisfies*

(a) *either the set of constraints $\Xi_1 \cup \Xi_2 \cup \{\Pi_1 = \Pi_2\} \cup \{\Phi_1 = \Phi_2\}$,*

(b) *or the set of constraints $\Xi_2 \cup \Xi_E \cup \{\Pi_2 = \pi_X, \Phi_2 = \phi_X\} \cup \{\Phi_E = \emptyset \mid$ if $\pi_X$ or $\phi_X$ occurs in $\Phi_E\}$.*

*Proof.* We define the basis $\Delta$ as follows:

$x : (\{\mathtt{t}\}, \mathtt{R}_\mathtt{t}) \in \Delta \quad \text{if} \quad \sigma(x) : \mathtt{t} \in \Psi$, and

$\rho : (\mathtt{P}', \mathtt{R}') \in \Delta \quad \text{if} \quad \vdash \sigma(\rho) : (\mathtt{P}', \mathtt{R}')$.

In this way we get that $\sigma \in \Sigma_\Delta$ and the type mapping $m$ is such that:

1. $m(\eta_x) = \mathtt{t}$ if and only if $x : (\{\mathtt{t}\}, \mathtt{R}_\mathtt{t}) \in \Delta$
2. $m(\pi_\rho) = \mathtt{P}'$ if and only if $\rho : (\mathtt{P}', \mathtt{R}') \in \Delta$
3. $m(\phi_\rho) = \mathtt{R}'$ if and only if $\rho : (\mathtt{P}', \mathtt{R}') \in \Delta$.

Let $\Re = P_1 \mapsto P_2$.

($\Leftarrow$) If the mapping $m$ satisfies the set of constraints $\Xi_1 \cup \Xi_2 \cup \{\Pi_1 = \Pi_2\} \cup \{\Phi_1 = \Phi_2\}$, then by Lemma 4.3.4 the rule $P_1 \mapsto P_2$ is $\Delta$-safe and we get $E[P_1\sigma] \Longrightarrow E[P_2\sigma]$ by applying rule ($\Re$-$\Delta$).

If the mapping $m$ satisfies the set of constraints $\Xi_2 \cup \Xi_E \cup \{\Pi_2 = \pi_X, \Phi_2 = \phi_X\} \cup \{\Phi_E = \emptyset \mid \text{ if } \pi_X \text{ or } \phi_X \text{ occurs in } \Phi_E\}$, then by Lemma 4.3.8 the context $E$ is OK for $(\mathtt{P}, \mathtt{R})$ and by Lemma 4.3.6 the rule $P_1 \mapsto P_2$ is $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe; we get $E[P_1\sigma] \Longrightarrow E[P_2\sigma]$ by applying rule ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$).

($\Rightarrow$) If $E[P_1\sigma] \Longrightarrow E[P_2\sigma]$ by applying rule ($\Re$-$\Delta$), then the rule $P_1 \mapsto P_2$ is $\Delta$-safe and then the mapping $m$ satisfies the set of constraints $\Xi_1 \cup \Xi_2 \cup \{\Pi_1 = \Pi_2\} \cup \{\Phi_1 = \Phi_2\}$ by Lemma 4.3.4.

If $E[P_1\sigma] \Longrightarrow E[P_2\sigma]$ by applying rule ($\Re$-$\Delta$-$(\mathtt{P}, \mathtt{R})$), then the rule $P_1 \mapsto P_2$ is $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe and the context $E$ is Ok for $(\mathtt{P}, \mathtt{R})$, then the mapping $m$ satisfies the set of constraints $\Xi_2 \cup \Xi_E \cup \{\Pi_2 = \pi_X, \Phi_2 = \phi_X\} \cup \{\Phi_E = \emptyset \mid \text{ if } \pi_X \text{ or } \phi_X \text{ occurs in } \Phi_E\}$ by Lemmas 4.3.8 and 4.3.6.

□

The mapping $m$ may be easily defined from the derivation of a type for $P_1\sigma$, and the checking that $m$ satisfies a set of constraints requires only some substitutions.

Note that the sets of constraints for typing the left-hand-side and the right-hand-side of $\Delta$-safe rules, and the right-hand-side of $\Delta$-$(\mathtt{P}, \mathtt{R})$-safe rules, can be inferred once for all. On the contrary, the set of constraints for typing the core context has to be inferred during the application of a $\Delta$-$(\mathtt{P},\mathtt{R})$-safe rule. However, as previously remarked, this set of constraints includes at most six constraints.

Theorem 4.2.17 implies that, during inference, we can firstly check if the rule is $\Delta$-safe, using the constraints associated with the rule. If these constraints are not satisfied, we can check whether the rule is a $\Delta$-$(\mathtt{P},\mathtt{R})$-safe rule (for some $(\mathtt{P},\mathtt{R})$), and if the type is OK for the context. We can summarize the idea in the following algorithm, where $\Delta$ is defined in the proof of Theorem 4.3.10:

- In the initial phase, for every rule $\Re = P_1 \mapsto P_2 \in \mathcal{R}$, we infer $\vdash P_1 : \Theta; (\Pi_1, \Phi_1); \Xi_1$ and $\vdash P_2 : \Theta; (\Pi_2, \Phi_2); \Xi_2$.
- During the reduction of the well-typed term $E[P_1\,\sigma]$ with $\Re = P_1 \mapsto P_2$, we firstly check whether the conditions of $(\Re\text{-}\Delta)$ hold, that is:
    1. we check whether the type mapping $m$, defined by
       $m(\eta_x) = \mathtt{t}$ if $\sigma(x) : \mathtt{t} \in \Psi$,
       $m(\pi_\rho) = \mathtt{P}'$ if $\vdash \sigma(\rho) : (\mathtt{P}', \mathtt{R}')$,
       $m(\phi_\rho) = \mathtt{R}'$ if $\vdash \sigma(\rho) : (\mathtt{P}', \mathtt{R}')$,
       satisfies $\Xi_2$. If the set of constraints is not satisfied, $P_2\,\sigma$ is not well typed, and neither $(\Re\text{-}\Delta)$ nor $(\Re\text{-}\Delta\text{-}(\mathtt{P},\mathtt{R}))$ would be applicable, i.e. $E[P_1\,\sigma]$ is not reducible via $P_1 \mapsto P_2$;
    2. we check if $m$ satisfies $\{\Pi_1 = \Pi_2\} \cup \{\Phi_1 = \Phi_2\}$. If this is the case, the rule is a $\Delta$-safe rule, and then we can apply $(\Re\text{-}\Delta)$,
    3. otherwise we check whether the conditions of $(\Re\text{-}\Delta\text{-}(\mathtt{P},\mathtt{R}))$ hold, where
       $\mathtt{P} = m(\Pi_2)$ and $\mathtt{R} = m(\Phi_2)$; in order to do this
        (a) we infer $\vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \phi_X)\}; (\Pi_E, \Phi_E); \Xi_E$, and
        (b) we check whether $m$ satisfies
        $$\Xi_E \cup \{\Pi_2 = \pi_X, \Phi_2 = \phi_X\} \cup \{\Phi_E = \emptyset \text{ if } \pi_X \text{ or } \phi_X \text{ occurs in } \Phi_E\}.$$
        If these constraints are satisfied, the context $E$ is OK for $(\mathtt{P},\mathtt{R})$, so we can use rule $(\Re\text{-}\Delta\text{-}(\mathtt{P},\mathtt{R}))$, otherwise neither rule $(\Re\text{-}\Delta)$ nor $(\Re\text{-}\Delta\text{-}(\mathtt{P},\mathtt{R}))$ is applicable, i.e. $E[P_1\,\sigma]$ is not reducible via $P_1 \mapsto P_2$.

| | $\pi_{\widetilde{x}}$ | $\phi_{\widetilde{x}}$ | $\pi_Y$ | $\phi_Y$ | $\pi_X$ | $\phi_X$ |
|---|---|---|---|---|---|---|
| (1) | $\mathtt{t}_b, \mathtt{t}_c$ | $\mathtt{t}_a$ | $\emptyset$ | $\emptyset$ | $\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c$ | $\emptyset$ |
| (2) | $\mathtt{t}_c$ | $\mathtt{t}_a$ | $\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c$ | $\emptyset$ | $\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c$ | $\emptyset$ |
| (3) | $\emptyset$ | $\emptyset$ | $\mathtt{t}_a$ | $\emptyset$ | $\mathtt{t}_a, \mathtt{t}_b$ | $\mathtt{t}_c$ |

Figure 4.4: Type mappings of Example 4.3.11

Note that, since the point 1 is in common with the algorithm for checking the $\Delta$-safeness, the above algorithm only adds the simple constraints in point 2 to the normal algorithm for performing the $\Delta$-$(\mathtt{P}, \mathtt{R})$-safeness of a rule, but if they are satisfied we do not have to infer the type for the context, the most complicate point of the algorithm.

**Example 4.3.11.** *We use the algorithm described above on the terms of Example 4.2.16: the constraints for $\Delta$-safe rules and OK relations for the contexts are reported in Examples 4.3.5 and 4.3.9, respectively. The type mappings derived from the instantiation are reported in Fig. 4.4.*

(1) *The type mapping in line (1) of Fig. 4.4 satisfies*

- *the constraints in $\Xi_2$ associated with $P_2$ (see Example 4.3.1) and*
- *the constraints that make the rule a $\Delta$-safe rule (see Example 4.3.5).*

(2) *The type mapping in line (2) of Fig. 4.4*

- *satisfies the constraints in $\Xi_2$ associated with $P_2$,*
- *does not satisfy the constraint $\{\mathtt{t}_a\} \cup \pi_{\widetilde{x}} = \{\mathtt{t}_a, \mathtt{t}_b\} \cup \pi_{\widetilde{x}}$ that make the rule a $\Delta$-safe rule (see Example 4.3.5) because*

$$\{\mathtt{t}_a, \mathtt{t}_c\} \neq \{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}$$

- *satisfies the set of constraints $(B)$ for the context (see Example 4.3.9), since*

$$(\mathtt{t}_a, \emptyset) \bowtie (\{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}, \emptyset) \quad \emptyset \subseteq \{\mathtt{t}_a\}$$

(3) *The type mapping in line (3) of Fig. 4.4*

- *satisfies the constraints in $\Xi_2$ associated with $P_2$,*

- *does not satisfy the constraint* $\{\mathtt{t}_a\} \cup \pi_{\widetilde{x}} = \{\mathtt{t}_a, \mathtt{t}_b\} \cup \pi_{\widetilde{x}}$ *that make the rule a* $\Delta$*-safe rule (see Example 4.3.5) because*

$$\{\mathtt{t}_a\} \neq \{\mathtt{t}_b, \mathtt{t}_c\}$$

- *does not satisfy the set of constraints* $(C)$ *for the context (see Example 4.3.9), since*

$$(\{\mathtt{t}_d\}, \emptyset) \bowtie (\{\mathtt{t}_a, \mathtt{t}_b\}, \emptyset) \quad \textit{does not hold.}$$

## 4.4 Case Study: Blood Type Compatibility

A blood type is a classification of blood based on the presence or absence of inherited antigenic substances on the surface of red blood cells: these antigens are the A antigen and the B antigen. Blood type A contains only A antigens, blood type B contains only B antigens, blood type AB contains both, blood type O contains none of them: this classification is called ABO blood type system.

The immune system will produce antibodies that can specifically bind to a blood group antigen that is not recognized as self: individuals of blood type A have Anti-B antibodies, individuals of blood type B have Anti-A antibodies, individuals of blood type O have both Anti-A and Anti-B antibodies, and individuals of blood type AB have none of them. These antibodies can bind to the antigens on the surface of the transfused red blood cells, often leading to the destruction of the cell: for this reason, it is vital that compatible blood is selected for transfusions.

Another antigen that refines the classification of blood types is the RhD antigen: if this antigen is present, the blood type is called positive, else it is called negative. Unlike the ABO blood classification, the RhD antigen is immunogenic, meaning that a person who is RhD negative is very likely to produce Anti-RhD antibodies when exposed to the RhD antigen, but it is also common for RhD-negative individuals not to have Anti-RhD antibodies. All these aspects led to the red blood cell compatibility table in Table 4.1.

We study the blood transfusion in a system consisting of a set of closed tissues. These tissues, containing blood cells and antibodies according to the rules described above, can join each other and exemplify a transfusion of different blood types. We model a red blood cell as a looping sequence containing the element $c$ on the surface and, depending on the blood type,

|  | Donor |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| Recipient | O- | O+ | A- | A+ | B- | B+ | AB- | AB+ |
| O- | √ |  |  |  |  |  |  |  |
| O+ | √ | √ |  |  |  |  |  |  |
| A- | √ |  | √ |  |  |  |  |  |
| A+ | √ | √ | √ | √ |  |  |  |  |
| B- | √ |  |  |  | √ |  |  |  |
| B+ | √ | √ |  |  | √ | √ |  |  |
| AB- | √ |  | √ |  | √ |  | √ |  |
| AB+ | √ | √ | √ | √ | √ | √ | √ | √ |

Table 4.1: Red blood cell compatibilities.

the elements $a$, $b$ and $r$ as the A antigen, the B antigen and the RhD antigen, respectively. We represent the antibodies as the single elements $\bar{a}$, $\bar{b}$ and $\bar{r}$, modeling, respectively, the Anti-A, Anti-B and Anti-RhD antibodies. Finally, we model a tissue (which can contains the red cells) as a looping sequence having only the element $t$ on the surface. In order to avoid undesirable behaviors using basic CLS, we should must write as many rules as the different combinations of the different blood types shown in Table 4.1. Using the typed extension of CLS, according to the antigen and antibodies requirements and exclusions, we only create the basic types shown in Table 4.2, and we use the single rule

$$(t)^{\circlearrowleft} \rfloor X \mid (t)^{\circlearrowleft} \rfloor Y \mapsto (t)^{\circlearrowleft} \rfloor (X \mid Y)$$

for modeling transfusion.

Let the system be a set of eight tissues, containing each possible recipient combination of blood with antibodies:

$$\begin{array}{lclc}
(t)^{\circlearrowleft} \rfloor ((c)^{\circlearrowleft} \rfloor \epsilon \mid \bar{a} \mid \bar{b} \mid \bar{r}) & \mid & (t)^{\circlearrowleft} \rfloor ((c \cdot a)^{\circlearrowleft} \rfloor \epsilon \mid \bar{b} \mid \bar{r}) & \mid \\
(t)^{\circlearrowleft} \rfloor ((c \cdot b)^{\circlearrowleft} \rfloor \epsilon \mid \bar{a} \mid \bar{r}) & \mid & (t)^{\circlearrowleft} \rfloor ((c \cdot r)^{\circlearrowleft} \rfloor \epsilon \mid \bar{a} \mid \bar{b}) & \mid \\
(t)^{\circlearrowleft} \rfloor ((c \cdot a \cdot b)^{\circlearrowleft} \rfloor \epsilon \mid \bar{r}) & \mid & (t)^{\circlearrowleft} \rfloor ((c \cdot a \cdot r)^{\circlearrowleft} \rfloor \epsilon \mid \bar{b}) & \mid \\
(t)^{\circlearrowleft} \rfloor ((c \cdot b \cdot r)^{\circlearrowleft} \rfloor \epsilon \mid \bar{a}) & \mid & (t)^{\circlearrowleft} \rfloor ((c \cdot a \cdot b \cdot r)^{\circlearrowleft} \rfloor \epsilon) &
\end{array}$$

They cannot react with each other, because the antibodies of the ones exclude the antigens of the others. If in the system arrives a donor, as a tissue without antibodies, having blood type O-:

$$(t)^{\circlearrowleft} \rfloor ((c)^{\circlearrowleft} \rfloor \epsilon),$$

| element | basic type | R set | E set |
|---|---|---|---|
| $c$ | $\mathtt{t}_c$ | $\emptyset$ | $\emptyset$ |
| $a$ | $\mathtt{t}_a$ | $\mathtt{t}_c$ | $\emptyset$ |
| $b$ | $\mathtt{t}_b$ | $\mathtt{t}_c$ | $\emptyset$ |
| $r$ | $\mathtt{t}_r$ | $\mathtt{t}_c$ | $\emptyset$ |
| $\bar{a}$ | $\mathtt{t}_{\bar{a}}$ | $\mathtt{t}_c$ | $\mathtt{t}_a$ |
| $\bar{b}$ | $\mathtt{t}_{\bar{b}}$ | $\mathtt{t}_c$ | $\mathtt{t}_b$ |
| $\bar{r}$ | $\mathtt{t}_{\bar{r}}$ | $\mathtt{t}_c$ | $\mathtt{t}_r$ |
| $t$ | $\mathtt{t}_t$ | $\emptyset$ | $\emptyset$ |

Table 4.2: Elements, basic types, R and E sets for red blood cell compatibility.

it can singularly react with each tissue, because it does not have antigens, whereas if in the system arrives a donor having blood type O+:

$$(t)^{\circlearrowleft} \rfloor ((c \cdot r)^{\circlearrowleft} \rfloor \epsilon),$$

it can singularly react with the tissues that do not contain the Anti-RhD antibody $\bar{r}$. As further example, if in the system arrives a donor having blood type A+:

$$(t)^{\circlearrowleft} \rfloor ((c \cdot a \cdot r)^{\circlearrowleft} \rfloor \epsilon),$$

it can singularly react with each tissue that does not contain Anti-RhD and Anti-A antibodies $\bar{r}$ and $\bar{a}$, so tissues containing A+ and AB+ blood types.

## 4.5 Conclusions

In this chapter we introduce a type system for CLS, used to define a typed semantics in which the applicability of rules is determined by type conditions on the applied rules and on the context of application. A type inference system and an algorithm for performing reductions are also presented.

As seen in Section 4.4, the use of a typed semantic for CLS allows to transfer the complexity of biological properties from rules to types, and so to study the behavior of the systems using only simple and general rules: we focused on disciplines deriving by the requirement/exclusion of certain elements, even if in nature it is not easy to find elements which completely exclude or require other elements. Our abstraction, however, allows us to deal with a simple qualitative model, and to observe some basic properties of biological systems. A more detailed analysis could also deal with quantities. In this case, typing is useful in modeling quantitative aspects of CLS semantics on the line of [11]. In particular, in [39], the authors show a simple example on how types could be used to model repellency also by quantitative means, that is slowing down undesired interactions.

As a future work, we plan to investigate type disciplines assuring different properties for CLS and to apply this approach to other calculi for describing evolution of biological systems, in particular to P systems.

# Chapter 5

# Enumerated Type Semantics for Calculus of Looping Sequences

## 5.1 Introduction

Homeostasis is the property of a system that regulates its internal environment and tends to maintain stable conditions that are optimal for survival: when this equilibrium is disturbed, built-in regulatory devices respond in order to restore the balance. Different living organisms employ homeostatic mechanisms to maintain some conditions in specific ranges: the human body, like in all the warm-blooded animals, maintain a near-constant body temperature using mechanisms such as vasodilation and vasoconstriction; microorganisms maintain the iron presence above a minimum level to maintain life but up to a maximum level to avoid iron toxicity. Biological molecules are usually made up of a certain number of different subcomponents: proteins are composed by different domains, some proteins are multimers, ribosomes are a mixture of RNA and proteins, etc. Monomers are molecules that may become chemically bounded to other monomers to form a polymer: for example, antibodies can be monomers, dimers or pentamers, the Triose phosphate isomerase, an enzyme essential for efficient energy production, is a dimer of identical subunits, and the Glutamate dehydrogenase 1, a mitochondrial matrix enzyme with a key role in the nitrogen and glutamate metabolism, is a hexamer. The type discipline in Chapter 4, that checks the simple presence or absence of molecules, cannot manage the case in which the number of certain molecules must be kept in a given interval, like for homeostasis or polymers. For this reason, in this Chapter we present the extension of the Type Disciplines for Required and Excluded Elements proposed in [18], where we assume that for each element of our system we can fix the minimum and the maximum number of other elements it requires. We enrich CLS with a type discipline and typed reductions that guarantee the soundness of reduction rules with respect to the properties of biological systems deriving from the minimum and the maximum requested numbers of elements.

## 5.2 Type Discipline

Let $\Gamma$ be the set of *basic types*: As in Section 4.2, we classify elements in $\mathcal{A}$ with basic types, and we use $\Psi$ to denote this classification. Different elements can have the same basic type, but when there is no ambiguity we denote the type associated with an element $a$ by $\mathtt{t}_a$. We assume the existence of two functions, $\mathtt{mn} : \Gamma \times \Gamma \rightarrow \mathbb{N}$ and $\mathtt{mx} : \Gamma \times \Gamma \rightarrow \mathbb{N} + \{\infty\}$ for every ordered pair of basic types $(\mathtt{t}_1, \mathtt{t}_2)$. These functions indicate the minimum and maximum number of elements of basic type $\mathtt{t}_2$ that can be present with an element of basic type $\mathtt{t}_1$: the absence of a maximum limit is denoted by the infinity symbol $\infty$. For example, $\mathtt{mn}(\mathtt{t}_\mathtt{a}, \mathtt{t}_\mathtt{b}) = \mathtt{3}$ and $\mathtt{mx}(\mathtt{t}_\mathtt{a}, \mathtt{t}_\mathtt{b}) = \mathtt{5}$ means that in the presence of some element of basic type $\mathtt{t}_a$ the number of elements of basic type $\mathtt{t}_b$ must be between 3 and 5. Note that in the cases $\mathtt{mn}(\mathtt{t}, \mathtt{t})$ and $\mathtt{mx}(\mathtt{t}, \mathtt{t})$ we are taking into account the number of *other* elements of the same basic type: for example, $\mathtt{mn}(\mathtt{t}, \mathtt{t}) = \mathtt{mx}(\mathtt{t}, \mathtt{t}) = \mathtt{0}$ means that an element of type $\mathtt{t}$ cannot tolerate *any other* element of the same basic type, while $\mathtt{mn}(\mathtt{t}, \mathtt{t}) = \mathtt{mx}(\mathtt{t}, \mathtt{t}) = \mathtt{1}$ means that an element of type $\mathtt{t}$ requires exactly *another one* element of the same basic type. For the sake of clarity, we will write 'an element $a$ requires $n$ elements $b$' if $n = \mathtt{mn}(\mathtt{t}_\mathtt{a}, \mathtt{t}_\mathtt{b})$, and 'an element $a$ tolerates $m$ elements $b$' if $m = \mathtt{mx}(\mathtt{t}_\mathtt{a}, \mathtt{t}_\mathtt{b})$. As done in Chapter 4, we consider only local properties, i.e elements influence each other if they are either present in the same compartment, or one is present in the looping sequence and the other one is present in the inner compartment of a containment operator.

The functions $\mathtt{mn}$ and $\mathtt{mx}$ must satisfy some consistency requirement.

**Definition 5.2.1 (Consistent Basic Types).** *A system composed by a set of basic types $\Gamma$ and the functions* $\mathtt{mn}$ *and* $\mathtt{mx}$ *is* consistent *if:*

*1.* $\forall\ \mathtt{t}_1, \mathtt{t}_2 \in \Gamma \quad \mathtt{mn}(\mathtt{t}_1, \mathtt{t}_2) \leq \mathtt{mx}(\mathtt{t}_1, \mathtt{t}_2)$*;*

*2.* $\forall\ \mathtt{t}_1, \mathtt{t}_2 \in \Gamma \quad \mathtt{mx}(\mathtt{t}_1, \mathtt{t}_2) = \mathtt{0} \implies \mathtt{mn}(\mathtt{t}_2, \mathtt{t}_1) = \mathtt{0}$*;*

*3.* $\forall\ \mathtt{t}_1, \mathtt{t}_2 \in \Gamma \quad \mathtt{mn}(\mathtt{t}_1, \mathtt{t}_2) > \mathtt{0} \implies \mathtt{mx}(\mathtt{t}_2, \mathtt{t}_1) > \mathtt{0}$.

The meaning of the conditions stated in Definition 5.2.1 is the following:

1. the minimum number of elements of basic type $\mathtt{t}_2$ required by the elements of basic type $\mathtt{t}_1$ must be lower than the maximum number of elements of the same basic type $\mathtt{t}_2$ tolerated by the elements of basic type $\mathtt{t}_1$;

2. if elements of basic type $\mathtt{t}_1$ do not tolerate elements of basic type $\mathtt{t}_2$, then the elements of basic type $\mathtt{t}_2$ cannot require elements of basic type $\mathtt{t}_1$.

3. if elements of basic type $\mathtt{t}_1$ require the presence of a certain number of elements having basic type $\mathtt{t}_2$, then the elements of basic type $\mathtt{t}_2$ must tolerate elements of basic type $\mathtt{t}_1$.

Note that intolerance is not symmetric: if elements of basic type $\mathtt{t}_1$ cannot tolerate (i.e. can tolerate 0) elements of basic type $\mathtt{t}_2$, it is possible that elements of a basic type $\mathtt{t}_2$ can tolerate elements of basic type $\mathtt{t}_1$.

**Example 5.2.2.** *The system*

$$\Gamma = \{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c\}$$

*where* mn, mx *are:*

| mn | $\mathtt{t}_a$ | $\mathtt{t}_b$ | $\mathtt{t}_c$ |
|---|---|---|---|
| $\mathtt{t}_a$ | 0 | 0 | 0 |
| $\mathtt{t}_b$ | 0 | 0 | 1 |
| $\mathtt{t}_c$ | 1 | 0 | 0 |

| mx | $\mathtt{t}_a$ | $\mathtt{t}_b$ | $\mathtt{t}_c$ |
|---|---|---|---|
| $\mathtt{t}_a$ | $\infty$ | 0 | 1 |
| $\mathtt{t}_b$ | $\infty$ | $\infty$ | $\infty$ |
| $\mathtt{t}_c$ | $\infty$ | 1 | $\infty$ |

*is consistent.*

In the P/R type discipline, proposed in Chapter 4, each basic type t is associated with a pair of sets of basic types $(\mathtt{R}_\mathtt{t}, \mathtt{E}_\mathtt{t})$, where $\mathtt{t} \notin \mathtt{R}_\mathtt{t} \cup \mathtt{E}_\mathtt{t}$ and $\mathtt{R}_\mathtt{t} \cap \mathtt{E}_\mathtt{t} = \emptyset$, i.e. the presence of elements of basic type t requires the presence of elements whose basic type belongs to $\mathtt{R}_\mathtt{t}$ and forbids the presence of elements whose basic type belongs to $\mathtt{E}_\mathtt{t}$. We can express the sets R and E of the

- **Multiset**: a *multiset* over a set $D$ is a pair $\langle D, f\rangle$, where $f : D \to \mathbb{N} \cup \{\infty\}$ is a function, called *multiplicity function*.
- **Empty multiset**: a multiset $A = \langle D, f_A\rangle$ is the empty multiset, denoted by $\emptyset$, if $\forall\, e \in D \quad f_A(e) = 0$.
- **Infinite multiset**: a multiset $A = \langle D, f_A\rangle$ is the infinite multiset, denoted by $D_\infty$, if $\forall\, e \in D \quad f_A(e) = \infty$.
- **Sub-multiset**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then $A$ is a *sub-multiset* of $B$, denoted $A \subseteq B$, if $\forall\, e \in D \quad f_A(e) \leq f_B(e)$.
- **Sum**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then their *sum*, denoted $A \uplus B$, is the multiset $C = \langle D, f_C\rangle$ such that $\forall\, e \in D$: $f_C(e) = f_A(e) + f_B(e)$.
- **Removal**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then the *removal* of multiset $B$ from $A$, denoted $A \ominus B$, is the multiset $C = \langle D, f_C\rangle$ such that $\forall\, e \in D \quad f_C(e) = max(f_A(e) - f_B(e), 0)$.
- **Union**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then their *union*, denoted $A \cup B$, is the multiset $C = \langle D, f_C\rangle$ such that $\forall\, e \in D \quad f_C(e) = max(f_A(e), f_B(e))$.
- **Intersection**: if $A = \langle D, f_A\rangle$ and $B = \langle D, f_B\rangle$ are two multisets, then their *intersection*, denoted $A \cap B$, is the multiset $C = \langle D, f_C\rangle$ such that $\forall\, e \in D \quad f_C(e) = min(f_A(e), f_B(e))$.
- We convene that $\forall\, m \in \mathbb{N} \cup \{\infty\}$, $\forall\, n \in \mathbb{N}$
  - $m \leq \infty$
  - $m + \infty = \infty$
  - $\infty - n = \infty$
  - $n - \infty = 0$.

Figure 5.1: Multiset Basic Definitions

P/R type discipline by means of the functions $\mathtt{mn}$ and $\mathtt{mx}$: given a basic type $\mathtt{t}$, we have $\mathtt{mn}(\mathtt{t},\mathtt{t}) = \mathtt{0}$ and $\mathtt{mx}(\mathtt{t},\mathtt{t}) = \infty$; for every $\mathtt{t}' \in \mathtt{R}_\mathtt{t}$ we have $\mathtt{mn}(\mathtt{t},\mathtt{t}') = \mathtt{1}$ and $\mathtt{mx}(\mathtt{t},\mathtt{t}') = \infty$; for every $\mathtt{t}'' \in \mathtt{E}_\mathtt{t}$ we have $\mathtt{mn}(\mathtt{t},\mathtt{t}'') = \mathtt{mx}(\mathtt{t},\mathtt{t}'') = \mathtt{0}$. For this reason, the present type discipline can be seen a refinement of the P/R type discipline.

Types are triples $(\mathtt{P},\mathtt{L},\mathtt{M})$ of multisets over the set $\Gamma$ of basic types, where $\mathtt{P}$ (*present-ms*) is the multiset of basic types of *present* elements (the elements present at the top level compartment of a pattern, i.e. in the outermost compartment), $\mathtt{L}$ (*at-least-ms*) is the multiset of the basic types still required by the present elements, and $\mathtt{M}$ (*at-most-ms*) is the multiset of the basic types still tolerated by the present elements. Some basic definitions about

multisets, taken from [64] and extended with infinity, are reported in Figure 5.1.
Given a basic type $\mathtt{t}$, we define its *down-ms* $\mathtt{D_t}$ as the multiset $\langle \Gamma, f_{\mathtt{D_t}} \rangle$, where $f_{\mathtt{D_t}}(\mathtt{t}') = \mathtt{mn}(\mathtt{t}, \mathtt{t}')$ (for $\mathtt{t}' \in \Gamma$), and its *up-ms* $\mathtt{U_t}$ as the multiset $\langle \Gamma, f_{\mathtt{U_t}} \rangle$, where $f_{\mathtt{U_t}}(\mathtt{t}') = \mathtt{mx}(\mathtt{t}, \mathtt{t}')$ (for $\mathtt{t}' \in \Gamma$).
A type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is *well formed* if:

1. the multiset $\mathtt{L}$ is a subset of the multiset $\mathtt{M}$, i.e. the multiplicity of each basic type in $\mathtt{L}$ is less or equal to the multiplicity of the same basic type in $\mathtt{M}$,

2. the multiset $\mathtt{L}$ is contained in the union of the *down-ms* of the types in $\mathtt{P}$,

3. the sum of the multisets $\mathtt{P}$ and $\mathtt{M}$ is contained in the intersection of the *up-ms* of each basic type in $\mathtt{P}$, taking into account the basic type itself.

More formally:

**Definition 5.2.3 (Well-Formed Types).** *A type* $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ *is* well formed *if* $\mathtt{L} \subseteq \mathtt{M}$*,* $\mathtt{L} \subseteq \bigcup_{\mathtt{t} \in \mathtt{P}} \mathtt{D_t}$ *and* $\mathtt{P} \uplus \mathtt{M} \subseteq \bigcap_{\mathtt{t} \in \mathtt{P}} (\mathtt{U_t} \uplus \{\mathtt{t}\})$*.*

The first limitation is obvious. The second one is more complex. The multiplicity of basic type $\mathtt{t}_1$ in $\mathtt{L}$ must be lower than the multiplicity of the same basic type in each minimum limit required by the elements in $\mathtt{P}$. Note that if a basic type in $\mathtt{P}$ requires $n$ elements of basic type $\mathtt{t}_1$, and a different basic type in $\mathtt{P}$ requires $m > n$ elements of the same basic type $\mathtt{t}_1$, the multiplicity of $\mathtt{t}_1$ in $\mathtt{L}$ cannot be greater than $m$, because any lower number may respect the first constraint, but surely violates the second one: for this reason, the multiplicity of $\mathtt{t}_1$ in $\mathtt{L}$ must be lower than the maximum of the minimum limits. Since all the requirement of a basic type $\mathtt{t}$ are contained into its *down-ms* $\mathtt{D_t}$, and in multiset theory the maximum between the multiplicities of multisets is given by the union operator (see Figure 5.1), then the *at-least-ms* must be a subset of the union of the *down-ms* of all the basic types in $\mathtt{P}$. Finally, let's explain the third constraint. Since the *up-ms* contains the maximum number of basic types still tolerated by the elements in the *present-ms*, the sum between $\mathtt{P}$ and $\mathtt{M}$ must be a subset of $\mathtt{U_t}$, for each $\mathtt{t}$ in $\mathtt{P}$, i.e. the number of elements present and still tolerated cannot overcome the maximum limit of the element tolerated by each element of the term.

Since by definition the *up-ms* of a basic type $\mathtt{t}$ does not take into account the presence of $\mathtt{t}$ itself, but $\mathtt{P}$ does it, we have to sum to the *up-ms* of a basic type $\mathtt{t}$ the basic type itself.

In the following we will consider only well-formed types.
Summarizing, the (well-formed) type of a pattern is $(\mathtt{P}, \mathtt{L}, \mathtt{M})$, where:

- $\mathtt{P}$: the number of elements of a certain basic type which are presents out of the outermost compartment,
- $\mathtt{L}$: the minimum number of elements of some basic types still required by the present elements,
- $\mathtt{M}$: the maximum number of elements of basic types still tolerated by the present elements,

checking that the maximum limit is never exceed and the minimum limit is reached in every compartment.

Types are assigned to patterns and terms with the typing rules in Figure 5.2, where bases, assigning types to element, term and sequence variables are defined by:

$$\Delta \ ::= \ \emptyset \ \ \Big| \ \ \Delta, x : (\{\mathtt{t}\}, \mathtt{D_t}, \mathtt{U_t}) \ \ \Big| \ \ \Delta, \rho : (\mathtt{P}, \mathtt{L}, \mathtt{M})$$

where $\rho$ denotes a sequence or term variable, i.e. $\rho \in \mathcal{TV} \cup \mathcal{SV}$. A basis is *well formed* if all types in the basis are well formed. Note that the definition of basis and the typing rules, and also other notions used in the remainder of the Chapter, are similar to the ones designed for the $\mathtt{P}/\mathtt{R}$ type discipline (see Figure 4.1).

The type of the empty sequence, ($\mathsf{Teps}$) rule, has the empty multiset as *present-ms*, because an empty sequence does not contain elements, the empty multiset as *at-least-ms* and $\Gamma_\infty$ as *at-most-ms*, because the absence of elements allows the absence of limits. The type of an element, ($\mathsf{Tel}$) rule, is composed by its basic type $\mathtt{t}$ as *present-ms*, and the *down-ms* and *up-ms* of $\mathtt{t}$. The type of a variable, ($\mathsf{Tvar}$) rule, is its type in the basis. The type of a sequence, a parallel composition or a looping sequence is derived from the types of their two sub-patterns. These sub-patterns must respect an obvious condition: the *present-ms* of the former must be a subset of the *at-most-ms* of the latter, i.e. the number of present elements in the former must not exceed the maximum number of the same elements tolerated by the latter, and vice versa. Note that, on the contrary, it is not necessary to check whether all

$$\Delta \vdash \epsilon : (\emptyset, \emptyset, \Gamma_\infty) \ (\mathsf{Teps}) \qquad \frac{a : \mathtt{t} \in \Psi}{\Delta \vdash a : (\{\mathtt{t}\}, \mathtt{D}_\mathtt{t}, \mathtt{U}_\mathtt{t})} \ (\mathsf{Tel})$$

$$\Delta, \chi : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash \chi : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \ (\mathsf{Tvar})$$

$$\frac{\Delta \vdash SP : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \quad \Delta \vdash SP' : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \quad (\mathtt{P}, \mathtt{L}, \mathtt{M}) \bowtie (\mathtt{P}', \mathtt{L}', \mathtt{M}')}{\Delta \vdash SP{\cdot}SP' : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \sqcup (\mathtt{P}', \mathtt{L}', \mathtt{M}')} \ (\mathsf{Tseq})$$

$$\frac{\Delta \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \quad \Delta \vdash P' : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \quad (\mathtt{P}, \mathtt{L}, \mathtt{M}) \bowtie (\mathtt{P}', \mathtt{L}', \mathtt{M}')}{\Delta \vdash P \mid P' : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \sqcup (\mathtt{P}', \mathtt{L}', \mathtt{M}')} \ (\mathsf{Tpar})$$

$$\frac{\Delta \vdash SP : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \quad \Delta \vdash P : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \quad (\mathtt{P}', \mathtt{L}', \mathtt{M}') \sqsubseteq (\mathtt{P}, \mathtt{L}, \mathtt{M})}{\Delta \vdash (SP)^{\circlearrowleft} \rfloor P : (\mathtt{P}, \mathtt{L} \ominus \mathtt{P}', \mathtt{M} \ominus \mathtt{P}')} \ (\mathsf{Tcomp})$$

Figure 5.2: Typing Rules

the minimum requests are reached, because this constraint depends on the elements in the whole compartment and the looping sequence containing it, if it exists: therefore this condition is checked only in the typing rule for a looping sequence.

Now we focus on (Tseq) and (Tpar) rules. The type of the obtained pattern is, like for the P/R type discipline, the join of the types $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ and $(\mathtt{P}', \mathtt{L}', \mathtt{M}')$ of the connected patterns defined as follows. But here the join operation is different from the one proposed in Definition 4.2.2. The *present-ms* of the join type is the sum of the *present-ms*, $\mathtt{P} \uplus \mathtt{P}'$, i.e. the number of one element in the new type is the sum of the numbers of the same element in the old types. For getting the *at-least-ms* of the join type we remove the *present-ms* of a type from the *at-least-ms* of the other one, obtaining for each type the number of elements required taking into account the presence of the basic types in the *present-ms* of the other type, and we consider the union of these multisets, for the same reason seen before for the well-formedness of types in Definition 5.2.3. For the *at-most-ms* of the join type we do the dual, taking the intersection of the removals, i.e. their minimum. To sum up:

**Definition 5.2.4 (Join of Types).** *Given two well-formed types* $(\mathtt{P}, \mathtt{L}, \mathtt{M})$

*and* $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$*, we define their* join $(\mathtt{P},\mathtt{L},\mathtt{M}) \sqcup (\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *by*

$$(\mathtt{P},\mathtt{L},\mathtt{M}) \sqcup (\mathtt{P}',\mathtt{L}',\mathtt{M}') = (\mathtt{P} \uplus \mathtt{P}', (\mathtt{L} \ominus \mathtt{P}') \cup (\mathtt{L}' \ominus \mathtt{P}), (\mathtt{M} \ominus \mathtt{P}') \cap (\mathtt{M}' \ominus \mathtt{P})).$$

The type obtained by join may be not well formed, because

1. its *at-least-ms* could not be a subset of its *at-most-ms*,
2. the multiplicity of a certain basic type in a present multiset could exceed the number of tolerated elements of the other type.

Since we want to restrict to well-formed types, as done in Definition 4.2.3, we define compatibility between types, that impose both conditions.

**Definition 5.2.5 (Type Compatibility).** *Two well-formed types* $(\mathtt{P},\mathtt{L},\mathtt{M})$ *and* $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *are* compatible *(written* $(\mathtt{P},\mathtt{L},\mathtt{M}) \bowtie (\mathtt{P}',\mathtt{L}',\mathtt{M}')$*) if* $(\mathtt{L}\ominus\mathtt{P}') \cup (\mathtt{L}'\ominus\mathtt{P}) \subseteq (\mathtt{M}\ominus\mathtt{P}') \cap (\mathtt{M}'\ominus\mathtt{P})$*,* $\mathtt{P} \subseteq \mathtt{M}'$ *and* $\mathtt{P}' \subseteq \mathtt{M}$*.*

Compatibility of two types is a necessary and sufficient condition to get well-formedness of the join.

**Proposition 5.2.6.** *Let* $(\mathtt{P},\mathtt{L},\mathtt{M})$*,* $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *be well-formed types:* $(\mathtt{P},\mathtt{L},\mathtt{M}) \sqcup (\mathtt{P}',\mathtt{L}',\mathtt{M}')$ *is well formed if and only if* $(\mathtt{P},\mathtt{L},\mathtt{M}) \bowtie (\mathtt{P}',\mathtt{L}',\mathtt{M}')$*.*

*Proof.* We have to show that $(\mathtt{L}\ominus\mathtt{P}') \cup (\mathtt{L}'\ominus\mathtt{P}) \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}\uplus\mathtt{P}'} \mathtt{D}_\mathtt{t}$ and $\mathtt{P}\uplus\mathtt{P}'\uplus[(\mathtt{M}\ominus\mathtt{P}') \cap (\mathtt{M}'\ominus\mathtt{P})] \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}\uplus\mathtt{P}'}(\mathtt{U}_\mathtt{t} \uplus \{\mathtt{t}\})$.
Since $(\mathtt{P},\mathtt{L},\mathtt{M})$ and $(\mathtt{P}',\mathtt{L}',\mathtt{M}')$ are well formed by hypothesis, we get $\mathtt{L} \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}} \mathtt{D}_\mathtt{t}$, $\mathtt{L}' \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}'} \mathtt{D}_\mathtt{t}$, $\mathtt{P}\uplus\mathtt{M} \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}}(\mathtt{U}_\mathtt{t}\uplus\{\mathtt{t}\})$, $\mathtt{P}'\uplus\mathtt{M}' \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}'}(\mathtt{U}_\mathtt{t}\uplus\{\mathtt{t}\})$. Then $(\mathtt{L}\ominus\mathtt{P}') \cup (\mathtt{L}'\ominus\mathtt{P}) \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}\uplus\mathtt{P}'} \mathtt{D}_\mathtt{t}$ follows from $\mathtt{L} \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}} \mathtt{D}_\mathtt{t}$ and $\mathtt{L}' \subseteq \bigcup_{\mathtt{t}\in\mathtt{P}'} \mathtt{D}_\mathtt{t}$.
We have $\mathtt{P}\uplus\mathtt{M} = \mathtt{P}\uplus\mathtt{M}\uplus\mathtt{P}'\ominus\mathtt{P}' \supseteq \mathtt{P}\uplus\mathtt{P}'\uplus[(\mathtt{M}\ominus\mathtt{P}')\cap(\mathtt{M}'\ominus\mathtt{P})]$ since $\mathtt{P}' \subseteq \mathtt{M}$, which implies $\mathtt{P}\uplus\mathtt{P}'\uplus[(\mathtt{M}\ominus\mathtt{P}')\cap(\mathtt{M}'\ominus\mathtt{P})] \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}}(\mathtt{U}_\mathtt{t}\uplus\{\mathtt{t}\})$ by $\mathtt{P}\uplus\mathtt{M} \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}}(\mathtt{U}_\mathtt{t}\uplus\{\mathtt{t}\})$.
Similarly we can show $\mathtt{P}\uplus\mathtt{P}'\uplus[(\mathtt{M}\ominus\mathtt{P}')\cap(\mathtt{M}'\ominus\mathtt{P})] \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}'}(\mathtt{U}_\mathtt{t}\uplus\{\mathtt{t}\})$, so we conclude $\mathtt{P}\uplus\mathtt{P}'\uplus[(\mathtt{M}\ominus\mathtt{P}')\cap(\mathtt{M}'\ominus\mathtt{P})] \subseteq \bigcap_{\mathtt{t}\in\mathtt{P}\uplus\mathtt{P}'}(\mathtt{U}_\mathtt{t}\uplus\{\mathtt{t}\})$.
Note that if $(\mathtt{L}\ominus\mathtt{P}') \cup (\mathtt{L}'\ominus\mathtt{P}) \not\subseteq (\mathtt{M}\ominus\mathtt{P}') \cap (\mathtt{M}'\ominus\mathtt{P})$ the join type is clearly not well formed. If $\mathtt{P} \not\subseteq \mathtt{M}'$ it means that there are basic types $\mathtt{t} \in \mathtt{P}$, $\mathtt{t}' \in \mathtt{P}'$ such that the number of present elements of basic type $\mathtt{t}$ is bigger than the number of elements of basic type $\mathtt{t}$ allowed by the elements of basic type $\mathtt{t}'$, taking into account also the elements of basic type $\mathtt{t}$ which belongs to $\mathtt{P}'$ or possibly to an inner compartment. Therefore the joined type will be not well formed. □

Finally, we consider the (Tcomp) rule. In the resulting type, the present elements are only the ones in the looping sequence, because a looping sequence makes the elements inside the compartment invisible from outside. Since the elements in the looping sequence are influenced by the ones inside the compartment, to obtain the *at-least-ms* and *at-most-ms* of its type we must subtract the elements present in the inner pattern from the *at-least-ms* and *at-most-ms* of the looping sequence.

**Definition 5.2.7 (Subtraction of Types).** *Given two well-formed types* $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ *and* $(\mathtt{P}', \mathtt{L}', \mathtt{M}')$*, we define their* subtraction *as* $(\mathtt{P}, \mathtt{L} \ominus \mathtt{P}', \mathtt{M} \ominus \mathtt{P}')$.

A type obtained by subtraction is always well formed, because we are taking away the same multiset from the *at-least-ms* and *at-most-ms* of a well-formed type. Since a looping sequence encloses a compartment, we must assure that all the at-least limits in the inner compartment are reached. To do so, we require that the *present-ms* of the looping sequence satisfies all the requests of the *at-least-ms* of the inner pattern. As seen for compatibility, we must also check that the *present-ms* of a type is a sub-multiset of the *at-most-ms* of the other type.

**Definition 5.2.8 (Types Satisfaction).** *Given two well-formed types* $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ *and* $(\mathtt{P}', \mathtt{L}', \mathtt{M}')$*,* $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ satisfies $(\mathtt{P}', \mathtt{L}', \mathtt{M}')$ *(written* $(\mathtt{P}', \mathtt{L}', \mathtt{M}') \sqsubseteq (\mathtt{P}, \mathtt{L}, \mathtt{M})$*) if* $\mathtt{L}' \subseteq \mathtt{P}$*,* $\mathtt{P} \subseteq \mathtt{M}'$ *and* $\mathtt{P}' \subseteq \mathtt{M}$.

It is easy to verify that, using the typing rules in Figure 5.2, from the empty basis we derive a well-formed type for a term, and from a well-formed basis we derive a well-formed type for a pattern.

Reduction rules are applied only to terms such that their types are well formed and the at-least limits are reached also in the outermost compartment, i.e. their types have the empty multiset as *at-least-ms*. These terms are interesting from a biological point of view because, since all the minimum requests are fulfilled, they represent complete systems. As done for $\mathtt{P}/\mathtt{R}$ type discipline in Definition 4.2.5, we call them *correct terms*:

**Definition 5.2.9 (Correct Terms).** *A term* $\vdash T : (\mathtt{P}, \mathtt{L}, \mathtt{M})$ *is* correct *if* $\mathtt{L} = \emptyset$.

For the sake of clarity, in the following examples a multiset $A$ will be denoted with the set notation by listing the types followed by their multiplicity, $\{\mathtt{t} : f_A(\mathtt{t}) \mid \mathtt{t} \in D\}$, where $\mathtt{t}_1, \mathtt{t}_2, \ldots, \mathtt{t}_k : m$ means that the basic types $\mathtt{t}_1, \mathtt{t}_2, \ldots, \mathtt{t}_k$ have multiplicity $m$. In the *at-most-ms* we write only the basic types having multiplicity 0 or finite, and we do not write the basic types having multiplicity $\infty$. On the contrary, in *present-ms* and *at-least-ms* we do not write the basic types having multiplicity 0. In this way, we highlight only the most significant cases: in fact, an infinite multiplicity in a *at-most-ms* means no constraint, and the same for a multiplicity of zero in a *at-least-ms*.

**Example 5.2.10.** *Assuming the set of basic types*

$$\Gamma = \{\mathtt{t}_a, \mathtt{t}_b, \mathtt{t}_c, \mathtt{t}_d\}$$

*and a classification which contains*

$$\{a : \mathtt{t}_a, b : \mathtt{t}_b, c : \mathtt{t}_c, d : \mathtt{t}_d\}$$

*where* `mn`*,* `mx` *are:*

| mn | $\mathtt{t}_a$ | $\mathtt{t}_b$ | $\mathtt{t}_c$ | $\mathtt{t}_d$ |
|---|---|---|---|---|
| $\mathtt{t}_a$ | 0 | 0 | 1 | 0 |
| $\mathtt{t}_b$ | 0 | 0 | 0 | 0 |
| $\mathtt{t}_c$ | 2 | 0 | 0 | 0 |
| $\mathtt{t}_d$ | 0 | 0 | 0 | 0 |

| mx | $\mathtt{t}_a$ | $\mathtt{t}_b$ | $\mathtt{t}_c$ | $\mathtt{t}_d$ |
|---|---|---|---|---|
| $\mathtt{t}_a$ | $\infty$ | $\infty$ | 1 | $\infty$ |
| $\mathtt{t}_b$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| $\mathtt{t}_c$ | $\infty$ | 1 | $\infty$ | $\infty$ |
| $\mathtt{t}_d$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |

*the term*

$$(A) \ \vdash (d)^{\circlearrowleft} \rfloor (c \mid a \mid a \mid a \mid a) : (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$$

*is correct, while the term*

$$\vdash a \mid \ (d)^{\circlearrowleft} \rfloor (c \mid a \mid a \mid a \mid a) : (\{\mathtt{t}_a, \mathtt{t}_d : 1\}, \{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 1\})$$

*is not correct, because the term of a basic type* $\mathtt{t}_a$ *requires exactly one element of basic type* $\mathtt{t}_c$*. Note that the term*

$$a \mid \ (d)^{\circlearrowleft} \rfloor (c \mid a)$$

*cannot have a type, because in the inner compartment containing an element of basic type* $\mathtt{t}_c$ *there are less than two elements of basic type* $\mathtt{t}_a$*.*
*Another untypable term is*

$$a \mid \ (d)^{\circlearrowleft} \rfloor (c \mid a \mid a \mid b \mid b)$$

*because in the same compartment or looping sequence containing an element of basic type* $\mathtt{t}_c$*, there are more than one element of basic type* $\mathtt{t}_b$*.*
*Finally, the term*

$$a \mid \ (d)^{\circlearrowleft} \rfloor (a \mid a \mid a)$$

*does not have a type, because there are elements of basic type* $\mathtt{t}_a$ *without any element of basic type* $\mathtt{t}_c$*.*

In the remaining of the present section we will define our typed semantics, and show that typed reductions preserve the correctness of terms. First of all, we need to adapt the Definition 4.2.8 to our type discipline.

**Definition 5.2.11.** *An instantiation $\sigma$* agrees *with a basis $\Delta$ (notation $\sigma \in \Sigma_\Delta$) if $\chi : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \in \Delta$ implies $\vdash \sigma(\chi) : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.*

The following Lemma, similar to Lemma 4.2.9 and that can be easily proved like it by induction on type derivations, will be useful for the Subject Reduction Theorem:

**Lemma 5.2.12.** *If $\sigma \in \Sigma_\Delta$, then $\vdash P\sigma : (\mathtt{P}, \mathtt{L}, \mathtt{M})$ if and only if $\Delta \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.*

We are looking for a typed semantics which applied to correct terms produces only correct terms. Note that, like in Definition 4.2.11, if $X : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash C[X] : (\mathtt{P}', \emptyset, \mathtt{M}')$, then every term obtained filling the hole of this evaluation context with a term having type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ will be correct. This fact leads us to the following definition:

**Definition 5.2.13 (Typed Hole).** *Given an evaluation context $E$ and a well-formed type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$, the type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is $OK$ for the context $E$ if $X : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash E[X] : (\mathtt{P}', \emptyset, \mathtt{M}')$ for some $\mathtt{P}'$, $\mathtt{M}'$.*

For rewriting rules, we are only interested in the types of the right-hand-sides, since they influence the type of the obtained term. For this reason, we define the $\Delta$-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safeness for rules, similar to the $\Delta$-$(\mathtt{P}, \mathtt{R})$-safeness of Definition 4.2.12.

**Definition 5.2.14 ($\Delta$-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe Rules).** *A rewrite rule $P_1 \mapsto P_2$ is $\Delta$-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$*-safe *if $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.*

**Example 5.2.15.** *Assuming the set of basic types and the classification in Example 5.2.10 and the basis*

$$\Delta = \{X : (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})\}$$

*the rule*

$$(d)^{\circlearrowleft} \rfloor (b \mid X) \mapsto b \mid \ (d)^{\circlearrowleft} \rfloor X$$

*is a $\Delta$-$(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$-safe rule. In fact we derive*

$$\Delta \vdash b \mid \ (d)^{\circlearrowleft} \rfloor X : (\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty).$$

Using Definitions 5.2.13 and 5.2.14 we derive that if we apply a rule whose right-hand-side has type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$, and this type is $OK$ for the context, we obtain a correct term. For this reason, these definitions are used for the typed semantics.

| | $P_1$ | $P_2$ | $X$ | $E$ |
|---|---|---|---|---|
| $(A)$ | $a$ | $b$ | — | $(d)^{\circlearrowleft} \rfloor (c \mid a \mid a \mid a \mid \square)$ |
| $(B)$ | $(d)^{\circlearrowleft} \rfloor (b \mid X)$ | $b \mid (d)^{\circlearrowleft} \rfloor X$ | $c \mid a \mid a \mid a$ | $\square$ |
| $(C)$ | $a$ | $b$ | — | $b \mid (d)^{\circlearrowleft} \rfloor (c \mid a \mid a \mid \square)$ |
| $(D)$ | $(d)^{\circlearrowleft} \rfloor (b \mid X)$ | $b \mid (d)^{\circlearrowleft} \rfloor X$ | $c \mid a \mid a$ | $b \mid \square$ |

Figure 5.3: Rules, Instantiations and Contexts of Example 5.2.18

| | $\Delta$ | type of $\mathtt{P}_2\sigma$ | type of $E[\mathtt{P}_2\sigma]$ |
|---|---|---|---|
| $(A)$ | — | $(\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty)$ | $(\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$ |
| $(B)$ | $X : (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$ | $(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$ | $(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$ |
| $(C)$ | — | $(\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty)$ | $(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$ |
| $(D)$ | $X : (\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$ | $(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$ | $(\{\mathtt{t}_b : 2, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$ |

Figure 5.4: Basis and Typings of Example 5.2.18

**Definition 5.2.16 (Typed Semantics).** *Given a finite set of rewriting rules $\mathcal{R}$, the* typed semantics *of* CLS *is the least relation closed with respect to $\equiv$ and satisfying the following sets of rules:*

$$\frac{\Re = P_1 \mapsto P_2 \in \mathcal{R} \text{ is a } \Delta\text{-}(\mathtt{P},\mathtt{L},\mathtt{M})\text{-safe rule} \qquad P_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma_\Delta \qquad E \in \mathcal{E} \qquad (\mathtt{P},\mathtt{L},\mathtt{M}) \text{ is } OK \text{ for } E}{E[P_1\sigma] \Longrightarrow E[P_2\sigma]}$$

Note that our typed semantics is similar to the second one proposed in Definition 4.2.14, and like this one (see 4.2.15) it preserves correctness.

**Theorem 5.2.17 (Subject Reduction).** *If $T \Longrightarrow T'$, then $\Delta \vdash T' : (\mathtt{P}', \emptyset, \mathtt{M}')$ for some $\mathtt{P}'$, $\mathtt{M}'$.*

*Proof.* From Definition 5.2.16, we have that $T'$ is $E[P_2\sigma]$, and, from Definition 5.2.14, we have that $\Delta \vdash P_2 : (\mathtt{P},\mathtt{L},\mathtt{M})$ for some $\Delta$, $\mathtt{P}$, $\mathtt{L}$, $\mathtt{M}$. Lemma 5.2.12 and $\sigma \in \Sigma_\Delta$ imply that $\vdash P_2\sigma : (\mathtt{P},\mathtt{L},\mathtt{M})$. Since, from Definition 5.2.16, the type $(\mathtt{P},\mathtt{L},\mathtt{M})$ is $OK$ for $E$, we conclude that $\vdash E[P_2\sigma] : (\mathtt{P}', \emptyset, \mathtt{M}')$ for some $\mathtt{P}'$, $\mathtt{M}'$. $\square$

**Example 5.2.18.** *We study the behavior of the term $(A)$ in Example 5.2.10 using the rules*

$$(1)\ a \mapsto b \quad \text{and} \quad (2)\ (d)^{\circlearrowleft} \rfloor (b \mid X) \mapsto b \mid (d)^{\circlearrowleft} \rfloor X.$$

*Rules, instantiations and contexts for the reductions are reported in Figure*

*5.3, and their basis and typings are reported in Figure 5.4.*
*On term (A) of Example 5.2.10, we can only apply rule (1), obtaining the correct term*

$$(B)\ \vdash (d)^{\circlearrowleft} \rfloor (c \mid a \mid a \mid a \mid b) : (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty).$$

*On term (B), we cannot apply rule (1), because the basic type of c allows no more than one element having basic type $\mathtt{t}_b$, and the term derived from the application of this rule would not respect this constraint. We can only apply rule (2), obtaining the correct term*

$$(C)\ \vdash b \mid (d)^{\circlearrowleft} \rfloor (c \mid a \mid a \mid a) : (\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty).$$

*On term (C) , we can only apply rule (1), obtaining the correct term*

$$(D)\ \vdash b \mid (d)^{\circlearrowleft} \rfloor (c \mid a \mid a \mid b) : (\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty).$$

*For the same reason as for the term (B), on the term (D) we can apply only rule (2), obtaining the correct term*

$$(E)\ \vdash b \mid b \mid (d)^{\circlearrowleft} \rfloor (c \mid a \mid a) : (\{\mathtt{t}_b : 2, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty).$$

*Rule (1) cannot be applied, because the basic type of c needs at least two elements having basic type $\mathtt{t}_a$, and the term derived from the application of this rule does not respect this constraint. Since also rule (2) cannot be applied, the term (E) cannot reduce using rules (1) and (2).*

We end this section by stating two lemmas on properties of typing rules which will be used to show the theorems of the next section.

**Lemma 5.2.19 (Weakening).** *If $\Delta \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$ and $\Delta \subseteq \Delta'$, then $\Delta' \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.*

*Proof.* By induction on derivations. □

**Lemma 5.2.20.** *If $\Delta \vdash E[P] : (\mathtt{P}, \mathtt{L}, \mathtt{M})$ then*

1. *$\Delta \vdash P : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$ for some $(\mathtt{P}', \mathtt{L}', \mathtt{M}')$, and*

2. *if $P'$ is such that $\Delta \vdash P' : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$, then $\Delta \vdash E[P'] : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.*

*Proof.* By induction on contexts. □

## 5.3 Inference

We use the machinery of principal typing [67] to infer the $OK$ relation between types and contexts and which rules are $\Delta$-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe. From the term we want to reduce we obtain a set of constraints: if they are fulfilled by a rule, then the rule can be applied to the term preserving correctness. In this way, we can decide the applicability of the rules.

Similarly to Section 4.3, we convene that for each element variable $x$ there is an *e-type* variable $\eta_x$ ranging over basic types, and for each term or sequence variable $\rho \in \mathcal{TV} \cup \mathcal{SV}$ there are three variables $\pi_\rho$, $\lambda_\rho$, $\mu_\rho$ (called *p-type* variable, *l-type* variable and *m-type* variable) ranging over multisets of basic types. Moreover we convene that $\Pi$ ranges over formal unions of multisets of basic types, *e-type* variables and *p-type* variables, $\Lambda$ ranges over unions of multisets of basic types and *l-type* variables, and $\Omega$ ranges over unions of multisets of basic types and *m-type* variables. We denote by $\delta$ a generic *p-type*, *l-type*, *m-type* or *e-type* variable.

A *basis scheme* $\Theta$ is a mapping from atomic variables to their *e-type* variables, and from sequence and term variables to triples of their *p-type* variables, *l-type* variables and *m-type* variables:

$$\Theta \ ::= \ \emptyset \ \Big| \ \Theta, x : \eta_x \ \Big| \ \Theta, \rho : (\pi_\rho, \lambda_\rho, \mu_\rho).$$

The rules for inferring principal typings use judgments of the shape:

$$\vdash P : \Theta; (\Pi, \Lambda, \Omega); \Xi$$

where $\Theta$ is the *principal basis* in which $P$ is well formed, $(\Pi, \Lambda, \Omega)$ is the *principal type* of $P$, and $\Xi$ is the set of constraints that should be satisfied. Figure 5.5 gives these inference rules, derived from the typing rules in Figure 5.2 and similar the inference rules in Figure 4.3.

Rules ($\mathsf{Reps}$) and ($\mathsf{Rel}$) directly derive from rules ($\mathsf{Teps}$) and ($\mathsf{Tel}$). The rules for typing variables (rules ($\mathsf{Rvar}_1$) and ($\mathsf{Rvar}_2$)) put the variable with its type in the basis. In rules ($\mathsf{Rseq}$), ($\mathsf{Rpar}$) and ($\mathsf{Rcomp}$), the principal type is derived as in ($\mathsf{Tseq}$), ($\mathsf{Tpar}$) and ($\mathsf{Tcomp}$) rules respectively. The set of constraints is the union between the constraints in the premise of the rule itself and the constraints in the premise of ($\mathsf{Tseq}$), ($\mathsf{Tpar}$) and ($\mathsf{Tcomp}$) rules, respectively. The principal basis is the union of the principal bases of the composing patterns, without renaming, because each variable $\rho$ or $x$ is associated to an unique triple of *p-type*, *l-type*, *m-type* variables or to an unique *e-type* variable, respectively.

The key difference between inference rules, in Figure 5.5, and typing rules, in Figure 5.2, is that the conditions of type compatibility and type satisfaction are not premises, but conclusions. In this way, at the end of inference all these conditions create a set of constraints, that must be checked to decide the applicability of rules.

$$\vdash \epsilon : \emptyset; (\emptyset, \emptyset, \infty); \emptyset \quad (\mathsf{Reps}) \quad \vdash x : \{x : (\{\eta_x\}, \mathtt{D}_{\eta_x}, \mathtt{U}_{\eta_x})\}; (\{\eta_x\}, \mathtt{D}_{\eta_x}, \mathtt{U}_{\eta_x}); \emptyset \quad (\mathsf{Rvar}_1)$$

$$\vdash \rho : \{\rho : (\pi_\rho, \lambda_\rho, \mu_\rho)\}; (\pi_\rho, \lambda_\rho, \mu_\rho); \emptyset \quad (\mathsf{Rvar}_2) \qquad \frac{a : \mathtt{t} \in \Psi}{\vdash a : \emptyset; (\{\mathtt{t}\}, \mathtt{D}_{\mathtt{t}}, \mathtt{U}_{\mathtt{t}}); \emptyset}\ (\mathsf{Rel})$$

$$\frac{\vdash SP : \Theta; (\Pi, \Lambda, \Omega); \Xi \quad \vdash SP' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'}{\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Pi, \Lambda, \Omega) \sqcup (\Pi', \Lambda', \Omega'); \Xi \cup \Xi' \cup \{(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')\}}\ (\mathsf{Rseq})$$

$$\frac{\vdash P : \Theta; (\Pi, \Lambda, \Omega); \Xi \quad \vdash P' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'}{\vdash P \mid P' : \Theta \cup \Theta'; (\Pi, \Lambda, \Omega) \sqcup (\Pi', \Lambda', \Omega'); \Xi \cup \Xi' \cup \{(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')\}}\ (\mathsf{Rpar})$$

$$\frac{\vdash SP : \Theta; (\Pi, \Lambda, \Omega); \Xi \quad \vdash P' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'}{\vdash (SP)^{\circlearrowleft} \rfloor P : \Theta \cup \Theta'; (\Pi, \Lambda \ominus \Pi', \Omega \ominus \Pi'); \Xi \cup \Xi' \cup \{(\Pi', \Lambda', \Omega') \sqsubseteq (\Pi, \Lambda, \Omega)\}}\ (\mathsf{Rcomp})$$

Figure 5.5: Inference Rules

**Example 5.3.1.** *We can use the inference rules in Figure 5.5 to infer the types of the right-side patterns of the rules in Example 5.2.18, where, again, we assume the set of basic types and the classification of Example 5.2.10, obtaining*

$$\begin{array}{l} \vdash b : \emptyset; (\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty); \emptyset \\ \vdash b \mid (d)^{\circlearrowleft} \rfloor X : \Theta; (\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty) \sqcup (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty); \Xi \end{array}$$

*where*

$$\begin{array}{rcl} \Theta & = & \{\ X : (\pi_X, \lambda_X, \mu_X)\ \} \\ \Xi & = & \{\ \ (\pi_X, \lambda_X, \mu_X) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty), \\ & & (\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty) \bowtie (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)\ \ \} \end{array}$$

Like in Theorems 4.3.2 and 4.3.3 for `P/R` type discipline, we can prove soundness and completeness of our inference rules. A *type mapping* maps *e-type* variables to basic types, *p-type* variables, *l-type* variables and *m-type* variables to multisets of basic types. A type mapping $m$ *satisfies* a set of constraints $\Xi$ if all constraints in $m(\Xi)$ are satisfied.

**Theorem 5.3.2 (Soundness of Type Inference).** *If* $\vdash P : \Theta; (\Pi, \Lambda, \Omega); \Xi$ *and* $m$ *is a type mapping which satisfies* $\Xi$, *then* $m(\Theta) \vdash P : (m(\Pi), m(\Lambda), m(\Omega))$.

*Proof.* By induction on derivations, and by cases on the last applied rule.

- For rules ($\mathsf{Reps}$), ($\mathsf{Rel}$), ($\mathsf{Rvar}_1$), and ($\mathsf{Rvar}_2$) the result is trivial.

- Rule ($\mathsf{Rseq}$). In this case the conclusion of the rule is
$$\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Pi, \Lambda, \Omega) \sqcup (\Pi', \Lambda', \Omega'); \Xi \cup \Xi' \cup \{(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')\}$$
and the assumptions are
$$\vdash SP : \Theta; (\Pi, \Lambda, \Omega); \Xi \text{ and } \vdash SP' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'.$$
Since $m$ satisfies $\Xi$ and $\Xi'$, by induction hypothesis, and weakening (Lemma 5.2.19), we derive
$$m(\Theta \cup \Theta') \vdash SP : (m(\Pi), m(\Lambda), m(\Omega))$$
$$m(\Theta \cup \Theta') \vdash SP' : (m(\Pi'), m(\Lambda'), m(\Omega')).$$
Moreover, since $m$ satisfies $(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')$, we have
$$(m(\Pi), m(\Lambda), m(\Omega)) \bowtie (m(\Pi'), m(\Lambda'), m(\Omega')).$$
Therefore the rule ($\mathsf{Tseq}$) can be applied, and
$$m(\Theta \cup \Theta') \vdash SP{\cdot}SP' : (m(\Pi), m(\Lambda), m(\Omega)) \sqcup (m(\Pi'), m(\Lambda'), m(\Omega')).$$

- For rules ($\mathsf{Rpar}$), and ($\mathsf{Rcomp}$) the result can be proved like for rule ($\mathsf{Rseq}$).

□

**Theorem 5.3.3 (Completeness of Type Inference).** *If* $\Delta \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$, *then* $\vdash P : \Theta; (\Pi, \Lambda, \Omega); \Xi$ *for some* $\Theta$, $\Pi$, $\Lambda$, $\Omega$, $\Xi$ *and there is a type mapping* $m$ *that satisfies* $\Xi$ *and such that* $\Delta \supseteq m(\Theta)$, $\mathtt{P} = m(\Pi)$, $\mathtt{L} = m(\Lambda)$, $\mathtt{M} = m(\Omega)$.

*Proof.* By induction on the derivation of $\Delta \vdash P : (\mathtt{P}, \mathtt{L}, \mathtt{M})$.

- If the last rule of the derivation is ($\mathsf{Teps}$), ($\mathsf{Tel}$), or ($\mathsf{Tvar}$) the result is obvious. Note that, for ($\mathsf{Tvar}$) in the inference we distinguish the case of element variables from sequence or term variables.

- Rule ($\mathsf{Tseq}$). The conclusion of the rule is
$$\Delta \vdash SP{\cdot}SP' : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \sqcup (\mathtt{P}', \mathtt{L}', \mathtt{M}'),$$
and the assumptions are
$$\Delta \vdash SP : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \quad \Delta \vdash SP' : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$$
and the condition $(\mathtt{P}', \mathtt{L}', \mathtt{M}') \bowtie (\mathtt{P}', \mathtt{L}', \mathtt{M}')$. By induction hypothesis, there are $\Theta$, $\Pi$, $\Lambda$, $\Omega$, $\Xi$, $\Theta'$, $\Pi'$, $\Lambda'$, $\Omega'$, $\Xi'$ such that
$$\vdash SP : \Theta; (\Pi, \Lambda, \Omega); \Xi \text{ and } \vdash SP' : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'.$$
These are the assumptions of rule ($\mathsf{Rseq}$), whose conclusion is
$$\vdash SP{\cdot}SP' : \Theta \cup \Theta'; (\Pi, \Lambda, \Omega) \sqcup (\Pi', \Lambda', \Omega'); \Xi \cup \Xi' \cup \{(\Pi, \Lambda, \Omega) \bowtie (\Pi', \Lambda', \Omega')\}.$$
Moreover, by induction there is a type mapping $m'$ satisfying $\Xi$ such

that $\Delta \supseteq m'(\Theta)$, $\mathtt{P} = m'(\Pi)$, $\mathtt{L} = m'(\Lambda)$ and $\mathtt{M} = m'(\Omega)$, and there is a type mapping $m''$ satisfying $\Xi'$ such that $\Delta \supseteq m''(\Theta')$, $\mathtt{P}' = m''(\Pi')$, $\mathtt{L}' = m''(\Lambda')$ and $\mathtt{M}' = m''(\Omega')$. Therefore, we derive $\Delta \supseteq m'(\Theta) \cup m''(\Theta')$ and $(\mathtt{P},\mathtt{L},\mathtt{M}) \sqcup (\mathtt{P}',\mathtt{L}',\mathtt{M}') = (m'(\Pi), m'(\Lambda), m'(\Omega)) \sqcup (m''(\Pi'), m''(\Lambda'), m''(\Omega'))$. Since the bases $m'(\Theta)$ and $m''(\Theta')$ are both subsets of the same basis $\Delta$, then for all the (*e-type*, *p-type*, *l-type* or *m-type*) variables $\delta$ such that $\delta \in dom(m') \cap dom(m'')$ we get $m'(\delta) = m''(\delta)$. Therefore the mapping $m$

$$m(\delta) = \begin{cases} m'(\delta) & \text{if } \delta \in dom(m') \\ m''(\delta) & \text{if } \delta \in dom(m'') \end{cases}$$

is well defined.
Moreover, since $m$ satisfies $\Xi$, $\Xi'$ and $(\Pi,\Lambda,\Omega) \bowtie (\Pi',\Lambda',\Omega')$, then $m$ satisfies also all the constraints of the conclusion of the rule (Rseq).

- If the last rule is (Tpar) or (Tcomp) the proof is similar.

□

We use the inference rules to decide applicability of typed reduction rules for $\Delta$-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe rules. The first step is to understand when a type mapping makes a rule $\Delta$-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe, i.e. when it satisfies the constraints in Definition 5.2.14.

**Lemma 5.3.4 (Characterization of $\Delta$-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe rules).** *A rule $P_1 \mapsto P_2$ is $\Delta$-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe if and only if the type mapping $m$ defined by the basis $\Delta$, i.e. such that*

- $m(\eta_x) = \mathtt{t}$ *if* $\Delta(x) = \{\mathtt{t}\}$
- $m(\pi_\rho) = \mathtt{P}'$ *if* $\Delta(\rho) = (\mathtt{P}',\mathtt{L}',\mathtt{M}')$
- $m(\lambda_\rho) = \mathtt{L}'$ *if* $\Delta(\rho) = (\mathtt{P}',\mathtt{L}',\mathtt{M}')$
- $m(\mu_\rho) = \mathtt{M}'$ *if* $\Delta(\rho) = (\mathtt{P}',\mathtt{L}',\mathtt{M}')$

*satisfies the set of constraints $\Xi_2 \cup \{\Pi_2 = \mathtt{P}\} \cup \{\Lambda_2 = \mathtt{L}\} \cup \{\Omega_2 = \mathtt{M}\}$, where $\vdash P_2 : \Theta_2; (\Pi_2,\Lambda_2,\Omega_2); \Xi_2$.*

*Proof.* ($\Leftarrow$) Let $\vdash P_2 : \Theta_2; (\Pi_2,\Lambda_2,\Omega_2); \Xi_2$ and $m$ satisfies $\Xi_2 \cup \{\Pi_2 = \mathtt{P}\} \cup \{\Lambda_2 = \mathtt{L}\} \cup \{\Omega_2 = \mathtt{M}\}$. From Theorem 5.3.2 we derive that $m(\Theta_2) \vdash P_2 : (\mathtt{P},\mathtt{L},\mathtt{M})$. By definition of $m$ we get $m(\Theta_2) = \Delta$. Therefore $\Delta \vdash P_2 : (\mathtt{P},\mathtt{L},\mathtt{M})$, and $P_1 \mapsto P_2$ is a $\Delta$-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe rule.

($\Rightarrow$) Let $P_1 \mapsto P_2$ be a $\Delta$-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe rule, then $\Delta \vdash P_2 : (\mathtt{P}, \mathtt{L}, \mathtt{M})$. From Theorem 5.3.3, we have that $\vdash P_2 : \Theta_2; (\Pi_2, \Lambda_2, \Omega_2); \Xi_2$, for some $\Theta_2, \Pi_2, \Lambda_2, \Omega_2, \Xi_2$, and there is a type mapping $m'$ satisfying $\Xi_2$ such that $\Delta \supseteq m'(\Theta_2)$, $\mathtt{P} = m'(\Pi_2)$, $\mathtt{L} = m'(\Lambda_2)$, and $\mathtt{M} = m'(\Omega_2)$. Therefore $m'$ satisfies $\Xi_2 \cup \{\Pi_2 = \mathtt{P}\} \cup \{\Lambda_2 = \mathtt{L}\} \cup \{\Omega_2 = \mathtt{M}\}$. From definition of $m$, we get $m(\Theta_2) = \Delta$, and since $\Delta \supseteq m'(\Theta_2)$, also $m$ satisfies $\Xi_2 \cup \{\Pi_2 = \mathtt{P}\} \cup \{\Lambda_2 = \mathtt{L}\} \cup \{\Omega_2 = \mathtt{M}\}$.

□

We can apply $\Delta$-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe rules only in contexts in which the type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is $OK$, so we must characterize also the $OK$ relation. To check this relation it is not necessary to consider the whole context, but only the part of the context influenced by the typing of the hole, given in Definition 4.3.7 as *core of the evaluation context*. Thanks to this notion, we can characterize the $OK$ relation using a shorter number of constraints.

**Lemma 5.3.5 (Characterization of $OK$ Relation).** *Let the evaluation context $E$ be such that $\vdash E[T] : (\mathtt{P}_0, \emptyset, \mathtt{M}_0)$ for some $T$, $\mathtt{P}_0$, $\mathtt{M}_0$. A type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is OK for $E$ if and only if the type mapping $m$ defined by*

- $m(\pi_X) = \mathtt{P}$,
- $m(\lambda_X) = \mathtt{L}$,
- $m(\mu_X) = \mathtt{M}$,

*satisfies the set of constraints*

$$\Xi \cup \{\Lambda = \emptyset \textit{ if } \mathsf{core}(E) = E\},$$

*where* $\vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \lambda_X, \mu_X)\}; (\Pi, \Lambda, \Omega); \Xi$.

*Proof.* ($\Leftarrow$) Lemma 5.2.20.(1) and $\vdash E[T] : (\mathtt{P}_0, \emptyset, \mathtt{M}_0)$ imply that all subterms of $\mathsf{core}(E)[X]$ are typable, i.e. that there are $\mathtt{P}_1$, $\mathtt{L}_1$, $\mathtt{M}_1$, $\mathtt{P}'_1$, $\mathtt{L}'_1$, $\mathtt{M}'_1$, $\mathtt{P}_2$, $\mathtt{L}_2$, $\mathtt{M}_2$, $\mathtt{P}'_2$, $\mathtt{L}'_2$, $\mathtt{M}'_2$ such that $\vdash T_1 : (\mathtt{P}_1, \mathtt{L}_1, \mathtt{M}_1)$, $\vdash S_1 : (\mathtt{P}'_1, \mathtt{L}'_1, \mathtt{M}'_1)$, $\vdash T_2 : (\mathtt{P}_2, \mathtt{L}_2, \mathtt{M}_2)$, $\vdash S_2 : (\mathtt{P}'_2, \mathtt{L}'_2, \mathtt{M}'_2)$ in the last case of the definition of $\mathsf{core}(E)[X]$, and suitable subsets of these typing judgments in the other two cases.

By Definition 4.3.7 we have the following cases.

- $E = \mathsf{core}(E)$
  Since $m$ satisfies $\{\Lambda = \emptyset$ if $\mathsf{core}(E) = E\}$, then $m(\Lambda) = \emptyset$. From Theorem 5.3.2, since $\vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \lambda_X, \mu_X)\}; (\Pi, \Lambda, \Omega); \Xi$ and $m$ satisfies $\Xi$, we derive that $X : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash \mathsf{core}(E)[X] : (m(\Pi), \emptyset, m(\Omega))$. Moreover, since $E[X] = \mathsf{core}(E)[X]$, we have that $X : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash E[X] : (m(\Pi), \emptyset, m(\Omega))$. Therefore, the type $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is OK for the context $E$.
- $\mathsf{core}(E) = (S_2)^{\circlearrowleft} \rfloor ((S_1)^{\circlearrowleft} \rfloor (\square \mid T_1) \mid T_2)$ and $\Pi = \mathtt{P}'_2$, $\Lambda = (\mathtt{L}'_2 \ominus (\mathtt{P}'_1 \uplus \mathtt{P}_2))$, and $\Omega = (\mathtt{M}'_2 \ominus (\mathtt{P}'_1 \uplus \mathtt{P}_2))$.
  Since the inferred type of $\mathsf{core}(E)$ does not depend by the type of the hole, and $\vdash E[T] : (\mathtt{P}_0, \emptyset, \mathtt{M}_0)$ for some $T$, then every type satisfying the constraints of $\mathsf{core}(E)$ is OK for $E$. Since $m$ satisfies $\Xi$, we conclude that $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is OK for the context $E$.

$(\Rightarrow)$ By Definition 5.2.13, since $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ is OK for $E$, then $X : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash E[X] : (\mathtt{P}', \emptyset, \mathtt{M}')$ for some $\mathtt{P}'$, $\mathtt{M}'$. Theorem 5.3.3 implies that $\vdash E[X] : \Theta'; (\Pi', \Lambda', \Omega'); \Xi'$ and there is a type mapping $m$ that satisfies $\Xi'$ and such that $\{X : (\mathtt{P}, \mathtt{L}, \mathtt{M})\} \supseteq m(\Theta')$, $m(\Pi') = \mathtt{P}'$, $m(\Lambda') = \emptyset$, $\mathtt{M}' = m(\Omega')$. By definition $\Theta' = \{X : (\pi_X, \lambda_X, \mu_X)\}$, we get $m(\pi_X) = \mathtt{P}$, $m(\lambda_X) = \emptyset$ and $m(\mu_X) = \mathtt{M}$. Being $\mathsf{core}(E)[X]$ a subterm of $E[X]$, by Lemma 5.2.20.(1) we get $X : (\mathtt{P}, \mathtt{L}, \mathtt{M}) \vdash \mathsf{core}(E)[X] : (\mathtt{P}'', \mathtt{L}', \mathtt{M}'')$ for some $\mathtt{P}''$, $\mathtt{L}'$, $\mathtt{M}''$. Theorem 5.3.3 implies that $\vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \lambda_X, \mu_X)\}; (\Pi, \Lambda, \Omega); \Xi$, and by construction $\Xi \subseteq \Xi'$, so $m$ satisfies $\Xi$ too. If $\mathsf{core}(E) = E$, then $\Lambda = \Lambda'$, which implies $m(\Lambda) = \emptyset$. Otherwise, neither $\pi_X$ nor $\lambda_X$ nor $\mu_X$ occurs in $\Psi$.

$\square$

**Example 5.3.6.** *Using Lemma 5.3.5, the constraints making OK the type associated with a generic variable $Y$ for the contexts in Example 5.2.18 are:*

$$\begin{array}{ll}
(A) & (\pi_Y, \lambda_Y, \mu_Y) \bowtie (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\}) \quad \emptyset = \emptyset \\
& ((\pi_Y, \lambda_Y, \mu_Y) \sqcup (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty) \\
(B) & \lambda_Y = \emptyset \\
(C) & (\pi_Y, \lambda_Y, \mu_Y) \bowtie (\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\}) \quad \emptyset = \emptyset \\
& ((\pi_Y, \lambda_Y, \mu_Y) \sqcup (\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty) \\
(D) & (\pi_Y, \lambda_Y, \mu_Y) \bowtie (\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty) \quad \lambda_Y \ominus \{\mathtt{t}_b : 1\} = \emptyset
\end{array}$$

*Note that $\Lambda$ is $\emptyset$ in (A) and (C).*

Once we have characterized the $\Delta$-$(\mathtt{P},\mathtt{L},\mathtt{M})$-safe rules, and also the $OK$ relation, we can infer the applicability of a rewrite rule by checking if the type mapping respects the constraints derived for these rules, as done for $\mathtt{P}/\mathtt{R}$ type discipline in Theorem 4.3.10.

**Theorem 5.3.7 (Applicability of rewrite rules).** *Let*

$$\vdash P_2 : \Theta_2; (\Pi_2, \Lambda_2, \Omega_2); \Xi_2$$

$$\vdash \mathsf{core}(E)[X] : \{X : (\pi_X, \lambda_X, \mu_X)\}; (\Pi_E, \Lambda_E, \Omega_E); \Xi_E$$

*and $P_1\sigma \not\equiv \epsilon$. Then the rule $P_1 \mapsto P_2$ can be applied to the term $E[P_1\sigma]$ such that $\vdash E[P_1\sigma] : (\mathtt{P}_0, \emptyset, \mathtt{M}_0)$ (for some $\mathtt{P}_0$, $\mathtt{M}_0$) if and only if the type mapping $m$ defined by*

- $m(\eta_x) = \mathtt{t}$ *if* $\sigma(x) : \mathtt{t} \in \Psi$,
- $m(\pi_\rho) = \mathtt{P}'$ *if* $\vdash \sigma(\rho) : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$,
- $m(\lambda_\rho) = \mathtt{L}'$ *if* $\vdash \sigma(\rho) : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$,
- $m(\mu_\rho) = \mathtt{M}'$ *if* $\vdash \sigma(\rho) : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$,

*satisfies the following sets of constraints:*

$$\Xi_2 \cup \Xi_E \cup \{(\pi_X = \Pi_2), (\lambda_X = \Lambda_2), (\mu_X = \Omega_2)\} \cup \{\Lambda_E = \emptyset \textit{ if } \mathsf{core}(E) = E\}$$

*Proof.* We define the basis $\Delta$ as follows:

- $x : (\{\mathtt{t}\}, \mathtt{D}_\mathtt{t}, \mathtt{U}_\mathtt{t}) \in \Delta$ if $\sigma(x) : \mathtt{t} \in \Psi$, and
- $\rho : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \in \Delta$ if $\vdash \sigma(\rho) : (\mathtt{P}', \mathtt{L}', \mathtt{M}')$.

In this way we get that $\sigma \in \Sigma_\Delta$ and the type mapping $m$ is such that:

- $m(\eta_x) = \mathtt{t}$ if and only if $x : (\{\mathtt{t}\}, \mathtt{D}_\mathtt{t}, \mathtt{U}_\mathtt{t}) \in \Delta$
- $m(\pi_\rho) = \mathtt{P}'$ if and only if $\rho : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \in \Delta$
- $m(\lambda_\rho) = \mathtt{L}'$ if and only if $\rho : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \in \Delta$
- $m(\mu_\rho) = \mathtt{M}'$ if and only if $\rho : (\mathtt{P}', \mathtt{L}', \mathtt{M}') \in \Delta$.

($\Leftarrow$) If the mapping $m$ satisfies the set of constraints $\Xi_2 \cup \Xi_E \cup \{(\pi_X = \Pi_2), (\lambda_X = \Lambda_2), (\mu_X = \Omega_2)\} \cup \{\Lambda_E = \emptyset \text{ if } \mathsf{core}(E) = E\}$, then $m(\pi_X) = m(\Pi_2) = \mathtt{P}, m(\lambda_X) = m(\Lambda_2) = \mathtt{L}, m(\mu_X) = m(\Omega_2) = \mathtt{M}$ for some $\mathtt{P}$, $\mathtt{L}$, $\mathtt{M}$, and by Lemma 5.3.5 the context $E$ is $OK$ for $(\mathtt{P}, \mathtt{L}, \mathtt{M})$ and by Lemma 5.3.4 the rule $P_1 \mapsto P_2$ is $\Delta$-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe; we get $E[P_1\sigma] \Longrightarrow E[P_2\sigma]$.

($\Rightarrow$) If $E[P_1\sigma] \Longrightarrow E[P_2\sigma]$ , then the rule $P_1 \mapsto P_2$ is $\Delta$-$(\mathtt{P}, \mathtt{L}, \mathtt{M})$-safe for some $\mathtt{P}$, $\mathtt{L}$, $\mathtt{M}$, and the context $E$ is $OK$ for $(\mathtt{P}, \mathtt{L}, \mathtt{M})$. By Lemmas 5.3.5 and 5.3.4, $m(\pi_X) = m(\Pi_2) = \mathtt{P}, m(\lambda_X) = m(\Lambda_2) = \mathtt{L}, m(\mu_X) = m(\Omega_2) = \mathtt{M}$, and the mapping $m$ satisfies the set of constraints $\Xi_2 \cup \Xi_E \cup \{(\pi_X = \Pi_2), (\lambda_X = \Lambda_2), (\mu_X = \Omega_2)\} \cup \{\Lambda_E = \emptyset \text{ if } \mathsf{core}(E) = E\}$.

$\square$

**Example 5.3.8.** *We use the Theorem 5.3.7 on the terms of Example 5.2.18. Each type mapping derived from the instantiation and from the constraints* $\{(\pi_Y = \Pi_2), (\lambda_Y = \Lambda_2), (\mu_Y = \Omega_2)\}$, *reported in Fig. 5.6, satisfies its own set of constraints for the right-hand of the rules and $OK$ relations for the evaluation contexts, reported in Examples 5.3.1 and 5.3.6, respectively.*

$(A)$ *rule constraints:* $\emptyset$
*context constraints:* $(\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty) \bowtie (\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$
$(\{\mathtt{t}_a : 3, \mathtt{t}_b, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b, \mathtt{t}_c : 0\}) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty) \quad \emptyset = \emptyset$

$(B)$ *rule constraints:* $(\{\mathtt{t}_a : 3, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\}) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$
$(\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty) \bowtie (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$
*context constraints:* $\emptyset = \emptyset$

$(C)$ *rule constraints:* $\emptyset$
*context constraints:* $(\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty) \bowtie (\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\})$
$(\{\mathtt{t}_a : 2, \mathtt{t}_b, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b, \mathtt{t}_c : 0\}) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty) \quad \emptyset = \emptyset$

$(D)$ *rule constraints:* $(\{\mathtt{t}_a : 2, \mathtt{t}_c : 1\}, \emptyset, \{\mathtt{t}_b : 1, \mathtt{t}_c : 0\}) \sqsubseteq (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$
$(\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty) \bowtie (\{\mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty)$
*context constraints:* $(\{\mathtt{t}_b, \mathtt{t}_d : 1\}, \emptyset, \Gamma_\infty) \bowtie (\{\mathtt{t}_b : 1\}, \emptyset, \Gamma_\infty) \quad \emptyset = \emptyset$

| | $\pi_X$ | $\lambda_X$ | $\mu_X$ | $\pi_Y$ | $\lambda_Y$ | $\mu_Y$ |
|---|---|---|---|---|---|---|
| (A) | — | — | — | $\mathtt{t}_b : 1$ | $\emptyset$ | $\Gamma_\infty$ |
| (B) | $\mathtt{t}_a : 3, \mathtt{t}_c : 1$ | $\emptyset$ | $\mathtt{t}_b : 1, \mathtt{t}_c : 0$ | $\mathtt{t}_b, \mathtt{t}_d : 1$ | $\emptyset$ | $\Gamma_\infty$ |
| (C) | — | — | — | $\mathtt{t}_b : 1$ | $\emptyset$ | $\Gamma_\infty$ |
| (D) | $\mathtt{t}_a : 2, \mathtt{t}_c : 1$ | $\emptyset$ | $\mathtt{t}_b : 1, \mathtt{t}_c : 0$ | $\mathtt{t}_b, \mathtt{t}_d : 1$ | $\emptyset$ | $\Gamma_\infty$ |

Figure 5.6: Type mappings of Example 5.3.8

| Name | Structure | Informations |
|---|---|---|
| $A$ | $\alpha_2\beta_2$ | The most common |
| $A_2$ | $\alpha_2\delta_2$ | It has a normal range of 1.5-3.5% |
| $F$ | $\alpha_2\gamma_2$ | The one presents in the fetus |
| $H$ | $\beta_4$ | It may be present in variants of $\alpha$ thalassemia |
| $Barts$ | $\delta_4$ | It may be present in variants of $\alpha$ thalassemia |

Table 5.1: Some hemoglobin variants in humans.

## 5.4 Examples

The type discipline presented in Section 5.2 can be used both to describe the structure of an element and to limit the presence of some element. We present an example for each use: for the structure description we present the hemoglobin variants, and for the limitation we present the regulation between cell death and division, an example of homeostatic balance in living organisms. The aim of these examples is not to describe a complete biological case study, but to give an idea of the possible uses of the type discipline.

### 5.4.1 Hemoglobin Variants

Hemoglobin (abbreviated Hb) is the oxygen-transport metalloprotein in the red blood cells of vertebrates, and in the tissues of some invertebrates. It consists mostly of proteins (the globin chains), which usually differ between species, and even within a species, although one sequence is usually a most common one in each species. In humans, the hemoglobin molecule is an assembly of four globular protein subunits: the most common, with a normal amount over 95%, is the hemoglobin A, consisting of two $\alpha$ and two $\beta$ subunits, but there are some hemoglobin variants, as reported in Table 5.1. Many of these cause no disease, but some of these cause a group of hereditary diseases, known as hemoglobinopathies: the most known are sickle-cell disease, in which red blood cells assume an abnormal and rigid shape, and thalassemias, that usually result in underproduction of normal globin proteins.

We want to model the different kinds of hemoglobin: we model an hemoglobin protein as a looping sequence having the element $h$ on the surface and containing, depending on the kind of hemoglobin, four subunits, chosen between $\alpha$, $\beta$, $\gamma$ and $\delta$, and one of the elements $A$, $A_2$, $F$, $H$ or $B$, representing the

| element | basic type | minimum & maximum |
|---|---|---|
| $\alpha$ | $\mathtt{t}_\alpha$ | — |
| $\beta$ | $\mathtt{t}_\beta$ | — |
| $\gamma$ | $\mathtt{t}_\gamma$ | — |
| $\delta$ | $\mathtt{t}_\delta$ | — |
| $A$ | $\mathtt{t}_A$ | $\mathtt{t}_\gamma, \mathtt{t}_\delta, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\alpha, \mathtt{t}_\beta : 2$ |
| $A_2$ | $\mathtt{t}_{A_2}$ | $\mathtt{t}_\beta, \mathtt{t}_\gamma, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\alpha, \mathtt{t}_\delta : 2$ |
| $F$ | $\mathtt{t}_F$ | $\mathtt{t}_\beta, \mathtt{t}_\delta, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\alpha, \mathtt{t}_\gamma : 2$ |
| $H$ | $\mathtt{t}_H$ | $\mathtt{t}_\alpha, \mathtt{t}_\gamma, \mathtt{t}_\delta, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\beta : 4$ |
| $Barts$ | $\mathtt{t}_B$ | $\mathtt{t}_\alpha, \mathtt{t}_\beta, \mathtt{t}_\gamma, \mathtt{t}_A, \mathtt{t}_{A_2}, \mathtt{t}_F, \mathtt{t}_H, \mathtt{t}_B : 0,\ \mathtt{t}_\delta : 4$ |

Table 5.2: Basic types for hemoglobin variants and subunits.

different kinds of hemoglobin in Table 5.1. For example, the term modeling the hemoglobin A is $(h)^{\circlearrowleft} \rfloor (A \mid \alpha{\cdot}\alpha{\cdot}\beta{\cdot}\beta)$, and the one for hemoglobin H is $(h)^{\circlearrowleft} \rfloor (H \mid \beta{\cdot}\beta{\cdot}\beta{\cdot}\beta)$. According to the structure of each hemoglobin variant, we create the basic types shown in Table 5.2.

Using the typed extension of CLS and these basic type, no rule can change the structure of the different kinds of hemoglobin without removing or modifying its structural element $A$, $A_2$, $F$, $H$ or $B$.

### 5.4.2 Cell Death and Division

In multicellular organisms, the life of cells is ruled by two oppose processes: a cell division process, resulting in cell multiplication (mitosis for eukaryotic cells and binary fission for prokaryotic cells), and a process of programmed cell death, called apoptosis. In an adult organism, the rate of these processes must be balanced: an excess of cell death leads to cell loss, and an excess of cell division leads to tumors. For this reason, the number of cells is kept relatively constant through cell death and division. Using our type discipline, we can model this behavior in a simple way. We model a cell as a looping sequence having the element $c$ on the surface and containing the other elements of interest for the study, in this case only the element $a$: $(c)^{\circlearrowleft} \rfloor a$. We assume the organism can support from 2 to 8 cells, and the element $a$ does not have any request. Assuming the set of basic types $\Gamma = \{\mathtt{t}_a, \mathtt{t}_c\}$, a basic type $\mathtt{t}_a$ for the element $a$ and a basic type $\mathtt{t}_c$ for the element $c$, the functions `mn` and `mx` that model this behavior are:

| mn | $\mathtt{t}_a$ | $\mathtt{t}_c$ |
|---|---|---|
| $\mathtt{t}_a$ | 0 | 0 |
| $\mathtt{t}_c$ | 0 | 1 |

| mx | $\mathtt{t}_a$ | $\mathtt{t}_c$ |
|---|---|---|
| $\mathtt{t}_a$ | $\Gamma_\infty$ | $\Gamma_\infty$ |
| $\mathtt{t}_c$ | $\Gamma_\infty$ | 7 |

In this model, the rules for cell death and cell division are

$$\begin{array}{ll}(de) & (\widetilde{x}\cdot c)^{\circlearrowleft} \rfloor X \mapsto \epsilon \\ (di) & (\widetilde{x}\cdot c)^{\circlearrowleft} \rfloor X \mapsto (\widetilde{x}\cdot c)^{\circlearrowleft} \rfloor X \mid (\widetilde{x}\cdot c)^{\circlearrowleft} \rfloor X\end{array}$$

respectively. For the right-side patterns of these rules we can infer the following types:

$$(rp1)\ \vdash \epsilon : \emptyset; (\emptyset, \emptyset, \Gamma_\infty); \emptyset$$

$$(rp2)\ \vdash (\widetilde{x}\cdot c)^{\circlearrowleft} \rfloor X \mid (\widetilde{x}\cdot c)^{\circlearrowleft} \rfloor X : \Theta_{\widetilde{x},X}; (\Pi,\Lambda,\Omega) \sqcup (\Pi,\Lambda,\Omega);$$
$$\Xi \cup \{(\Pi,\Lambda,\Omega) \bowtie (\Pi,\Lambda,\Omega)\}$$

where

$$\begin{array}{rcl}\Theta_{\widetilde{x},X} & = & \{\widetilde{x} : (\pi_{\widetilde{x}}, \lambda_{\widetilde{x}}, \mu_{\widetilde{x}}), X : (\pi_X, \lambda_X, \mu_X)\} \\ \Pi & = & \{\mathtt{t}_c : 1\} \uplus \pi_{\widetilde{x}} \\ \Lambda & = & ((\{\mathtt{t}_c : 1\} \ominus \pi_{\widetilde{x}}) \cup (\lambda_{\widetilde{x}} \ominus \{\mathtt{t}_c : 1\})) \ominus \pi_X \\ \Omega & = & ((\{\mathtt{t}_c : 7\} \ominus \pi_{\widetilde{x}}) \cap (\mu_{\widetilde{x}} \ominus \{\mathtt{t}_c : 1\})) \ominus \pi_X \\ \Xi & = & \{(\{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 7\}) \bowtie (\pi_{\widetilde{x}}, \lambda_{\widetilde{x}}, \mu_{\widetilde{x}}) \quad \cup \\ & & (\pi_X, \lambda_X, \mu_X)\} \sqsubseteq ((\{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 7\}) \sqcup (\pi_{\widetilde{x}}, \lambda_{\widetilde{x}}, \mu_{\widetilde{x}})).\end{array}$$

We want to study a system composed by two cells

$$(c)^{\circlearrowleft} \rfloor a \mid (c)^{\circlearrowleft} \rfloor a$$

and we try to apply rules $(de)$ and $(di)$ on it. Because both rules have the same left side pattern, they use the same instantiation, $\sigma(\widetilde{x}) = \epsilon$ and $\sigma(X) = a$, and also the same evaluation context $(c)^{\circlearrowleft} \rfloor a \mid \square$, for which we derive, using the rules in Figure 5.5, the inferred type:

$$\vdash (c)^{\circlearrowleft} \rfloor a \mid Y : \Theta_Y; (\Pi_E, \Lambda_E, \Omega_E); \Xi_E$$

where

$$\begin{array}{rcl}\Theta_Y & = & \{Y : (\pi_Y, \lambda_Y, \mu_Y)\} \\ \Pi_E & = & \{\mathtt{t}_c : 1\} \uplus \pi_Y \\ \Lambda_E & = & (\{\mathtt{t}_c : 1\} \ominus \pi_Y) \cup (\lambda_Y \ominus \{\mathtt{t}_c : 1\}) \\ \Omega_E & = & (\{\mathtt{t}_c : 7\} \ominus \pi_Y) \cap (\mu_Y \ominus \{\mathtt{t}_c : 1\}) \\ \Xi_E & = & \{(\{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 1\}, \{\mathtt{t}_c : 7\}) \bowtie (\pi_Y, \lambda_Y, \mu_Y).\end{array}$$

We use Theorem 5.3.7 for checking the applicability of the rules. First of all, from the instantiation $\sigma$ we get the type mapping

| $\pi_{\widetilde{x}}$ | $\lambda_{\widetilde{x}}$ | $\mu_{\widetilde{x}}$ | $\pi_X$ | $\lambda_X$ | $\mu_X$ |
|---|---|---|---|---|---|
| $\emptyset$ | $\emptyset$ | $\Gamma_\infty$ | $\mathtt{t}_a : 1$ | $\emptyset$ | $\Gamma_\infty$ |

and we use it to instantiate the type and the constraints of the patterns $(rp1)$ and $(rp2)$, obtaining that the constraints of both patterns are satisfied, and

$$\Pi = \{\mathtt{t}_c : 1\} \qquad \Lambda = \{\mathtt{t}_c : 1\} \qquad \Omega = \{\mathtt{t}_c : 7\}.$$

Now we check the other constraints in Theorem 5.3.7. For the rule $(di)$ we have

$$\pi_Y = \{\mathtt{t}_c : 2\} \qquad \lambda_Y = \emptyset \qquad \mu_Y = \{\mathtt{t}_c : 6\}.$$

The constraints of the context, $\Xi_E$, and the constraint $\Lambda_E = \emptyset$ are satisfied, then we can apply the rule.
Finally, for the rule $(de)$ we have

$$\pi_Y = \emptyset \qquad \lambda_Y = \emptyset \qquad \mu_Y = \Gamma_\infty.$$

The constraints of the context, $\Xi_E$, are satisfied, but the constraint $\{\mathtt{t}_c : 1\} = \Lambda_E = \emptyset$ is not satisfied, then we cannot apply the rule: in fact, this rule kills a cell, an invalid behavior in an organism composed by only 2 cells, the minimum required number. In a dual way, cell division is not possible in an organism composed by 8 cells.

## 5.5 Conclusions

In this Chapter we introduce a type discipline for the Calculus of the Looping Sequences which allows to describe and to limit the structure of systems and sub-systems: this behavior cannot be easily reproduced only by means of reduction rules.

In Chapter 4 some rules are classified as $\Delta$-safe (see Definition 4.2.7), i.e. rules having the same type for the left-side and the right-side patterns: these rules do not change the type of terms to which they are applied. In the present type discipline, for typing we count the number of elements of a term. Since rules usually change something in the term, adding or removing elements, in a rule the number of elements in the right-hand-side is different from the number of elements in the left-hand-side, and therefore their types are also different. According to this idea, we do not include $\Delta$-safe rules in the present semantics, because using the present type discipline very few reduction rules are $\Delta$-safe.

Finally, in biological models we usually do not know the precise numbers of elements in the system, but their concentration, as percentage of the single elements in the whole system: for example, the corpuscles in blood are usually given as a percentage or as an absolute number per litre. Our type discipline cannot manage these cases, because it checks the exact numbers of elements in every compartment. As a possible future development, we plan to modify our type discipline to work on these cases, checking, in every compartment, not the exact numbers, but the ratio of elements with respect to the other elements.

# Chapter 6

# Stochastic Semantics for the Calculus of Looping Sequences

## 6.1 Introduction

In the stochastic version of CLS (SCLS for short), proposed in [11] and summarized in Section 2.3, rates are associated with rewrite rules in order to model the speed of the described activities. Therefore, transitions derived in SCLS are driven by a rate that models the parameter of an exponential distribution and characterizes the stochastic behavior of the transition. The choice of the next rule to be applied and of the time of its application is based on the classical Gillespie's algorithm [44].

Defining a stochastic semantics for CLS requires a correct enumeration of all the possible and distinct ways to apply each rewrite rule within a term. A single pattern may have several, though isomorphic, matches within a CLS term. Here we present the solution proposed in [22], where we simplify the counting mechanism used in [11] by imposing some restrictions on the patterns modeling the rewrite rules. Each rewrite rule states explicitly the types of the elements whose occurrences may speed-up or slow-down a reaction. The occurrences of the elements of these types are then processed by a rate function which is used to compute the actual rate of a transition. We show how we can define patterns in our stochastic framework to model some common biological activities, and, in particular, we underline the possibility to combine the modeling of positive and negative catalyzers within a single rule by reproducing a general case of osmosis.

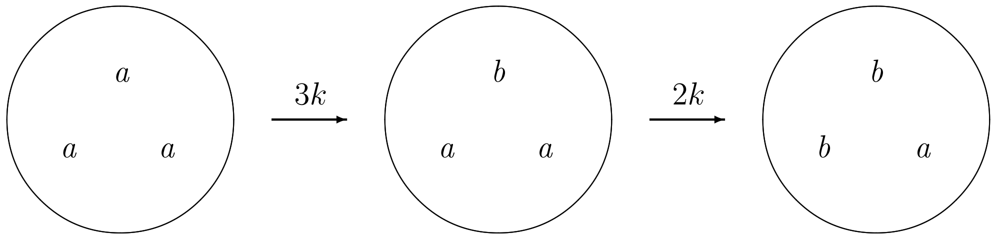

Figure 6.1: Application of $a \to b$ with kinetic constant $k$.

While standard quantitative bio-inspired formalisms give stochastic semantics based on constant rates, we equip the rewrite rules of our calculus with a rate function. This makes possible the definition of a stochastic semantics that is more general than the classical one based on collision analysis (which is practical for very low level analyzes, such as chemical interactions). In particular, we can define rules, whose evolutions follow different probability distributions. This is useful for higher level simulations, for example cellular or tissue interactions, or in other cases such as in the presence of enzymes (molecules that speed up the reaction) or inhibitors (molecules that slow down the reaction) where the reaction rate equation becomes complicated, and must be calculated using non-linear equations. We show how a particular interpretation of the rate function could be used to recover Gillespie's method.

More specifically, we add to the reduction rules of CLS the information on the relevant objects and a function which computes the rate of the reduction starting from the numbers of relevant objects which can occur in the instantiations of the variables.

As a simple example, consider a molecule $a$ becoming a molecule $b$ with a kinetic constant $k$. The rate of this transformation is proportional to the concentration of $a$'s. Figure 6.1 shows how $a$ molecules contained inside a membrane evolve with a rate calculated by means of $k$ and the concentration of $a$'s. This transformation is modeled in our calculus by the rewrite rule:

$$a \mid X \xrightarrow[\phi]{\langle\langle \bar{t} \rangle\rangle} b \mid X$$

where $t$ is the type of molecule $a$, the overline means that $a$ occurs in a parallel composition, and the function $\phi$ is $\lambda n.(n+1) \times k$. When reducing a term by means of this rule, we compute the rate by applying the function

$\phi$ to the number of parallel occurrences of molecules of type $t$, in the term matching the variable $X$ (representing the environment in which the rule will be applied).

As a complete modeling application, we illustrate the expressiveness of our formalism by adapting to our framework the model for lactose operon in *Escherichia Coli* proposed in [11].

## 6.2 Typed Stochastic CLS

In this section we show how an abstraction on the elements, that induces an abstraction on the terms, may be used to enhance the expressiveness of CLS. In particular, we use this abstraction to focus on quantitative aspects of CLS, by showing how to model the speeds of the biological activities.

As in Sections 4.2 and 5.2, we classify elements in $\mathcal{A}$ with basic types, assuming a fixed typing $\Psi$.

A term $T$ is abstracted by saying that the *type of* $T$ is the multiset of the types of the elements in its outermost parallel composition. We distinguish between occurrences of elements in parallel with other terms, and occurrences of elements within a sequence by having two names for each type. In particular the type of $T$ will contain a $\bar{t}$ for each occurrence of $a : t \in \Psi$ in parallel with some other term, and $\widetilde{t}$ for each occurrence of $a : t \in \Psi$ which is either in a sequence or in the looping sequence of a compartment (in both cases we only consider the outermost parallel composition). We use $\overset{*}{t}$ to range over both $\bar{t}$ and $\widetilde{t}$, and $\tau$ to range over types of terms. By $\overset{*}{t} \in_n \tau$ we denote that $\overset{*}{t}$ occurs $n$ times in $\tau$, and $\uplus$ is the sum on multisets[1]. In the following, when we say *type* we refer to either $t$'s, or $\overset{*}{t}$'s, or $\tau$'s.

The following definition formalizes the mappings *ptype* on terms and *stype* on sequences.

**Definition 6.2.1 (Mappings** *ptype* **and** *stype***).** *The mappings ptype and stype are defined by induction on terms and sequences as follows:*

$$stype(S) = \begin{cases} stype(S_1) \uplus stype(S_2) & \text{if } S \equiv S_1 \cdot S_2 \\ \{\widetilde{t}\} & \text{if } S \equiv a \text{ and } a : t \in \Psi \end{cases}$$

[1]For a formal definition of the sum operation on multisets, refer to Figure 5.1.

$$ptype(T) = \begin{cases} stype(S) & \text{if } T \equiv (S)^{\circlearrowleft} \rfloor T' \\ ptype(T_1) \uplus ptype(T_2) & \text{if } T \equiv T_1 \mid T_2 \\ stype(S_1 \cdot S_2) & \text{if } T \equiv S_1 \cdot S_2 \\ \{\bar{t}\} & \text{if } T \equiv a \text{ and } a : t \in \Psi \end{cases}$$

For example, if $a : t_a, b : t_b, c : t_c \in \Psi$, we have

$$\begin{aligned} &ptype(a \mid a \mid c) = \{\overline{t_a}, \overline{t_a}, \overline{t_c}\} \\ &ptype(b \cdot c \cdot c) = \{\widetilde{t_b}, \widetilde{t_c}, \widetilde{t_c}\} \\ &ptype(a \mid a \mid c \mid (b \cdot c \cdot c)^{\circlearrowleft} \rfloor a) = \{\overline{t_a}, \overline{t_a}, \overline{t_c}, \widetilde{t_b}, \widetilde{t_c}, \widetilde{t_c}\} \\ &ptype((b \cdot c \cdot c)^{\circlearrowleft} \rfloor (a \mid a \mid a \mid c)) = \{\widetilde{t_b}, \widetilde{t_c}, \widetilde{t_c}\} \end{aligned}$$

while if $a : t, b : t, c : t' \in \Psi$, we get

$$\begin{aligned} &ptype(a \mid a \mid c) = \{\bar{t}, \bar{t}, \overline{t'}\} \\ &ptype(b \cdot c \cdot c) = \{\widetilde{t}, \widetilde{t'}, \widetilde{t'}\} \\ &ptype(a \mid a \mid c \mid (b \cdot c \cdot c)^{\circlearrowleft} \rfloor a) = \{\bar{t}, \bar{t}, \overline{t'}, \widetilde{t}, \widetilde{t'}, \widetilde{t'}\} \\ &ptype((b \cdot c \cdot c)^{\circlearrowleft} \rfloor (a \mid a \mid a \mid c)) = \{\widetilde{t}, \widetilde{t'}, \widetilde{t'}\}. \end{aligned}$$

Term transitions are labeled with a *rate* $r$, a real number, $T \xrightarrow{r} T'$, modeling the speed of the transition. The number $r$ depends on the types and multiplicity of the elements interacting.

To compute the rate of transitions we associate to each rule, $P \mapsto P'$ the information which is relevant to the application of the rule. This is expressed by giving:

- for each variable $\chi$ in the pattern $P$, the types of the elements that influence the speed of the application of the rule,
- a weighting function that combines the multiplicity of types on single variables, producing the final rate.

We provide this information as follows. Given a pattern $P$, let $V(P) = \langle \chi_1, \ldots, \chi_m \rangle$ be the list of (sequence, term, and element) variables of $P$ in left-to-right order of occurrence.

- To each $\chi_i$ we associate a list $\Pi_i = \langle \overset{*}{t}{}_1^{(i)}, \ldots, \overset{*}{t}{}_{q_i}^{(i)} \rangle$ of types,
- Moreover, let $\phi : \mathbf{N}^q \to \mathbf{R}$ be a function from a list of $q = \sum_{1 \le i \le m} q_i$ integers to a real.

The rewrite rules of our *typed Stochastic* CLS (TSCLS for short) are of the shape:

$$P \xrightarrow[\phi]{\overrightarrow{\Pi}} P'$$

where $\overrightarrow{\Pi} = \langle \Pi_1, \ldots, \Pi_m \rangle$.

For example, as discussed in the following subsection, the transformation of the element $a$ into the element $b$ inhibited by the presence of the element $c$ can be described by the rule

$$a \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_a, \bar{t}_c \rangle\rangle} b \mid X \tag{6.1}$$

where $\phi = \lambda n_1 n_2 . \frac{(n_1+1)\times k}{\text{if } n_2=0 \text{ then } 1 \text{ else } n_2 \times k'}$, and $k, k'$ are the kinetic constants of the state change of $a$ into $b$ and the deceleration due to the presence of one inhibitor $c$, respectively.

**Remark 6.2.2.** *We consider local interactions, that is interactions between elements in the same compartment. As for the CLS semantics, given a term we match the left-hand-side pattern of a rule against the subterm contained in the hole of a context. However, in our case, when applying a rule we have to take into account a whole compartment, since our function depends only on the content of the subterm matching the pattern. For instance, for the previous example, matching the term* $(S)^{\circlearrowleft} \rfloor (a \mid a \mid c \mid c)$, *we do not want a context* $E = (S)^{\circlearrowleft} \rfloor (\square \mid c)$ *that would cause a miscounting of the c's present in the compartment.*

Given the above remark, we restrict Definition 2.2.6, to allow only a hole filling a compartment or the whole term.

**Definition 6.2.3 (Stochastic Contexts).** Stochastic Contexts $\mathtt{C}$ *are defined as:*

$$\mathtt{C} ::= \square \quad \Big| \quad \mathtt{T} \mid (\mathtt{S})^{\circlearrowleft} \rfloor \mathtt{C}$$

*where* $T \in \mathcal{T}$ *and* $S \in \mathcal{S}$. *We denote by* $\mathcal{SC}$ *the infinite set of stochastic contexts.*

We can now define the typed semantics.

**Definition 6.2.4 (Typed Stochastic Semantics).** *Given a finite set $\mathcal{R}$ of rewrite rules, the* semantics *of* TSCLS *is the least relation closed with respect to $\equiv$ and satisfying the following rule:*

$$\frac{\begin{array}{c} P_1 \xrightarrow[\phi]{\langle \Pi_1,\ldots,\Pi_m\rangle} P_2 \in \mathcal{R} \quad \Pi_i = \langle \overset{*(i)}{t_1}, \ldots, \overset{*(i)}{t_{q_i}} \rangle \\ \sigma \in \Sigma \quad P_1\sigma \not\equiv \epsilon \quad \mathtt{C} \in \mathcal{SC} \quad \mathtt{V(P_1)} = \langle \chi_1, \ldots, \chi_m \rangle \\ ptype(\sigma(\chi_i)) = \tau_i \qquad \overset{*(i)}{t_j} \in_{n_j^{(i)}} \tau_i \quad (1 \le j \le q_i) \quad (1 \le i \le m) \\ r = \phi\, n_1^{(1)} \ldots n_{q_1}^{(1)} \cdots n_1^{(m)} \ldots n_{q_m}^{(m)} \end{array}}{\mathtt{C}[\mathtt{P}_1\sigma] \xrightarrow{\mathtt{r}} \mathtt{C}[\mathtt{P}_2\sigma]}$$

**Example 6.2.5.** *Applying rule (6.1) with the empty context to the term $a \mid a \mid c$ we have:*

$$a \mid a \mid c \xrightarrow{\frac{2\times k}{1\times k'}} a \mid b \mid c \xrightarrow{\frac{1\times k}{1\times k'}} b \mid b \mid c$$

*and to the term $a \mid a \mid c \mid (b \cdot c \cdot c)^{\circlearrowleft} \rfloor a$ we have:*

$$a \mid a \mid c \mid (b \cdot c \cdot c)^{\circlearrowleft} \rfloor a \xrightarrow{\frac{2\times k}{1\times k'}} a \mid b \mid c \mid (b \cdot c \cdot c)^{\circlearrowleft} \rfloor a \xrightarrow{\frac{1\times k}{1\times k'}} b \mid b \mid c \mid (b \cdot c \cdot c)^{\circlearrowleft} \rfloor a$$

*Similarly, applying (6.1) to the term $(b \cdot c \cdot c)^{\circlearrowleft} \rfloor (a \mid a \mid a \mid c)$ with the context $\epsilon \mid (b \cdot c \cdot c)^{\circlearrowleft} \rfloor \square$ we get:*

$$\begin{array}{rcl} (b \cdot c \cdot c)^{\circlearrowleft} \rfloor (a \mid a \mid a \mid c) & \xrightarrow{\frac{3\times k}{1\times k'}} & (b \cdot c \cdot c)^{\circlearrowleft} \rfloor (a \mid a \mid b \mid c) \\ & \xrightarrow{\frac{2\times k}{1\times k'}} & (b \cdot c \cdot c)^{\circlearrowleft} \rfloor (a \mid b \mid b \mid c) \\ & \xrightarrow{\frac{1\times k}{1\times k'}} & (b \cdot c \cdot c)^{\circlearrowleft} \rfloor (b \mid b \mid b \mid c) \end{array}$$

*Note that we cannot simply use Definition 2.2.6 for the contexts in the stochastic framework, since we would not count correctly the numbers of elements which influence the speed of transformations (see Remark 6.2.2). For example, rule (6.1) applied to the term $a \mid a \mid c$ with the context $\square \mid a \mid c$ would produce the wrong transition:*

$$a \mid a \mid c \xrightarrow{k} a \mid b \mid c.$$

Given the Continuous Time Markov Chain (CTMC) obtained from the transition system resulting from our typed stochastic semantics, we can follow a standard simulation procedure. Roughly speaking, the algorithm starts from the initial term (representing a state of the CTMC) and performs a sequence of steps by moving from state to state. At each step a global clock variable (initially set to zero) is incremented by a random quantity which is exponentially distributed with the exit rate of the current state as parameter, and the next state is randomly chosen with a probability proportional to the rates of the exit transitions.

The *race condition* described above implements the fact that when different reactions are competing with different rates, the ones which are not chosen should restart the competition at the following step.

From the transition rates of Definition 6.2.4, we can define an exponential probability distribution of the moment in which the next reaction will take place as follows. Given a term $T$, a global time $\mathfrak{T}$ and all the transitions $R_1, \ldots, R_M$ that can be applied to $T$, with rates, respectively, $r_1, \ldots, r_M$ such that $r = \sum_{i=1}^{M} r_i$, the standard simulation procedure consists of the following two steps:

- The time $\mathfrak{T} + \delta$ at which the next stochastic reduction will occur is randomly chosen with $\delta$ exponentially distributed with parameter $r$;

- The reduction $R_i$ that will occur at time $\mathfrak{T} + \delta$ is randomly chosen with probability $r_i/r$.

Thus, the overall complexity of a single simulation step, for a term with $n$ stochastic contexts on which we can apply a rule and with a set of $m$ rules, is given by the $n \times m$ matchings needed to compute the set of all possible transitions $R_1, \ldots, R_M$.

### 6.2.1 Modeling Guidelines

In the remaining of this section we will put at work the TSCLS calculus in order to model biomolecular events of interest.

(i) As discussed in Section 6.1, the application rate in the case of the *change of state of an elementary object* is proportional to the number of objects which are present. For this reason, if $t_a$ is the type of the object $a$ and $k$ is the kinetic constant of the state change of $a$ into $b$ we can describe this chemical reaction by the following rewrite rule:

$$a \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_a \rangle\rangle} b \mid X$$

where $\phi = \lambda n.(n+1) \times k$. Notice that only occurrences of $a$ in parallel can become $b$, so using this rule we get for example:

$$(m)^{\circlearrowleft} \rfloor (a \mid a \mid a \cdot a) \xrightarrow{2k} (m)^{\circlearrowleft} \rfloor (b \mid a \mid a \cdot a)$$

where $m$ is any membrane.

(ii) In the process of *complexation*, two elementary objects in the same compartment are combined to produce a new object. The application rate is then proportional to the product of the numbers of occurrences of the two objects. Assuming that $t_a$ and $t_b$ are the types of $a$ and $b$ we get:

$$a \mid b \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_a, \bar{t}_b \rangle\rangle} c \mid X$$

where $\phi = \lambda n_1 n_2.(n_1 + 1) \times (n_2 + 1) \times k$ and $k$ is the kinetic constant of the modeled chemical reaction.

Using the same conventions a similar and simpler rule describes *decomplexation*:

$$c \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_c \rangle\rangle} a \mid b \mid X$$

where $\phi = \lambda n.(n+1) \times k$.

(iii) Another phenomenon which can be easily rendered in our formalism is the *osmosis* regulating the quantity of water inside and outside a cell for a dilute solution of non-dissociating substances. In fact, in this case, according to [65], the total flow is $L_p \frac{S}{V} \Delta\psi_w$, where $L_p$ is the hydraulic conductivity constant, which depends on the semi-permeability properties of the membrane, $S$ is the surface of the cell, $V$ is the volume of the cell, and $\Delta\psi_w = \psi_{w(ext)} - \psi_{w(int)}$ is the difference between the water potentials outside and inside the cell. The water potential for non-dissociating substances is the sum of the solute potential $\psi_s = -\mathsf{R}\mathsf{T}c_s$ (where $\mathsf{R}$ is the gas constant, $\mathsf{T}$ is the absolute temperature and $c_s$ is the solute concentration) and the pressure potential $\psi_p$ (which depends on the elastic properties of the membrane and on the cell wall). We can therefore consider the rate of flow of water proportional (via a constant $k$) to $\frac{S}{V}(c_{s(ext)} - c_{s(int)})$, where the sign of this real gives the direction of the flow. The membrane crossing of the element $a$ according to the concentration of the elements $b$ inside and outside the cell is given by the pairs of rules:

$$(\widetilde{x})^{\circlearrowleft} \rfloor (X \mid a) \mid Y \xrightarrow[\phi]{\langle\langle\rangle,\langle\bar{t}_a,\bar{t}_b\rangle,\langle\bar{t}_a,\bar{t}_b\rangle\rangle} (\widetilde{x})^{\circlearrowleft} \rfloor X \mid a \mid Y$$
$$(\widetilde{x})^{\circlearrowleft} \rfloor X \mid a \mid Y \xrightarrow[\phi']{\langle\langle\rangle,\langle\bar{t}_a,\bar{t}_b\rangle,\langle\bar{t}_a,\bar{t}_b\rangle\rangle} (\widetilde{x})^{\circlearrowleft} \rfloor (X \mid a) \mid Y$$

where
$$\phi = \lambda n_1 n_2 n_3 n_4 . \tfrac{S}{V} \times \left(\tfrac{n_2}{(n_1+1)V_a + n_2 V_b} - \tfrac{n_4}{(n_3+1)V_a + n_4 V_b}\right) \times k$$
$$\phi' = \lambda n_1 n_2 n_3 n_4 . \tfrac{S}{V} \times \left(\tfrac{n_4}{(n_3+1)V_a + n_4 V_b} - \tfrac{n_2}{(n_1+1)V_a + n_2 V_b}\right) \times k$$

and $V_a$, $V_b$ are the volumes of the elements $a$ and $b$, respectively.

The *positive catalysis* of osmosis by the presence of elements $c$ on the membrane is rendered by:

$$(\widetilde{x})^{\circlearrowleft} \rfloor (X \mid a) \mid Y \xrightarrow[\phi]{\langle\langle\widetilde{t}_c\rangle,\langle\bar{t}_a,\bar{t}_b\rangle,\langle\bar{t}_a,\bar{t}_b\rangle\rangle} (\widetilde{x})^{\circlearrowleft} \rfloor X \mid a \mid Y$$
$$(\widetilde{x})^{\circlearrowleft} \rfloor X \mid a \mid Y \xrightarrow[\phi']{\langle\langle\widetilde{t}_c\rangle,\langle\bar{t}_a,\bar{t}_b\rangle,\langle\bar{t}_a,\bar{t}_b\rangle\rangle} (\widetilde{x})^{\circlearrowleft} \rfloor (X \mid a) \mid Y$$

where
$$\phi = \lambda n_1 n_2 n_3 n_4 n_5 . (n_1 \times k_c + 1) \times \tfrac{S}{V} \times \left(\tfrac{n_3}{(n_2+1)V_a + n_3 V_b} - \tfrac{n_5}{(n_4+1)V_a + n_5 V_b}\right) \times k$$
$$\phi' = \lambda n_1 n_2 n_3 n_4 n_5 . (n_1 \times k_c + 1) \times \tfrac{S}{V} \times \left(\tfrac{n_5}{(n_4+1)V_a + n_5 V_b} - \tfrac{n_3}{(n_2+1)V_a + n_3 V_b}\right) \times k$$

and $k_c$ is the acceleration due to the presence of one element $c$.

Similarly the *inhibition* of osmosis by the presence of elements $c$ on the membrane is rendered by:

$$(\widetilde{x})^{\circlearrowleft} \rfloor (X \mid a) \mid Y \xrightarrow[\phi]{\langle\langle\widetilde{t}_c\rangle,\langle\bar{t}_a,\bar{t}_b\rangle,\langle\bar{t}_a,\bar{t}_b\rangle\rangle} (\widetilde{x})^{\circlearrowleft} \rfloor X \mid a \mid Y$$
$$(\widetilde{x})^{\circlearrowleft} \rfloor X \mid a \mid Y \xrightarrow[\phi']{\langle\langle\widetilde{t}_c\rangle,\langle\bar{t}_a,\bar{t}_b\rangle,\langle\bar{t}_a,\bar{t}_b\rangle\rangle} (\widetilde{x})^{\circlearrowleft} \rfloor (X \mid a) \mid Y$$

where
$$\phi = \lambda n_1 n_2 n_3 n_4 n_5 . \tfrac{1}{\text{if } n_1 = 0 \text{ then } 1 \text{ else } n_1 \times k_c} \times \tfrac{S}{V} \times \left(\tfrac{n_3}{(n_2+1)V_a + n_3 V_b} - \tfrac{n_5}{(n_4+1)V_a + n_5 V_b}\right) \times k$$
$$\phi' = \lambda n_1 n_2 n_3 n_4 n_5 . \tfrac{1}{\text{if } n_1 = 0 \text{ then } 1 \text{ else } n_1 \times k_c} \times \tfrac{S}{V} \times \left(\tfrac{n_5}{(n_4+1)V_a + n_5 V_b} - \tfrac{n_3}{(n_2+1)V_a + n_3 V_b}\right) \times k$$

and $k_c$ is the deceleration due to the presence of one element $c$.

(iv) If the rule

$$P_1 \xrightarrow[\phi]{\overrightarrow{\Pi}} P_2$$

describes an event, in order to express that this event is *positively catalyzed* by an element $c$ we can modify the rewrite rule as follows.

If $P_1 \equiv P_1' \mid X$, the type list of $X$ is $\Pi_X$ and the weighting function $\phi$ is $\lambda\overrightarrow{n}\,\overrightarrow{n_X}.e$, where $\overrightarrow{n}$ takes into account the types of the elements occurring in $P_1'$ and $\overrightarrow{n_X}$ takes into account the types of the elements occurring in $X$, we define:

- $\Pi_X'$ as the list whose head is $\bar{t}_c$ and whose tail is $\Pi_X$,
- $\phi' = \lambda\overrightarrow{n}n_c\overrightarrow{n_X}.e \times (n_c \times k + 1)$,

where $k$ is the acceleration due to the presence of one positive catalyzer $c$. The new rule is obtained from the old one by replacing $\Pi_X'$ and $\phi'$ to $\Pi_X$ and $\phi$, respectively.

Otherwise if $P_1 \not\equiv P_1' \mid X$, the new rule is:

$$P_1 \mid X \xrightarrow[\phi']{\overrightarrow{\Pi}^\frown\langle\langle\bar{t}_c\rangle\rangle} P_2 \mid X$$

where $^\frown$ represents list concatenation and if $\phi = \lambda\overrightarrow{n}.e$, then $\phi' = \lambda\overrightarrow{n}n_c.e \times (n_c \times k + 1)$.

Similarly we can represent the effect of an *inhibitor* using $\phi' = \lambda\overrightarrow{n}n_c.e \times \frac{1}{\text{if } n_c=0 \text{ then } 1 \text{ else } (n_c \times k)}$.

We can also represent in one rule both positive and negative catalyzers. For example to add the effect of a positive catalyzer $c$ and an inhibitor $d$ to the rule $P_1 \xrightarrow[\phi]{\overrightarrow{\Pi}} P_2$ if $P_1 \equiv P_1' \mid X$ and $\Pi_X$, $\phi$ are as above we define:

- $\Pi_X' = \langle\bar{t}_c, \bar{t}_d\rangle^\frown\Pi_X$,
- $\phi' = \lambda\overrightarrow{n}n_c n_d\overrightarrow{n_X}.e \times \frac{n_c \times k + 1}{\text{if } n_d=0 \text{ then } 1 \text{ else } n_d \times k'}$,

where $k$ is the acceleration due to the presence of one positive catalyzer $c$ and $k'$ is the deceleration due to the presence of one inhibitor $d$.

Otherwise if $P_1 \not\equiv P_1' \mid X$, the new rule is:

$$P_1 \mid X \xrightarrow[\phi']{\overrightarrow{\Pi} \frown \langle\langle \bar{t}_c, \bar{t}_d \rangle\rangle} P_2 \mid X$$

where if $\phi = \lambda \overrightarrow{n}.e$, then $\phi' = \lambda \overrightarrow{n} n_c n_d.e \times \frac{n_c \times k+1}{\text{if } n_d=0 \text{ then } 1 \text{ else } n_d \times k'}$.

Looking at the previous examples, we claim that our formalism enlightens better than other formalisms the duality between the roles of positive and negative catalyzers.

### Recovering Gillespie's Framework

We might restrict TSCLS in order to match Gillespie's framework. Since we just need to deal with simple molecular populations, we restrict our calculus eliminating the sequencing and the looping operators. We denote by $\mathcal{T}_G$ the infinite set of terms representing Gillespie's molecular populations.

**Definition 6.2.6.** *$\mathcal{T}_G$ is the infinite set of TSCLS terms built as the parallel composition of atomic elements.*

The usual notation for chemical reactions can be expressed by:

$$\ell_1 s_1 + \ldots + \ell_m s_m \xrightarrow{k} \ell_1' p_1 + \ldots + \ell_n' p_n \tag{6.2}$$

where $s_i$ and $p_i$ are the reagents and product molecules, respectively, $\ell_i, \ell_i'$ are the stoichiometric coefficients and $k$ is the kinetic constant.

We denote by $\mathcal{P}_G$ the infinite set of patterns built as the parallel composition of atomic elements and exactly one variable. In particular, we restrict to rewrite rules modeling chemical reactions of the shape of rule (6.2). Namely, assuming that each species of the molecular population has a different type, and, in particular, $t_1, \ldots, t_m$ are the types of $s_1, \ldots, s_m$, a chemical reaction of the form described by rule (6.2) can be expressed by the following TSCLS rewrite rule:

$$\ell_1 \times s_1 \mid \ldots \mid \ell_m \times s_m \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_1, \ldots, \bar{t}_m \rangle\rangle} \ell_1' \times p_1 \mid \ldots \mid \ell_n' \times p_n \mid X \tag{6.3}$$

where $\ell \times s$ stands for a parallel composition $s \mid \ldots \mid s$ of length $\ell$ and similarly for $\ell \times p$.

We now need to define the weighting function $\phi$ of rule (6.3) used to model Gillespie's collision based stochastic simulation algorithm. Intuitively, items (i) and (ii) of Section 6.2.1 already go in this direction (they actually define particular subcases of general chemical reactions expressed by rule (6.3)).

In particular, the collision based framework defined by Gillespie, when the stoichiometry $\ell$ of a reagent is greater than 1, picks one of all the possible combinations of $\ell$ reagents. This leads to binomial distributions of the reagents involved. Namely, we define the weighting function $\phi$ as:

$$\phi = \lambda n_1 \ldots n_m . \binom{n_1 + \ell_1}{\ell_1} \times \ldots \times \binom{n_m + \ell_m}{\ell_m} \times k \tag{6.4}$$

where $k$ is the kinetic constant of the modeled chemical reaction.

By construction, the following holds.

**Proposition 6.2.7.** *Molecular populations defined as $\mathcal{T}_G$ terms with a fixed set of rules of the shape of rule (6.3) interpret Gillespie's framework for the evolution of chemically reacting systems into TSCLS.*

Even if Gillespie's method is defined for simple populations of species, it has been greatly reused in more complex frameworks, e.g. in calculi where compartmentalization and linked structures where taken into account (see, for example, [36, 47, 11, 33, 32, 51]). It is debatable whether such an extension of the usage of Gillespie's method is still correct: the assumptions behind this method are quite strict and considering as same collisions happening between free molecules and molecules bound on a membrane or a protein structure could not always result in a faithful model. We can manage this kind of situations with ad-hoc instantiations of the weight function, allowing us to define more general evolutions than the ones ruled by the law of mass action, (see, e.g., items (iii) and (iv) of Section 6.2.1 and the example about cell division in Section 6.2.1). There are also some cases in which Gillespie's method appears to be reasonably applicable also when its strict assumptions are not fully satisfied. As an example, see the lactose operon case study in Section 6.3, in which, on the lines of [11], we apply a Gillespie based analysis also to rules involving compartments (rules R13 and R14).

## Cell Division: an Example of Multi-Match Patterns

We have just seen how the weighting function introduced in our stochastic semantics can be used to interpret the classical Gillespie's model. We have also seen how more complex interactions can be modeled (see items (iii) and (iv) of Section 6.2.1). In this subsection we consider a more complicated but intriguing case.

In [51], interesting considerations about the structure of rewrite rules are raised. Actually, different conditions can be placed on the structure of the patterns, some of them might be natural when modeling biological systems, some might be not.

Consider the following rule modeling the splitting of a cell and the distribution of its content to the newly produced cells (abstract away, for the moment, from the type list and the weighting function):

$$(\widetilde{x} \cdot \widetilde{y})^{\circlearrowleft} \rfloor (X \mid Y) \to (\widetilde{x})^{\circlearrowleft} \rfloor X \mid (\widetilde{y})^{\circlearrowleft} \rfloor Y$$

The left pattern contains multiple variables within the same sub-term giving rise to several different variables instantiations for a same term. For example some possible matchings for the term $(a \cdot b \cdot c)^{\circlearrowleft} \rfloor (d \mid e \mid f)$ are:

- $\widetilde{x} = a$, $\widetilde{y} = b \cdot c$, $X = d$, $Y = e \mid f$;
- $\widetilde{x} = a \cdot b$, $\widetilde{y} = c$, $X = d \mid e$, $Y = f$;
- $\widetilde{x} = b$, $\widetilde{y} = c \cdot a$, $X = e$, $Y = d \mid f$.

In such cases it is not clear how the stochastic rate should be parceled out among the possible matches of the four variables. Gillespie's method, which does not deal with compartments, non linear and multi-match rules, could not be used in this kind of situations. In [51, 33] this kind of patterns are prevented and such a rule could not be used to directly model a natural biological phenomenon such as cell division. In our framework, we might resort to the weighting function to deal with this kind of situations.

To add some detail, we refer to the splitting example proposed in [51]. In [62], Rosenfeld et al. propose a methodology to analyze the gene regulation function of a particular protein. They start by considering a high concentration of a repressor protein within a single cell. During cell division, each daughter cell receives approximately one half the population of the repressor.[2] As a consequence, after few divisions, the concentration of the repressor becomes low enough to trigger the production of the target protein.

Abstracting away the set of reactions occurring inside the cell and leading the gene expression, we focus on the rule modeling cell division. Consider a simple cell containing a certain number, say $n$, of repressor proteins: $(cell)^{\circlearrowleft} \rfloor (n \times rep)$. Let $t_r$ be the type of the repressor protein, a TSCLS rule modeling a split distributing about $n/2$ repressor proteins to the two freshly produced cells could be defined as follows:

$$(cell)^{\circlearrowleft} \rfloor (X \mid Y) \xrightarrow[\phi]{\langle\langle \bar{t}_r \rangle \langle \bar{t}_r \rangle\rangle} (cell)^{\circlearrowleft} \rfloor X \mid (cell)^{\circlearrowleft} \rfloor Y$$

with the weighting function:

$$\phi = \lambda n_1 n_2 . \frac{k}{1 + |n_1 - n_2| \times k'}$$

where $k$ and $k'$ are used to weight the distribution of the repressor proteins to the new cells (the more far is the partition from the ideal half, the lower the value returned by $\phi$).

The example could be extended in the natural way to take into account any other species within the dividing cell. For the sake of simplicity, we presented here a very naive example of partitioning, more complex functions could be defined to randomly distribute the population of each species of a cell between its two children.

## 6.3 An Application: The Lactose Operon

As application, we study the well-known regulation process of the lactose operon in *Escherichia coli.* We borrow the model and its details from [11], the stochastic version of CLS summarized in Section 2.3.

E. coli is a bacterium often present in the intestine of many animals. It is one of the most deeply studied of all living things and it is a favorite organism for genetic engineering. Cultures of E. coli can be made to produce unlimited quantities of the product of an introduced gene. As most bacteria,

[2]For simplicity we do not consider a detailed volumetrical analysis. We just suppose the volume of the cell increases during the mitosis phase and that the two resulting daughter cells have the same volume of the mother cell.

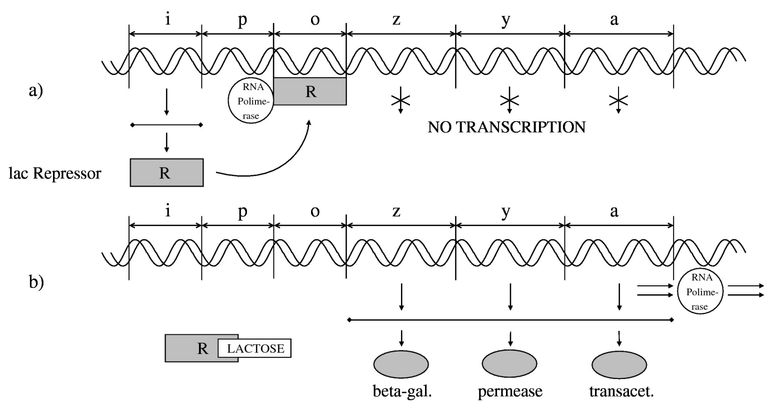

Figure 6.2: The regulation process in the Lac Operon.

E.coli is often exposed to a constantly changing physical and chemical environment, and reacts to changes in its environment through changes in the kinds of enzymes it produces. In order to save energy, bacteria do not synthesize degradative enzymes unless the substrates for these enzymes are present in the environment. For example, E. coli does not synthesize the enzymes that degrade lactose unless lactose is in the environment. This result is obtained by controlling the transcription of some genes into the corresponding enzymes.

Two enzymes are involved in lactose degradation: the *lactose permease*, which is incorporated in the membrane of the bacterium and actively transports the sugar into the cell, and the *beta galactosidase*, which splits lactose into glucose and galactose. The bacterium produces also the *transacetylase* enzyme, whose role in the lactose degradation is marginal.

The sequence of genes in the DNA of E. coli which produces the described enzymes, is known as the *lactose operon*.

The first three genes of the operon (i, p and o) regulate the production of the enzymes, and the last three (z, y and a), called *structural genes*, are transcribed (when allowed) into the mRNA for beta galactosidase, lactose permease and transacetylase, respectively.

The regulation process is as follows (see Figure 6.2): gene i encodes the *lac Repressor*, which, in the absence of lactose, binds to gene o (the *operator*). Transcription of structural genes into mRNA is performed by the RNA polymerase enzyme, which usually binds to gene p (the *promoter*) and scans the operon from left to right by transcribing the three structural genes z, y and a into a single mRNA fragment. When the lac Repressor is bound to gene o, it becomes an obstacle for the RNA polymerase, and the transcription of the structural genes is not performed. On the other hand, when lactose is present inside the bacterium, it binds to the Repressor and this cannot stop anymore the activity of the RNA polymerase. In this case the transcription is performed and the three enzymes for lactose degradation are synthesized.

### 6.3.1 Typed Stochastic CLS Model

A detailed mathematical model of the regulation process can be found in [69]. It includes information on the influence of lactose degradation on the growth of the bacterium.

We give a TSCLS model of the gene regulation process, with stochastic rates taken from [68]. We model the membrane of the bacterium as the looping sequence $(m)^{\circlearrowleft}$, where the alphabet symbol $m$ generically denotes the whole membrane surface in normal conditions. Moreover, we model the lactose operon as the sequence $lacI \cdot lacP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA$ ($lacI–A$ for short), in which each symbol corresponds to a gene. We replace $lacO$ with $RO$ in the sequence when the lac Repressor is bound to gene o, and $lacP$ with $PP$ when the RNA polymerase is bound to gene p. When the lac Repressor and the RNA polymerase are unbound, they are modeled by the symbols $repr$ and $polym$, respectively. We model the mRNA of the lac Repressor as the symbol $Irna$, a molecule of lactose as the symbol $LACT$, and beta galactosidase, lactose permease and transacetylase enzymes as symbols $betagal, perm$ and $transac$, respectively. Finally, since the three structural genes are transcribed into a single mRNA fragment, we model such mRNA as a single symbol $Rna$.

The transcription of the DNA, the binding of the lac Repressor to gene o, and the interaction between lactose and the lac Repressor are modeled by the following set of stochastic typed rewrite rules:

$$lacI–A \mid X \xrightarrow[\phi]{\langle\langle\rangle\rangle} lacI–A \mid Irna \mid X \tag{R1}$$

where $\phi = 0.02$.

$$Irna \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_I \rangle\rangle} Irna \mid repr \mid X \tag{R2}$$

where $t_I$ is the type of $Irna$ and $\phi = \lambda n.(n+1) \times 0.1$.

$$lacI\text{–}A \mid polym \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_{po} \rangle\rangle} lacI \cdot PP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA \mid X \tag{R3}$$

where $t_{po}$ is the type of $polym$ and $\phi = \lambda n.(n+1) \times 0.1$.

$$lacI \cdot PP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA \mid X \xrightarrow[\phi]{\langle\langle\rangle\rangle} lacI\text{–}A \mid polym \mid X \tag{R4}$$

where $\phi = 0.01$.

$$lacI \cdot PP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA \mid X \xrightarrow[\phi]{\langle\langle\rangle\rangle} lacI\text{–}A \mid polym \mid Rna \mid X \tag{R5}$$

where $\phi = 20$.

$$Rna \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_R \rangle\rangle} Rna \mid betagal \mid perm \mid transac \mid X \tag{R6}$$

where $t_R$ is the type of $Rna$ and $\phi = \lambda n.(n+1) \times 0.1$.

$$lacI\text{–}A \mid repr \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_r \rangle\rangle} lacI \cdot lacP \cdot RO \cdot lacZ \cdot lacY \cdot lacA \mid X \tag{R7}$$

where $t_r$ is the type of $repr$ and $\phi = \lambda n.(n+1) \times 1$.

$$lacI \cdot PP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA \mid repr \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_r \rangle\rangle} lacI \cdot PP \cdot RO \cdot lacZ \cdot lacY \cdot lacA \mid X \tag{R8}$$

where $t_r$ is the type of $repr$ and $\phi = \lambda n.(n+1) \times 1$.

$$lacI \cdot lacP \cdot RO \cdot lacZ \cdot lacY \cdot lacA \mid X \xrightarrow[\phi]{\langle\langle\rangle\rangle} lacI\text{–}A \mid repr \mid X \tag{R9}$$

where $\phi = 0.01$.

$$lacI \cdot PP \cdot RO \cdot lacZ \cdot lacY \cdot lacA \mid X \xrightarrow[\phi]{\langle\langle\rangle\rangle} lacI \cdot PP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA \mid repr \mid X \tag{R10}$$

where $\phi = 0.01$.

$$repr \mid LACT \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_r, \bar{t}_L \rangle\rangle} RLACT \mid X \tag{R11}$$

where $t_r$ and $t_L$ are the types of *repr* and $LACT$ and $\phi = \lambda n_1 n_2.(n_1 + 1) \times (n_2 + 1) \times 0.005$.

$$RLACT \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_{RL} \rangle\rangle} repr \mid LACT \mid X \tag{R12}$$

where $t_{RL}$ is the type of $RLACT$ and $\phi = \lambda n.(n + 1) \times 0.1$.

Rules (R1) and (R2) describe the transcription and translation of gene i into the lac Repressor (assumed for simplicity to be performed without the intervention of the RNA polymerase). Rules (R3) and (R4) describe binding and unbinding of the RNA polymerase to gene p. Rules (R5) and (R6) describe the transcription and translation of the three structural genes. Transcription of such genes can be performed only when the sequence contains *lacO* instead of *RO*, that is when the lac Repressor is not bound to gene o. Rules (R7)-(R10) describe binding and unbinding of the lac Repressor to gene o. Finally, rules (R11) and (R12) describe the binding and unbinding, respectively, of the lactose to the lac Repressor.

The following rules describe the behavior of the three enzymes for lactose degradation:

$$(\widetilde{x})^{\circlearrowleft} \rfloor (perm \mid X) \mid Y \xrightarrow[\phi]{\langle\langle\rangle, \langle \bar{t}_{pe} \rangle, \langle\rangle\rangle} (perm \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \mid Y \tag{R13}$$

where $t_{pe}$ is the type of *perm* and $\phi = \lambda n.(n + 1) \times 0.1$.

$$(\widetilde{x})^{\circlearrowleft} \rfloor X \mid LACT \mid Y \xrightarrow[\phi]{\langle\langle \widetilde{t}_{pe} \rangle, \langle\rangle, \langle \bar{t}_L \rangle\rangle} (\widetilde{x})^{\circlearrowleft} \rfloor (LACT \mid X) \mid Y \tag{R14}$$

where $t_{pe}$ and $t_L$ are the types of *perm* and $LACT$, respectively, and $\phi = \lambda n_1 n_2.n_1 \times (n_2 + 1) \times 0.001$.

$$LACT \mid X \xrightarrow[\phi]{\langle\langle \bar{t}_L, \bar{t}_b \rangle\rangle} GLU \mid GAL \mid X \tag{R15}$$

where $t_L$ and $t_b$ are the types of $LACT$ and *betagal*, and $\phi = \lambda n_1 n_2.(n_1 + 1) \times n_2 \times 0.001$.

$$\begin{array}{ll}
 & EcoliLact \\
\xrightarrow{\text{R3, } 30\times 0.1} & 10000 \times LACT \mid (m)^{\circlearrowleft} \rfloor (lacI \cdot PP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA \mid \\
 & 29 \times polym \mid 100 \times repr) \\
\xrightarrow{\text{R5, } 20} & 10000 \times LACT \mid (m)^{\circlearrowleft} \rfloor (lacI\text{–}A \mid 30 \times polym \mid 100 \times repr \mid \\
 & Rna) \\
\xrightarrow{\text{R6, } 0.1} & 10000 \times LACT \mid (m)^{\circlearrowleft} \rfloor (lacI\text{–}A \mid 30 \times polym \mid 100 \times repr \mid \\
 & Rna \mid betagal \mid perm \mid transac) \\
\xrightarrow{\text{R13, } 0.1} & 10000 \times LACT \mid (perm \cdot m)^{\circlearrowleft} \rfloor (lacI\text{–}A \mid 30 \times polym \mid \\
 & 100 \times repr \mid Rna \mid betagal \mid transac) \\
\xrightarrow{\text{R14, } 10000\times 0.001} & 9999 \times LACT \mid (perm \cdot m)^{\circlearrowleft} \rfloor (lacI\text{–}A \mid 30 \times polym \mid \\
 & 100 \times repr \mid Rna \mid betagal \mid transac \mid LACT) \\
\xrightarrow{\text{R15, } 0.001} & 9999 \times LACT \mid (perm \cdot m)^{\circlearrowleft} \rfloor (lacI\text{–}A \mid 30 \times polym \mid \\
 & 100 \times repr \mid Rna \mid betagal \mid transac \mid GLU \mid GAL).
\end{array}$$

Figure 6.3: An example of stochastic reduction.

Rule (R13) describes the incorporation of the lactose permease in the membrane of the bacterium, rule (R14) the transportation of lactose from the environment to the interior performed by the lactose permease, and rule (R15) the decomposition of the lactose into glucose (denoted *GLU*) and galactose (denoted *GAL*) performed by the beta galactosidase.

The initial state of the bacterium when no lactose is present in the environment and when 10000 molecules of lactose are present are modeled, respectively, by the following terms (where $n \times T$ stands for a parallel composition $T \mid \ldots \mid T$ of length $n$):

$$Ecoli \ ::= \ (m)^{\circlearrowleft} \rfloor (lacI\text{–}A \mid 30 \times polym \mid 100 \times repr) \qquad (6.5)$$

$$EcoliLact \ ::= \ Ecoli \mid 10000 \times LACT \qquad (6.6)$$

Now, starting from the term *EcoliLact*, a possible stochastic trace generated by our semantics, given the rules above, is shown in Figure 6.3[3].

[3]For simplicity we just show the rate of the transition reaching the target state considered in the trace. We avoid to report explicitly the whole exit rate from a given term, which should be computed, following the standard simulation algorithm, by summing up the rates for all the possible target states. For the sake of readability, we also show, on the transitions, the labels of the rules leading the state change.

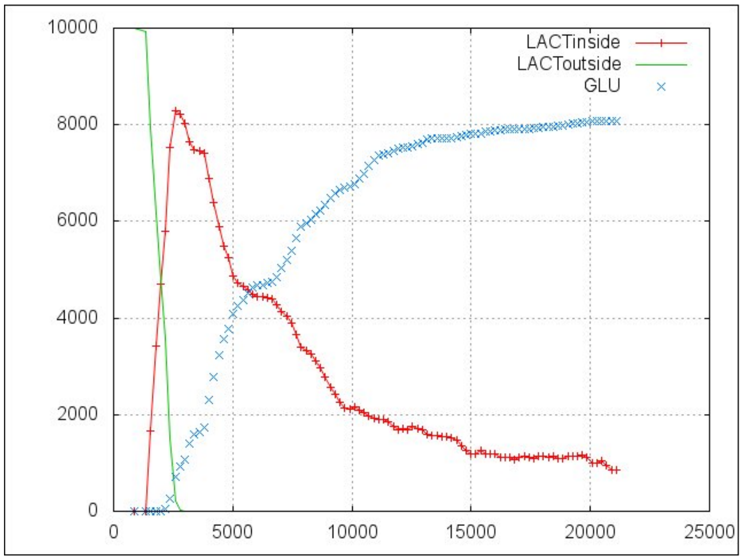

Figure 6.4: Simulation results: absorption and degradation of lactose into glucose.

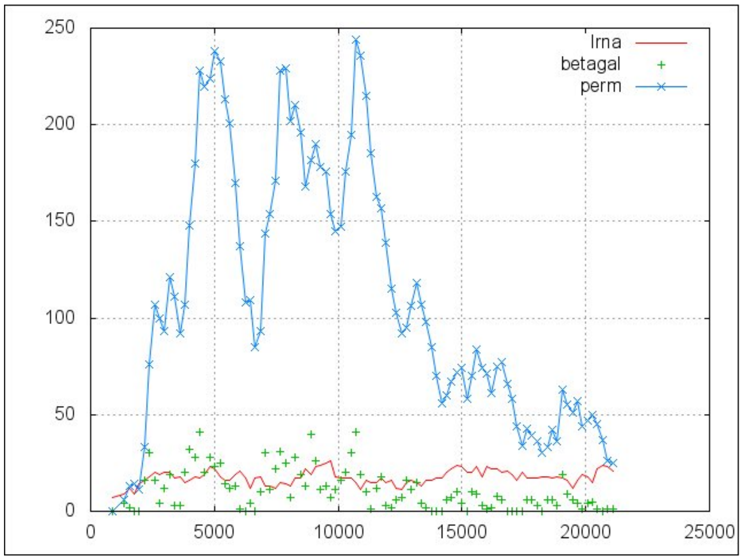

Figure 6.5: Simulation results: production of enzymes.

In Figure 6.4 and Figure 6.5 we show the results of a TSCLS simulation of the term *EcoliLact* obtained with a prototype simulator for TSCLS written in JAVA. For a more realistic simulation we added to the model also the rules describing the spontaneous degradation of the elements involved in the model[4]. In particular, in Figure 6.4, we show the absorption of lactose showing the concentrations of lactose outside and inside the bacterium and, inside the bacterium, the degradation of lactose into glucose (passage of time, per seconds, is modeled on the X, number of elements is given by the Y axis). In Figure 6.5, we show the number of the enzymes $Irna$, *betagal* and *perm* (notice how the production of the *perm* enzyme inside the bacterium is activated after the absorption of the lactose).

## 6.4 Conclusions

This chapter presents a first proposal for using a type abstraction in describing quantitative aspects of biological systems.

Let us compare the SCLS with our calculus. The most obvious difference is that our rules, similar to what happens in [23] for a variant of the ambient calculus, are equipped with *rate functions*, rather than rate constants. This is a big advantage, because such functions allow us to define kinetics that are more complex than the standard mass-action ones, as shown in Section 6.2.1.

Another obvious difference is the counting mechanism: ours, based on types, is simpler than the one of SCLS in practice: while a single pattern may have several, though isomorphic, matches within a CLS term, in Typed Stochastic CLS we state explicitly the types of the elements whose occurrences affect the speed of a reduction. We do not need the abstractions introduced in SCLS (i.e. Concrete Terms, Supports and Occurrences, see Section 2.3) but just the simpler notions of ptype and stype, in Definition 6.2.1. On the other side, by abstracting sequences with the multiset of the types of their elements we lose the information on the ordering of the elements. Therefore, we cannot define a function computing correctly the kinetic constant for the example in Remark 2.3.2, since the function should depend on the number of $a$'s that in both cases is the same. However, as shown in Section 6.2.1, for a restricted set of terms we can correctly real-

[4]omitted here for simplicity, a complete description of the simulated model is available at: `http://www.di.unito.it/~giannini/TSCLSim/`

ize the "mass-action law". The counting mechanism affects also the rule schemata. In fact, as discussed in Remark 6.2.2, the hole of our contexts encompass a whole compartment, while in SCLS the hole of a context may be a subterm of a compartment. In order to regain the lost expressiveness, as we can see from the examples, we must add a $\mid X$ to the patterns in the left-hand-side and right-hand-side of a rule.

All these differences have simplified the development of our automatic simulation tool. Note that the simulator developed for the SCLS calculus[5], for efficiency reasons, does not implement the complex counting of matches defined in [11], but computes the kinetic constant of a reduction by counting the number of matches based on the occurrences of the elements of the pattern present in the term, as we described in Section 6.2.1. A comparison between the efficiencies of the SCLS and the TSCLS simulators is meaningless considering the differences between the functionalities of the semantics presented in [11] and those implemented in the SCLS simulator.

[5]available at `http://www.di.unipi.it/msvbio/wiki/sclsm`

# Chapter 7

# A Minimal OO Calculus for Modelling Biological Systems

## 7.1 Introduction

Homogeneous biological entities are usually named according to their behavior. Enzymes are proteins that catalyze (i.e. increase the rates of) chemical reactions, receptors are proteins embedded in a membrane to which one or more specific kinds of signaling molecules may attach producing a biological response, hydrolases are enzymes that catalyze the hydrolysis of a chemical bond, and so on. Subsequently, chemical elements are classified following these groups: for example, the *lactase* is a *hydrolase*, then its peculiarity with respect to the other biological entities is that it catalyzes the hydrolysis of a particular molecule. Such classification suggests Computer Science types: each biological entity is classified with a type, containing the sound operations for it. Moreover, this classification is useful for checking the correctness of chemical reactions. Lactase is not just a hydrolase, but a *glycoside* hydrolase, i.e. it catalyzes the hydrolysis of the glycosidic linkage of a sugar to release smaller sugars: if the substrate or the products are not sugars, somewhere there is an error. Type checking tools do the same, checking the correspondence between the types of the arguments and the types of the parameters in a module call operation. Finally, the biological classification follows a subtype relation. Lactase hydrolyze the lactose, that is a disaccharide: since disaccharides are a subtype of sugar, the hydrolysis operation associated to the glycoside hydrolase is correct.

Here we present the Minimal Object-Oriented Core Calculus for term-rewriting formalisms, i.e. formalisms based on term rewriting, proposed in [18]: it aims to model the notion of types used in biology as above described. We implement only the Object-Oriented paradigm skills that, in our view, are basic in modeling biological systems, that is encapsulation, method invocation, subtyping and a simple inheritance. The purpose of this calculus is to facilitate the organizations of rules, and to improve their re-use, in the model or even in other models.

## 7.2 Core Calculus

A rewrite system is composed by a term, representing the structure of the modeled system, and a series of reduction rules, representing the possible evolutions of the system: depending on the formalism, these rules can be embedded in terms, like in P Systems, or defined in a separate part, like in the Calculus of Looping Sequences; note that an hybrid solution is the Calculus of Looping Sequences with Local Rules proposed in Chapter 3.

In our core calculus, a class contains methods (*encapsulation*) and extends another class (*subtyping*), inheriting all its methods (*inheritance*). Methods are formed by a sequence of variables, the *arguments*, and a sequence of *reduction rules* containing these variables. They are called on symbols of the model, representing biological entities, with a sequence of values as arguments (*method invocation*). A method invocation is replaced by the reduction rules of the method, in which the variables are replaced by the values used as arguments. These reduction rules are then used for the evolution of the model.

For example, the hypothetical class of glycoside hydrolase contains a method to hydrolyze a sugar into two sugars, all of them passed as arguments. This method contains the sequence of reduction rules that models hydrolysis. We assign to lactase the glycoside hydrolase type, and then call on it the hydrolysis method, passing as arguments the lactose and the sugar products. By invocation, we obtain the reduction rules specific for lactase, that will be used for the evolution of the model.

In the remainder of this Section we present the formal definition of the calculus. Its syntax, definitions and rules are inspired by the ones proposed by Igarashi, Pierce and Wadler for Featherweight Java [46], a minimal core calculus for modeling the Java Type System.

### 7.2.1 Syntax

*Syntax*

| | | | |
|---|---|---|---|
| $CT$ | ::= | | *class table declaration* |
| | | $\overline{CL}$ | |
| $CL$ | ::= | | *class declaration* |
| | | `class C extends D`$\{\overline{M}\}$ | $(\mathtt{C} \neq \mathtt{Object})$ |
| $M$ | ::= | | *method declaration* |
| | | $m(\overline{\mathtt{C}}\ \overline{x})\ \overline{R}$ | |
| $R$ | ::= | | *reduction rule declaration* |
| | | according to the formalism syntax | |
| | | contains variables, values and `this` | |
| $I$ | ::= | | *method invocation* |
| | | $v.m(\overline{v})$ | |
| $x$ | | | *variable* |
| $v$ | | | *value* |
| `this` | | | *this* |

Figure 7.1: Syntax

The syntax is given in Figure 7.1. The metavariables `C` and `D` range over class names; $m$ ranges over method names; $CL$ ranges over class declarations; $M$ ranges over method declarations; $R$ ranges over reduction rules; $I$ ranges over method invocations; $x$ ranges over parameter names; $v$ ranges over values, i.e. the symbols of the model. We assume that the set of variables includes the special variable `this`, that identifies the object calling a module. Notice that this is never used as argument of a method. Note that the syntax of reduction rules $R$ is not defined, but it depends on the rewriting formalism used.

**Remark 7.2.1.** *For the sake of generality, in running examples we use the biological rule notation to represent reduction rules, following the simple syn-*

*tax*

$$\begin{array}{lll} E & ::= & \textit{element composition} \\ & & v \ | \ x \ | \ E+E \\ R & ::= & \textit{rule declaration} \\ & & E \ \rightarrow \ E \end{array}$$

*The plus sign represents a juxtaposition between elements, and the arrow represents a chemical reaction. We use the notation $E_1 \rightleftharpoons E_2$ instead of the pair of reduction rules $E_1 \rightarrow E_2$ and $E_2 \rightarrow E_1$.*

We write $\overline{M}$ as shorthand for $M_1 \dots M_n$, and $\overline{\mathtt{C}}$ for $\mathtt{C}_1, \dots, \mathtt{C}_n$ (similarly $\overline{x}$, $\overline{v}$, etc.). We abbreviate operations on pairs of sequences in the same way, writing $\overline{\mathtt{C}}\ \overline{x}$ for $\mathtt{C}_1\ x_1, \dots, \mathtt{C}_n\ x_n$, where $n$ is the length of $\overline{\mathtt{C}}$ and $\overline{x}$. Sequences of parameter names and method declarations are assumed to contain no duplicate names.

The declaration `class C extends D`$\{\overline{M}\}$ introduces a class named `C` having the class `D` as superclass. The new class has the suite of methods $\overline{M}$. The methods declared in `C` are added to the ones declared by `D` and its superclasses, and may override methods with the same name that are already present in `D`, or add new functionalities. The class `Object` has no methods and does not have superclasses.

The method declaration $m(\overline{\mathtt{C}}\ \overline{x})\ \overline{R}$ introduces a method named $m$ with parameters $\overline{x}$ of types $\overline{\mathtt{C}}$. The body of the method is a sequence of reduction rules $\overline{R}$, expressed in the syntax of the formalism. The variables $\overline{x}$ and the special variable `this` are bound in $\overline{R}$.

A class table $CT$ is a mapping from class names `C` to class declarations $CL$. We assume a fixed class table $CT$ satisfying some sanity conditions: (1) $CT(\mathtt{C}) =$ `class C`... for every $\mathtt{C} \in dom(CT)$; (2) $\mathtt{Object} \notin dom(CT)$; (3) for every class name `C` (except `Object`) appearing in $CT$, we have $\mathtt{C} \in dom(CT)$; (4) there are no cycles in the subtype relation induced by $CT$, i.e. a class cannot extends one of its subclasses.

The fixed type environment $\Lambda$ contains the association between values $v$ and their types `C`, written $v : \mathtt{C}$. We assume that $\Lambda$ satisfies some sanity conditions: (1) if $v : \mathtt{C} \in \Lambda$ for some $v$, then $\mathtt{C} \in dom(CT)$; (2) every value in the set of values is associated to exactly one type in $\Lambda$.

**Example 7.2.2.** *We define the class of molecules as follows:*

```
class Molecule extends Object{}
```

*The* `Molecule` *class has the* `Object` *class as superclass, and it does not have methods, i.e. molecules do not have any particular behavior.*

*An enzyme is a protein that catalyze chemical reactions. In an enzymatic reaction, the molecules involved in the reaction (called substrates) are converted into different molecules (called the products), while the enzyme itself is not consumed by the reaction. We define the class of enzymes as follows:*

```
class Enzyme extends Object
{
  action(Molecule S, Molecule P)
    S + this → this + P
}
```

*For the sake of simplicity, in our example an enzyme extends an object rather than a protein, jumping a hierarchy level. According to the enzyme definition, the* `Enzyme` *class contains only one method, action, which converts the variable molecule S (the substrate) into the variable molecule P (the product) in presence of the enzyme (the* `this` *keyword).*

Class tables and environment types are used to create a triple $(CT, \Lambda, P)$, where $P$ is the model, designed according to the formalism specifications, in which all the reduction rules are replaced by method invocations.The class table $CT$ and the type set $\Lambda$ are fixed, i.e. they are determined during the model creation and cannot vary during model evolution.

## 7.2.2 Auxiliary Definitions

For the typing and evaluation of rules, we need a few auxiliary definitions: these are given in Figure 7.2.

The type of a method $m$ in a class `C`, written **mtype**$(m, \texttt{C})$, is the sequence of types $\overline{\texttt{C}}$ of the arguments of the method $m$ defined in the class `C` (or in one of its superclasses, if not defined in `C`). For example,

$$\textbf{mtype}(\textit{action}, \texttt{Enzyme}) \;=\; (\texttt{Molecule}, \texttt{Molecule})$$

The body of a method $m$ in a class `C`, written **mbody**$(m, \texttt{C})$, is a pair $(\overline{x}, \overline{R})$ of a sequence of variables $\overline{x}$ and a sequence of reduction rules $\overline{R}$. The elements of the pair are the arguments and the reduction rules of the method $m$ defined in the class `C` (or in one of its superclasses, if not defined in `C`). For example,

$$\textbf{mbody}(\textit{action}, \texttt{Enzyme}) \;=\; ((S, P), S + \texttt{this} \rightarrow \texttt{this} + P)$$

*Method type lookup*

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\{\overline{M}\} \quad m(\overline{\mathtt{C}}\ \overline{x})\ \overline{R} \in \overline{M}}{\mathbf{mtype}(m, \mathtt{C}) = \overline{\mathtt{C}}}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\{\overline{M}\} \quad m \text{ is not defined in } \overline{M}}{\mathbf{mtype}(m, \mathtt{C}) = \mathbf{mtype}(m, \mathtt{D})}$$

*Method body lookup*

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\{\overline{M}\} \quad m(\overline{\mathtt{C}}\ \overline{x})\ \overline{R} \in \overline{M}}{\mathbf{mbody}(m, \mathtt{C}) = (\overline{x}, \overline{R})}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\{\overline{M}\} \quad m \text{ is not defined in } \overline{M}}{\mathbf{mbody}(m, \mathtt{C}) = \mathbf{mbody}(m, \mathtt{D})}$$

Figure 7.2: Auxiliary Definitions

### 7.2.3 Evaluation

The unique evaluation rule concerns the method invocation $\mathsf{v}.m(\bar{\mathsf{t}})$. If the value $\mathsf{v}$ has type $\mathtt{C}$ in $\Lambda$, and the method $m$ has arguments $\overline{x}$ and body $\overline{R}$ in $\mathtt{C}$, then its evaluation is the sequence of reduction rules $\overline{R}$, in which all the occurrences of the variables $\overline{x}$ are replaced with the values $\bar{\mathsf{t}}$, and all the occurrences of `this` are replaced with the value $\mathsf{v}$. Note that a method invocation is placed in the model instead of reduction rules: once evaluated, the reduction rules of the method becomes the (instantiated) reduction rules of the model.

**Example 7.2.3.** *Phosphoglucose isomerase is an enzyme that catalyzes the conversion of glucose-6-phosphate into fructose 6-phosphate (and vice versa) in the second step of glycolysis. In order to model this behavior, in $\Lambda$ we associate to the value* ph-iso *(the phosphoglucose isomerase) the type* `Enzyme`*, and to the values* glu-6-ph *and* fru-6-ph *(the glucose-6-phosphate and fructose 6-phosphate, respectively) the type* `Molecule`

$$\Lambda = \{\mathsf{ph\text{-}iso} : \mathtt{Enzyme}, \mathsf{glu\text{-}6\text{-}ph} : \mathtt{Molecule}, \mathsf{fru\text{-}6\text{-}ph} : \mathtt{Molecule}\}$$

*Method Invocation*

$$\frac{v : \mathtt{C} \in \Lambda \quad \mathbf{mbody}(m, \mathtt{C}) = (\overline{x}, \overline{R})}{v.m(\overline{t}) \rightarrow [\overline{x} \mapsto \overline{t},\ \mathtt{this} \mapsto v]\overline{R}} \text{ (e-meth)}$$

Figure 7.3: Evaluation

*Instead of the reduction rules, in the model we place the calling of the action method on the* ph-iso *enzyme, using the molecules as arguments*

$$\mathsf{ph\text{-}iso}.action(\mathsf{glu\text{-}6\text{-}ph}, \mathsf{fru\text{-}6\text{-}ph})$$

*Following the evaluation rule in Figure 7.3, this method invocation is replaced by the reduction rule*

$$\mathsf{glu\text{-}6\text{-}ph} + \mathsf{ph\text{-}iso} \rightarrow \mathsf{ph\text{-}iso} + \mathsf{fru\text{-}6\text{-}ph}$$

*As a consequence, we obtain the reduction rule modeling the conversion of glucose-6-phosphate into fructose 6-phosphate. In order to obtain the conversion in the other side, we call the action method on the* ph-iso *enzyme by swapping the arguments*

$$\mathsf{ph\text{-}iso}.action(\mathsf{fru\text{-}6\text{-}ph}, \mathsf{glu\text{-}6\text{-}ph})$$

*This method invocation is then replaced by the reduction rule*

$$\mathsf{fru\text{-}6\text{-}ph} + \mathsf{ph\text{-}iso} \rightarrow \mathsf{ph\text{-}iso} + \mathsf{glu\text{-}6\text{-}ph}$$

*After method evaluation, we obtain the reduction rules of the model, representing the possible evolution of the system.*

## 7.2.4 Typing

The rules for subtyping are formally defined in Figure 7.4. The subtype relation between classes is given by the class declarations in the class table $CT$. The subtype relation is reflexive and transitive.

**Example 7.2.4.** *For* `Enzyme` *and* `Molecule` *classes we derive the following subtype relations:*

| | | |
|---|---|---|
| $\mathtt{Enzyme} <: \mathtt{Enzyme}$ | $\mathtt{Molecule} <: \mathtt{Molecule}$ | (by rule (t-sub1)) |
| $\mathtt{Enzyme} <: \mathtt{Object}$ | $\mathtt{Molecule} <: \mathtt{Object}$ | (by rule (t-sub3)) |

*Subtyping*

$$\texttt{C} <: \texttt{C} \quad \text{(t-sub1)} \qquad \frac{\texttt{C} <: \texttt{D} \quad \texttt{D} <: \texttt{E}}{\texttt{C} <: \texttt{E}} \text{(t-sub2)}$$

$$\frac{CT(\texttt{C}) = \texttt{class C extends D}\{\overline{M}\}}{\texttt{C} <: \texttt{D}} \text{(t-sub3)}$$

Figure 7.4: Subtyping

*Note that since* `Enzyme` *is not a subtype of* `Molecule`*, then an enzyme cannot be a substrate nor a product of the* `Enzyme`*'s action method.*

The typing rules for method invocations and for method and class declarations are given in Figure 7.5. Typing statements for method invocations have the form $\texttt{v}.m(\bar{\texttt{t}})$ `OK`, asserting that the method invocation $\texttt{v}.m(\bar{\texttt{t}})$ is well formed. The typing rule checks that the types of the values used as arguments in a method invocation are subtypes of the types of the arguments required by the method.

Typing statements for method declarations have the form $M$ `OK in C`, and assert that the method declaration $M$ is well formed in the class `C`. The typing rule checks that the reduction rules in the method of a class are well formed, assuming the types of the arguments and the class. Typing statements for class declarations have the form $CL$ `OK`, stating that the class declaration $CL$ is well formed. The typing rule checks that each method declaration in the class is well formed. Since each formalism may have its own constraints for checking the well-formedness of a rule, the modeler must add the proper typing rules, in addition to the ones in Figure 7.5.

**Example 7.2.5.** *As expected, both* `Enzyme` *and* `Molecule` *classes are* `OK`*.*

Note that the inheritance is very simple: a class inherits all the methods of its superclass, and it can modify the body and the arguments of a method declared in its superclass. In this way, lower classes can reuse the names of higher classes methods, i.e. more specialized biological entities can focus and specialize the behavior of more generic biological entities by reusing the name associated to a reduction rule.

*Invocation typing*

$$\frac{v : \mathtt{C} \in \Lambda \quad \mathbf{mtype}(m, \mathtt{C}) = \overline{\mathtt{C}} \quad \overline{t} : \overline{D} \in \Lambda \quad \overline{D} <: \overline{C}}{v.m(\overline{t})\ \mathtt{OK\ in\ C}} \text{ (t-invmeth)}$$

*Method typing*

$$\frac{\overline{x} : \overline{\mathtt{C}}, \mathtt{this} : \mathtt{C} \vdash \overline{R}\ \mathtt{OK}}{m(\overline{\mathtt{C}}\ \overline{x})\ \overline{R}\ \mathtt{OK\ in\ C}} \text{ (t-clmeth)}$$

*Class typing*

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D}\{\overline{M}\} \quad \overline{M}\ \mathtt{OK\ in\ C}}{\mathtt{class\ C\ extends\ D}\{\overline{M}\}\ \mathtt{OK}} \text{ (t-class)}$$

Figure 7.5: Typing

**Example 7.2.6.** *An hydrolase is an enzyme, but it cannot catalyze any reaction except hydrolysis. For this reason, we design hydrolase class as follows:*

```
class Hydrolase extends Enzyme
{
  action(Molecule S, Molecule P1, Molecule P2)
    S + H2O + this → this + P1 + P2
}
```

*The* Hydrolase *class is an extension of the* Enzyme *class that overrides the action method. In this way, the generic catalysis described in the* Enzyme*'s action method is no more available in the* Hydrolase *class, but the override action method makes available only the specific hydrolysis.*

*In a similar way, the glycoside hydrolase is an hydrolase, but its substrate and products are sugars. Then the glycoside hydrolase class is designed as an extension of* Hydrolase *class, that overrides the action method by modifying* *the types of the arguments, from generic molecules to sugars:*

```
class Sugar extends Molecule{}

class GlycosideHydrolase extends Hydrolase
{
  action(Sugar S, Sugar P_1, Sugar P_2)
    S + H_2O + this → this + P_1 + P_2
}
```

## 7.3 Modeling Enzyme Kinetics

In this section we show how our calculus can be used to model biological behaviors. As an example, we design classes and method invocations to describe Michaelis-Menten enzyme kinetic, the two-substrates enzyme kinetic and the competitive inhibition kinetic.

### Michaelis-Menten Model

In the Michaelis-Menten Model, the enzyme reaction is divided in two stages. In the first stage, the substrate $\mathsf{S}$ binds reversibly to the enzyme $\mathsf{E}$, forming the enzyme-substrate complex $\mathsf{ES}$, then in the second one the enzyme catalyzes the chemical step in the reaction and releases the product $\mathsf{P}$:

$$\mathsf{E} + \mathsf{S} \rightleftharpoons \mathsf{ES} \rightarrow \mathsf{E} + \mathsf{P}$$

This basic construct is also used in most complex enzyme reactions. In order to model this behavior, we create two classes, the `Enzyme` class and the `EnzymeCom` class. The first one models an enzyme: it associates itself with a substrate, and produces an enzyme-substrate complex. The second one models an enzyme-substrate complex: it dissociates itself in an enzyme and a product.

```
class Enzyme extends Object        class EnzymeCom extends Enzyme
{                                  {
  ass(Molecule S, EnzymeCom ES)      dis(Enzyme E, Molecule P)
    S + this → ES                      this → E + P
}                                  }
```

Since an enzyme-substrate complex can act as an enzyme, the `EnzymeCom` class extends the `Enzyme` class. In this way, the `EnzymeCom` class inherits from `Enzyme` the *ass* method by auxiliary definitions.
The type environment is

$$\Lambda = \{\mathsf{E} : \texttt{Enzyme}, \mathsf{ES} : \texttt{EnzymeCom}, \mathsf{S} : \texttt{Molecule}, \mathsf{P} : \texttt{Molecule}\}$$

The method invocations for reproducing the described behavior are

$$\mathsf{E}.\mathit{ass}(\mathsf{S}, \mathsf{ES}) \qquad \mathsf{ES}.\mathit{dis}(\mathsf{E}, \mathsf{S}) \qquad \mathsf{ES}.\mathit{dis}(\mathsf{E}, \mathsf{P})$$

## Two-substrates Enzymes

Several enzymes catalyze a reaction, usually divided into three stages, between two substrates. In the first stage, the substrate S1 binds reversibly to the enzyme E, forming the enzyme-substrate complex ES1. In the second stage, the substrate S2 binds reversibly to the enzyme-substrate complex ES1, forming the enzyme-substrate complex ES1S2. Finally, in the last stage the enzyme complex ES1S2 catalyzes the chemical step in the reaction and releases the product P:

$$\mathsf{E} + \mathsf{S1} \rightleftharpoons \mathsf{ES1} + \mathsf{S2} \rightleftharpoons \mathsf{ES1S2} \rightarrow \mathsf{E} + \mathsf{P}$$

Note that this is only one of all the possible interactions between an enzyme and two substrates. To model this behavior, we use the classes `Enzyme` and `EnzymeCom`, and we assign the following types:

$$\begin{aligned}\Lambda = \; & \{\mathsf{E} : \texttt{Enzyme}, \mathsf{ES1} : \texttt{EnzymeCom}, \mathsf{ES1S2} : \texttt{EnzymeCom}, \\ & \mathsf{S1} : \texttt{Molecule}, \mathsf{S2} : \texttt{Molecule}, \mathsf{P} : \texttt{Molecule}\}\end{aligned}$$

The method invocations are the following:

$$\begin{gathered}\mathsf{E}.\mathit{ass}(\mathsf{S1}, \mathsf{ES1}) \qquad \mathsf{ES1}.\mathit{dis}(\mathsf{E}, \mathsf{S1}) \qquad \mathsf{ES1}.\mathit{ass}(\mathsf{S2}, \mathsf{ES1S2}) \\ \mathsf{ES1S2}.\mathit{dis}(\mathsf{ES1}, \mathsf{S2}) \qquad \mathsf{ES1S2}.\mathit{dis}(\mathsf{E}, \mathsf{P})\end{gathered}$$

## Competitive Inhibition

Enzyme reaction rates can be decreased by molecules called enzyme inhibitors. There exist a lot of inhibitors kinetics: among the others, we study the Competitive Inhibition, in which an inhibitor and a substrate compete for the enzyme (i.e. they cannot bind at the same time). In fact, the inhibitor $\mathsf{I}$ binds to enzyme $\mathsf{E}$ producing the complex $\mathsf{EI}$, and stops a substrate $\mathsf{S}$ from entering the enzyme's active site and producing the complex $\mathsf{ES}$:

$$\begin{gathered}\mathsf{E}+\mathsf{S} \rightleftharpoons \mathsf{ES} \rightarrow \mathsf{E}+\mathsf{P}\\ \mathsf{E}+\mathsf{I} \rightleftharpoons \mathsf{EI}\end{gathered}$$

This case is an extension of the Michaelis-Menten Model, and is modeled by adding to the previous model the following type environment and method invocations:

$$\Lambda' = \{\mathsf{EI} : \mathtt{EnzymeCom}, \mathsf{I} : \mathtt{Molecule}\}$$

$$\mathsf{E}.\mathit{ass}(\mathsf{I}, \mathsf{EI}) \qquad \mathsf{EI}.\mathit{dis}(\mathsf{E}, \mathsf{I})$$

## 7.4 Implementation in Term-Rewriting Formalisms

The calculus in this Chapter is designed to be applied to the most popular term-rewriting formalisms for modeling biological systems, following four steps:

1. set the syntax of reduction rules of the term-rewriting formalism as the syntax of reduction rules of the core calculus;

2. if the reduction rules must respect certain conditions handled by typing, then add the proper typing rules for checking their well-formedness;

3. define the class table $CT$, and assign types to values in the type environment $\Lambda$ according to their biological behavior;

4. create a triple $(CT, \Lambda, P)$, where $P$ is a model designed according to the formalism specifications, except for the reduction rules, that are replaced by method invocations.

After the evaluation of the method invocations in $P$, we obtain the model $P'$ in the formalism form, in which all the reduction rules are consistent with the biological classification and behavior defined in $CT$ and $\Lambda$.

### 7.4.1 An Application: Porins

As an example of implementation, we model the Porins behavior in two different term-rewriting formalisms: the Calculus of Looping Sequences (CLS) and the P systems. Porins are proteins that cross a cellular membrane and act as a pore through which molecules can diffuse. The molecules which diffuse across the porin depends on the porin itself. Among the porins, aquaporins selectively conduct water molecules in and out of the cell, while preventing the passage of ions and other solutes. Some of them, known as aquaglyceroporins, transport also other small uncharged solutes, such as glycerol, CO2, ammonia and urea across the membrane (see [45]). We design the `Porin` class to model the porin behavior, and we present an example of triple and its evaluation, in CLS and P systems formalisms. In particular, we model two kinds of aquaporins: the former transports only water, while the latter transports both urea and water.

#### Calculus of Looping Sequences

A CLS model (see Chapter 2) is a pair $(T, \mathcal{R})$, where $T$ is the term depicting the initial state of the system, and $\mathcal{R}$ is the set of rewrite rules. Since in $CLS$ the reduction rules have the form $P \to P$, the rule syntax of a class becomes

$$R ::= P \to P.$$

Using classes and methods, the set $\mathcal{R}$ becomes a set of method invocations, $\mathcal{R} = \{\overline{I}\}$, which must be evaluated in an initial phase of system initialization, before the evaluation of terms, in order to obtain the rewrite rules of the model.

A class modeling the porin behavior with rewrite rules in CLS syntax is the following:

```
class Porin extends Object
{
  $in$(Molecule $S$)
    $S \mid (\texttt{this} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \to (\texttt{this} \cdot \widetilde{x})^{\circlearrowleft} \rfloor (S \mid X)$

  $out$(Molecule $S$)
    $(\texttt{this} \cdot \widetilde{x})^{\circlearrowleft} \rfloor (S \mid X) \to S \mid (\texttt{this} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X$
}
```

We use the symbols $\mathsf{w}$ for water, $\mathsf{u}$ for urea, $\mathsf{AW}$ for the aquaporin that transports only water and $\mathsf{AWU}$ for the aquaporins that transports both water and urea. In our term, both kinds of aquaporins are included into a membrane:

$$T = \mathsf{w} \mid \ \ldots \ \mid \mathsf{w} \mid \mathsf{u} \mid \ \ldots \ \mid \mathsf{u} \mid \ (\mathsf{AW})^{\circlearrowleft} \rfloor (\epsilon) \mid \ (\mathsf{AWU})^{\circlearrowleft} \rfloor (\epsilon)$$

The type environment is the following:

$$\Lambda = \quad \{\mathsf{AW} : \texttt{Porin}, \mathsf{AWU} : \texttt{Porin}, \mathsf{w} : \texttt{Molecule}, \mathsf{u} : \texttt{Molecule}\}$$

and the class table $CT$ contains the `Porin` and `Molecule` classes. The triple is $(CT, \Lambda, P)$, where $P$ is composed by the term $T$ and the rule set containing the following method invocations:

$$\begin{array}{ccc} \mathsf{AW}.in(\mathsf{w}) & \mathsf{AW}.out(\mathsf{w}) & \mathsf{AWU}.in(\mathsf{w}) \\ \mathsf{AWU}.out(\mathsf{w}) & \mathsf{AWU}.in(\mathsf{u}) & \mathsf{AWU}.out(\mathsf{u}) \end{array}$$

After the evaluation of the triple, the CLS model is composed by the term $T$ and the rewrite rules

$$\begin{array}{rr} \mathsf{w} \mid (\mathsf{AW} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \to (\mathsf{AW} \cdot \widetilde{x})^{\circlearrowleft} \rfloor (\mathsf{w} \mid X) & (\mathsf{AW} \cdot \widetilde{x})^{\circlearrowleft} \rfloor (\mathsf{w} \mid X) \to \mathsf{w} \mid (\mathsf{AW} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \\ \mathsf{w} \mid (\mathsf{AWU} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \to (\mathsf{AWU} \cdot \widetilde{x})^{\circlearrowleft} \rfloor (\mathsf{w} \mid X) & (\mathsf{AWU} \cdot \widetilde{x})^{\circlearrowleft} \rfloor (\mathsf{w} \mid X) \to \mathsf{w} \mid (\mathsf{AWU} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \\ \mathsf{u} \mid (\mathsf{AWU} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \to (\mathsf{AWU} \cdot \widetilde{x})^{\circlearrowleft} \rfloor (\mathsf{u} \mid X) & (\mathsf{AWU} \cdot \widetilde{x})^{\circlearrowleft} \rfloor (\mathsf{u} \mid X) \to \mathsf{u} \mid (\mathsf{AWU} \cdot \widetilde{x})^{\circlearrowleft} \rfloor X \end{array}$$

### P systems

A P system [53] is a n-tuple $\Pi = (V, \mu, M_1, \ldots, M_n, (R_1, \rho_1), \ldots, (R_n, \rho_n), i_0)$, where

- $V$: alphabet;
- $\mu$: membrane structure of degree $n$, with the membrane and the regions labeled in a one-to-one manner with elements in a given set $L$;
- $M_i$: multisets of symbols (or strings) in $V$, denoting the symbols contained in the membrane $i$;
- $R_i$: finite sets of reduction rules (called *evolution rules*) $x \to y$ contained in the membrane $i$ and such that $x \in V^*$ and $y = y'$ or $y = y'\delta$, where $y' \in (V \times \{here, out\})^* \cup (V \times \{in_j \mid j \in L\})^*$;

- $\rho_i$: partial order relations over $R_i$;
- $i_0$: a label in $L$ which specifies the output membrane. If empty, then the output region is the environment (the space containing the outermost membrane).

Consider an evolution rule $x \rightarrow y$ in the set $R_i$: if the symbols in $x$ appear in $M_i$, then these symbols are replaced by the symbols in $y$ according to the rule. If a symbol $a$ appears in $y$ in a pair $(a, here)$, then it will remain in $M_i$. If a symbol $a$ appears in $y$ in a pair $(a, out)$, then it becomes a symbol of the membrane immediately outside the membrane $i$, according to the membrane structure $\mu$. If a symbol $a$ appears in $y$ in a pair $(a, in_j)$, and the membrane $j$ is contained in the membrane $i$ according to the membrane structure $\mu$, then it becomes a symbol of the membrane $j$. If $y = y'\delta$, then the membrane $i$ and the evolution rules in $R_i$ disappear, and all the symbols in $M_i$ are added to the symbols of the membrane immediately outside the membrane $i$. Evolution rules are applied following the priority in $\rho_i$, and in a non-deterministic way in case of same priority. In a single evolution step, all symbols in all membranes evolve in parallel, and every applicable evolution rule is applied as many times as possible (*maximal parallelism*).

According to the definitions of evolution rules, the rule syntax becomes

$$R ::= x \rightarrow y$$

Using classes and methods, each set $R_i$ becomes a set of method invocations, $R_i = \overline{I_i}$.

Note that in P systems there are two kinds of symbols which may be involved in an evolution rule: the biological entities (contained in $V$), and the labels of membranes (contained in $L$). Since they are different entities, we must design a distinct class for each of them. As a solution, we construct the class `BioObject` for biological entities, and `Label` for labels, both extending `Object`.

```
class BioObject extends Object{}
class Label extends Object{}
```

Every biological entities must extend `BioObject` or one of its subclasses: for this reason, the new definition of the class `Molecule` is

```
class Molecule extends BioObject{}
```

A class modeling the porin behavior with P-system evolution rules is the following:

```
class Porin extends BioObject
{
  in(Molecule S, Label J)
    S → S(in_J)

  out(Molecule S)
    S → S(out)
}
```

In our model, the aquaporin that transports only water ($\mathsf{w}$) is contained into the membrane labeled by 1, and the other one, that transports both urea ($\mathsf{u}$) and water, is contained into the membrane labeled by 2. The type environment is the following:

$$\Lambda = \{\mathsf{A} : \texttt{Porin}, \mathsf{w} : \texttt{Molecule}, \mathsf{u} : \texttt{Molecule}, 0 : \texttt{Label}, 1 : \texttt{Label}, 2 : \texttt{Label}\}$$

and the class table $CT$ contains the `Porin`, `Molecule` and `BioObject` classes. The triple is $(CT, \Lambda, \Pi)$, where $\Pi$ is the following[1]:

$$\Pi = (\{\mathsf{u}, \mathsf{w}, \mathsf{A}\}, [[]_2[]_3]_1, \{\mathsf{u}, \ldots \mathsf{u}, \mathsf{w}, \ldots, \mathsf{w}\}, \emptyset, \emptyset, (\mathsf{A}.in(\mathsf{w}, 1), \mathsf{A}.in(\mathsf{w}, 2), \mathsf{A}.in(\mathsf{u}, 2)), (\mathsf{A}.out(\mathsf{w})), (\mathsf{A}.out(\mathsf{w}), \mathsf{A}.out(\mathsf{u})), 1)$$

After the evaluation of the method invocations, we obtain the P system

$$\Pi' = (\{\mathsf{u}, \mathsf{w}, \mathsf{A}\}, [[]_2[]_3]_1, \{\mathsf{u}, \ldots \mathsf{u}, \mathsf{w}, \ldots, \mathsf{w}\}, \emptyset, \emptyset, (\mathsf{w} \to \mathsf{w}(in_1), \mathsf{w} \to \mathsf{w}(in_2), \mathsf{u} \to \mathsf{u}(in_2)), (\mathsf{w} \to \mathsf{w}(out)), (\mathsf{w} \to \mathsf{w}(out), \mathsf{u} \to \mathsf{u}(out)), 1)$$

## 7.5 Conclusions

As seen in Section 1.4, several formalisms implements modules for managing the complexity of biological processes. Using classes, our calculus can organize biological functionalities in boxes, like for modules, but moreover it can exploit the features of Object-Oriented programming, such as inheritance and subtyping. On the other side, the rules in a class are not visible from outside, then the resolution of the errors becomes more difficult. Respect

[1]For the sake of clarity, we assume that the evolution rules have the same priority, omitting the partial order relation $\rho$ over them

to other solutions seen in Section 1.4, our calculus does not specify a meta-language: this choice allows to use the calculus with different term-rewriting formalisms, but it pays off in terms of expressiveness, because we cannot exploit the expressive power of a particular syntax.

The modularity of our classes allows to design libraries for generic biological processes, which can be instantiated and re-used repeatedly in different contexts with different arguments, and even in different models, ensuring that their reduction rules are consistent with the biological ontology defined in them. These libraries could be designed and refined by experts, and then made available to all modelers, thereby creating a scientific common for model building. The same library could also be adapted from a formalism to another, rewriting the reduction rules and with small alteration to the hierarchy, if needed. That modularity allows System Biology to evolve in a decentralized manner: any user can develop novel abstractions of any biological functionality in a formalism, and contribute these back to the community, that can adapt these classes to another formalism.

Note that the calculus proposed in this Chapter implements only very basic features of the Object-Oriented paradigm. In our opinion, these features are the most common and useful in biological modeling, but increasing the complexity of the modeled systems the need of new features could emerge. For example, sometimes molecules have different roles depending on the context: our calculus cannot deal with this behavior, because each value is associated to exactly one type. For this reason, a possible development is surely the study and implementation of other basic and high-level constructs of Imperative and Object-Oriented paradigms, such as data structures, multiple inheritance or parametric polymorphism (also known as generics).

Finally, in our calculus the modeler decide which reduction rules to include in a model, but in this way a raw modeler could forget some important rule. A possible evolution is to infer the reduction rules directly from the composition of the model, according to the association between classes and values defined in the type environment. For example, if the term of the model contains a porin, then the system may infer the proper reduction rules to include, in this case the ones modeling the passage of elements through membranes. Moreover, in this way the reduction rules in a model could become dynamic: they could evolve following the evolution of the model, in a correct (from a biological point of view) way, without any external intervention. For example, if, during the evolution of the model, a lactase is created in the term, then the proper reduction rules, in this case the ones modeling hydrolysis, could be added.

# Chapter 8

# Conclusions

Biological molecules cannot survive if left alone, they usually live and grow in symbiosis with other molecules: the interaction between different biological actors is the key for surviving. And the key for interaction is proximity: only close molecules may interact. When we study small systems, we can suppose all the involved actors are so close each other that everyone can interact with all the other ones, but this assumption is no more true in more realistic systems. Biological models for complex systems must take into account the relative positions of their molecules, so we need formalisms able to express somehow the concept of space. As seen in Section 1.3, several spatial formalisms have been proposed, with different solutions. Here, we adopt a formalism based on membranes, the Calculus of Looping Sequences: among the other approaches, a formalism dealing with membranes is more flexible and less computationally heavy. In fact, into a membrane we do not need to know the exact position of each molecule, because we can assume they are close enough to interact each other, and according to the model we can decide the size of this environment. As proposed in Section 2.4.5, this schema is also highly scalable, even if it needs a supplementary framework to be readable by an human user. As proposed in Chapter 3, it is better parallelizable, because different membranes can evolve in parallel, if they do not interact with each other. Moreover, it allows to define rules having scope only into a membrane, and other rules having global validity.

While the implementation of spatial framework in biological models has intuitive motivations, the reason why types may improve biological formalisms looks less obvious, on a first sight: as a proof, few of the ones presented in Chapter 1 implement a type discipline. After having read this essay the motivation is clearer: a type discipline allows to transfer the complexity of biological properties from rules to types. Types are ideal for expressing general constraints derived from biological behaviors, leaving to rules the evolution of the specific system. The role of types in biological formalisms is the same used in (typed) programming languages: their syntax is quite simple, in order to be easy to understand and to use. In addition, the coder knows the limitations imposed by types, even if their physical implementation is completely hidden. To express the same conditions via syntax would make this one very complex. Moreover, modifications to the logic of types usually does not affect syntax. In order to study the utility of a type system, we need a formalism with few syntactic constraints, and huge expressivity. Among membrane systems, the Calculus of Looping Sequences is the ideal candidate: as showed in Section 2.2.1, it allows to describe several biomolecular events, and to write generic rules, by means of variables.

Once settled the main formalism, we need to choose how to implement a type system. It can have place in two circumstances, either at compile time or at runtime, i.e. limiting either the rules that can be written by modelers, or the ones used by the system in the current evolution step. Both approaches are implemented here: the first one in Chapter 7, while the second one in Chapters 3, 4, 5, 6. The first approach is quite obvious: syntactic constraints are not sufficient to ensure the biological correctness of rules, so types add additional constraints, more related to biological behaviors. For example, in the Calculus in Chapter 7 the modeler can define a hierarchy of types containing a prototype of rules, then associate each molecule with a type, and finally add the rules to the model. Rules cannot be arbitrarily defined, because they depend on the types of their involved molecules. The Calculus is inspired by the general implementation of a type system: as this one is implemented in different programming languages, in the same way the Calculus can be applied to several formalisms. While this approach aims to ensure the biological correctness of existing rules, the second one aims to add another goal: to decrease the quantity of rules of the models. While a model grows in complexity, the number of its rules grows, too. This growth becomes a problem for human users, that have to write (and check) them one by one. The solution is given by the second approach, that allows to define a limited number of generic rules, that will be applied to the specific case in the current step by the type system. The improvement derived by this idea is visible in the example in Section 4.4: without types, for managing blood transfusion we would write a rule for each compatible transfusion. The same goes for the hemoglobin variants and cell homeostasis in Section 5.4. As a drawback, since the typing constraints are hidden, their presence must be known a priori by modelers.

However, sometimes biological constraints can be broken. Some of them could be seen as practical suggestions, that if not followed could drive to undesirable behaviors, such as a disease or even the death of the system. Thinking on the examples in Sections 4.4 and 5.4, one could accidentally transfuse incompatible blood types, and the balance between cell death and division can be broken, leading to death or tumors. On the contrary, the typed semantics in Chapters 4 and 5 completely exclude this kind of situations. According to this idea, we could modify their typed semantics, allowing transitions which lead to untypable terms, but signaling that some undesired state has been reached. In the first case, we could modify the transitions driven by $\Delta$-safe rules behavior and $\Delta$-$(\mathtt{P},\mathtt{R})$-safe rules behavior, with the following two rules that raise an error when some undesired reduction is performed:

$$\frac{\begin{array}{c} P_1 \mapsto P_2 \in \mathcal{R} \text{ is not a } \Delta\text{-}(\mathtt{P},\mathtt{R})\text{-safe rule} \qquad P_1\sigma \not\equiv \epsilon \\ \sigma \in \Sigma_\Delta \qquad E \in \mathcal{E} \end{array}}{E[P_1\sigma] \xrightarrow{typabilityError} E[P_2\sigma]}$$

$$\frac{\begin{array}{c} P_1 \mapsto P_2 \in \mathcal{R} \text{ is a } \Delta\text{-}(\mathtt{P},\mathtt{R})\text{-safe rule} \qquad P_1\sigma \not\equiv \epsilon \\ \sigma \in \Sigma_\Delta \qquad E \in \mathcal{E} \qquad (\mathtt{P},\mathtt{R}) \text{ is not OK for } E \end{array}}{E[P_1\sigma] \xrightarrow{contextError} E[P_2\sigma]}$$

and modifying the algorithm in Section 4.3, raising an error in point 3b. In the second case, the rule to add could be the following one:

$$\frac{\begin{array}{c} \Re = P_1 \mapsto P_2 \in \mathcal{R} \text{ is a } \Delta\text{-}(\mathtt{P},\mathtt{L},\mathtt{M})\text{-safe rule} \qquad P_1\sigma \not\equiv \epsilon \\ \sigma \in \Sigma_\Delta \qquad E \in \mathcal{E} \qquad (\mathtt{P},\mathtt{L},\mathtt{M}) \text{ is not } OK \text{ for } E \end{array}}{E[P_1\sigma] \xrightarrow{Error} E[P_2\sigma]}$$

In both cases, the modeler is advised that some unwanted behavior is happening in the system. In this way, the modeler can check if, starting from the initial term and using the given rules, we can reach a non-typable term, i.e. a term that breaks some biological property, or either that some unwanted behavior is happening in the system and readjust it to avoid the undesired situations.

The application of Type Theory skills to biological formalisms is a new field of research, in the young research field of System Biology. Nowadays we are performing the first steps, and their results are really promising: increase of biological accuracy, and decrease of complexity of models, at the price of a more complex type engine. We can find several possible future development in this field. Here we have implemented some biological properties by means of types, but other ones can be studied. Types can be applied to rules instead of molecules, as in Chapter 3, or to calculate other information useful for the model, such as the rate of a reduction, as done in Chapter 6. New formalisms exploiting the expressivity freedom gained thanks to types, and so less related to a set of predetermined biological properties, can be designed. A lot of work still need to be done, but in the future the presence of type system in biological models will increase, and types will become essential in formalisms for System Biology, as they are in programming languages.

# Bibliography

[1] Biocham. http://contraintes.inria.fr/BIOCHAM/.

[2] Marco Aldinucci, Mario Coppo, Ferruccio Damiani, Maurizio Drocco, Elio Giovannetti, Elena Grassi, Eva Sciacca, Salvatore Spinella, and Angelo Troina. *CWC Simulator*. Dipartimento di Informatica, Università di Torino, 2010. http://cwcsimulator.sourceforge.net/.

[3] Marco Aldinucci and Massimo Torquati. *FastFlow website*. FastFlow, October 2009. http://mc-fastflow.sourceforge.net/.

[4] Rajeev Alur, Calin Belta, Vijay Kumar, and Max Mintz. Hybrid Modeling and Simulation of Biomolecular Networks. In *Hybrid Systems: Computation and Control*, volume 2034 of *LNCS*, pages 19–32. Springer, 2001.

[5] Bogdan Aman and Gabriel Ciobanu. Simple, Enhanced and Mutual Mobile Membranes. *Transictions on Computational System Biology*, 11:26–44, 2009.

[6] Bogdan Aman and Gabriel Ciobanu. *Mobility in Process Calculi and Natural Computing*. Natural Computing Series. Springer, 2011.

[7] Bogdan Aman and Gabriel Ciobanu. Mutual Mobile Membranes with Objects on Surface. *Natural Computing*, 10(2):777–793, 2011.

[8] Bogdan Aman, Mariangiola Dezani-Ciancaglini, and Angelo Troina. Type Disciplines for Analysing Biologically Relevant Properties. In *Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'08)*, volume 227 of *ENTCS*, pages 97–111. Elsevier, 2009.

[9] Roberto Barbuti, Andrea Maggiolo-Schettini, and Paolo Milazzo. Extending the Calculus of Looping Sequences to Model Protein Interaction at the Domain Level. In *International Symposium on Bioinformatics Research and Applications (ISBRA'07)*, volume 4463 of *LNBI*, pages 638–649. Springer, 2006.

[10] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, Giovanni Pardini, and Luca Tesei. Spatial P Systems. *Natural Computing*, 10(1):3–16, 2011.

[11] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, Paolo Tiberi, and Angelo Troina. Stochastic Calculus of Looping Sequences for the Modelling and Simulation of Cellular Pathways. *Transactions on Computational Systems Biology*, IX:86–113, 2008.

[12] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. A Calculus of Looping Sequences for Modelling Microbiological Systems. *Fundamenta Informaticæ*, 72(1–3):21–35, 2006.

[13] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. Bisimulation Congruences in the Calculus of Looping Sequences. In *International Colloquium on Theoretical Aspects of Computing (ICTAC'06)*, volume 4281 of *LNCS*, pages 93–107. Springer, 2006.

[14] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. The Calculus of Looping Sequences for Modeling Biological Membranes. In *Workshop on Membrane Computing*, volume 4860 of *LNCS*, pages 54–76. Springer, 2007.

[15] Roberto Barbuti, Andrea Maggiolo-Schettini, Paolo Milazzo, and Angelo Troina. Bisimulations in Calculi Modelling Membranes. *Formal Aspects of Computing*, 20(4-5):351–377, 2008.

[16] Daniela Besozzi and Gabriel Ciobanu. A P System Description of the Sodium-Potassium Pump. In *Workshop on Membrane Computing (WMC'04)*, volume 3365 of *LNCS*, pages 210–223. Springer, 2005.

[17] Livio Bioglio. Typeed Reductions of CLS. In *Italian Conference on Theoretical Computer Science (ITCTS'09)*, 2009.

[18] Livio Bioglio. Enumerated Type Semantics for the Calculus of Looping Sequences. *RAIRO - Theoretical Informatics and Applications*, 45(01):35 –58, 2011.

[19] Livio Bioglio, Cristina Calcagno, Mario Coppo, Ferruccio Damiani, Eva Sciacca, Salvatore Spinella, and Angelo Troina. A Spatial Calculus of Wrapped Compartments. In *5th International Meeting on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'11)*, pages 25–39, 2011.

[20] Livio Bioglio, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Angelo Troina. A Calculus of Looping Sequences with Local Rules. In *7th Workshop on Developments in Computational Models (DCM'11)*, volume 88 of *EPTCS*, pages 43–58, 2011.

[21] Livio Bioglio, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Angelo Troina. Type Directed Semantics for the Calculus of Looping Sequences. *International Journal of Software and Informatics*, 2012. to appear.

[22] Livio Bioglio, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Angelo Troina. Typed Stochastic Semantics for the Calculus of Looping Sequences. *Theoretical Computer Science*, 431:165 –180, 2012.

[23] Luca Bortolussi and Maria G. Vigliotti. CoBiC: Context-dependent Bioambient Calculus. In *Quantitative Aspects of Programming Languages (QAPL'09)*, volume 253 of *ENTCS*, pages 187–201. Elsevier, 2009.

[24] Linda Brodo, Pierpaolo Degano, and Corrado Priami. A Stochastic Semantics for BioAmbients. In *Parallel Computing Technologies (PACT'07)*, volume 4671 of *LNCS*, pages 22–34, 2007.

[25] Nadia Busi. Using Well-structured Transition Systems to Decide Divergence for Catalytic P Systems. *Theoretical Computer Science*, 372:125–135, 2007.

[26] Federico Buti, Diletta Cacciagrano, Flavio Corradini, Emanuela Merelli, and Luca Tesei. BioShape: a Spatial Shape-Based Scale-Independent Simulation Environment for Biological Systems. *Procedia CS*, 1(1):827–835, 2010.

[27] Luca Cardelli. Brane Calculi. Interactions of Biological Membranes. In *Computational Methods in Systems Biology (CMSB'04)*, volume 3082 of *LNCS*, pages 257–280. Springer, 2005.

[28] Luca Cardelli and Philippa Gardner. Processes in Space. In *Proc. of the 6th international conference on Computability in Europe*, CiE'10, pages 78–87. Springer-Verlag, 2010.

[29] Paolo Cazzaniga, Dario Pescini, Francisco J. Romero-campero, Daniela Besozzi, and Giancarlo Mauri. Stochastic Approaches in P Systems for Simulating Biological Systems. In *Proceedings of the Fourth Brainstorming Week on Membrane Computing*, pages 145–164, 2006.

[30] Gabriel Ciobanu, Linqiang Pan, Gheorghe Paun, and Mario J. Pérez-Jiménez. P Systems with Minimal Parallelism. *Theoretical Computer Science*, 378(1):117–130, 2007.

[31] Gabriel Ciobanu, Mario J. Pérez-Jiménez, and Gheorghe Paun, editors. *Applications of Membrane Computing*. Natural Computing Series. Springer, 2006.

[32] Mario Coppo, Ferruccio Damiani, Maurizio Drocco, Elena Grassi, Mike Guether, and Angelo Troina. Modelling Ammonium Transporters in Arbuscular Mycorrhiza Symbiosis. *Transactions on Computational Systems Biology*, XIII:85–109, 2011.

[33] Mario Coppo, Ferruccio Damiani, Maurizio Drocco, Elena Grassi, Eva Sciacca, Salvatore Spinella, and Angelo Troina. Hybrid Calculus of Wrapped Compartments. In *Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'10)*, volume 40, pages 102–120. EPTCS, 2010.

[34] Mario Coppo, Ferruccio Damiani, Maurizio Drocco, Elena Grassi, and Angelo Troina. Stochastic Calculus of Wrapped Compartments. In *8th Workshop on Quantitative Aspects of Programming Languages (QAPL'10)*, volume 28, pages 82–98. EPTCS, 2010.

[35] Vincent Danos, Jérôme Feret, Walter Fontana, Russ Harmer, and Jean Krivine. Rule-Based Modelling and Model Perturbation. *Transactions on Computational Systems Biology XI*, 11:116–137, 2009.

[36] Vincent Danos, Jérôme Feret, Walter Fontana, and Jean Krivine. Scalable Modelling of Biological Pathways. In *ASIAN Symposium on Programming Languages and Systems (APLAS'07)*, volume 4807, pages 139–157, 2007.

[37] Vincent Danos and Cosimo Laneve. Core Formal Molecular Biology. In *European Symposium on Programming (ESOP'03)*, volume 2618 of *LNCS*, pages 302–318, 2003.

[38] Vincent Danos and Cosimo Laneve. Formal Molecular Biology. *Theoretical Computer Science*, 325:69–110, 2004.

[39] Mariangiola Dezani-Ciancaglini, Paola Giannini, and Angelo Troina. A Type System for a Stochastic CLS. In *Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC'09)*, volume 11, pages 91–105. EPTCS, 2009.

[40] Mariangiola Dezani-Ciancaglini, Paola Giannini, and Angelo Troina. A Type System for Required/Excluded Elements in CLS. In *Developments in Computational Models (DCM'09)*, volume 9 of *EPTCS*, pages 38–48, 2009.

[41] Daniell Díaz-Pernil, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez, and Agustin Riscos-Núñez. A P-Lingua Programming Environment for Membrane Computing. In *Workshop on Membrane Computing*, volume 5391 of *LNCS*, pages 187–203, 2008.

[42] Pavel Dolezal, Vladimir Likic, Jan Tachezy, and Trevor Lithgow. Evolution of the Molecular Machines for Protein Import into Mitochondria. *Science*, 313(5785):314–318, 2006.

[43] François Fages and Sylvain Soliman. Abstract Interpretation and Types for Systems Biology. *Theoretical Computer Science*, 403(1):52–70, 2008.

[44] Daniel T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions,.

[45] Jochen S. Hub and Bert L. de Groot. Mechanism of Selectivity in Aquaporins and Aquaglyceroporins. In *Proceedings of National Academy of Sciences of the USA*, volume 105, pages 1198–1203, 2008.

[46] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: a Minimal Core Calculus for Java and GJ. *ACM Transactions on Programming Languages and System*, 23:396–450, 2001.

[47] Jean Krivine, Robin Milner, and Angelo Troina. Stochastic Bigraphs. In *Mathematical Foundations of Programming Semantic (MFPS'08)*, volume 218 of *ENTCS*, pages 73–96. Elsevier, 2008.

[48] Aneil Mallavarapu, Matthew Thomson, Benjamin Ullian, and Jeremy Gunawardena. little b. http://www.littleb.org.

[49] Hiroshi Matsuno, Atsushi Doi, Masao Nagasaki, and Satoru Miyano. Hybrid Petri Net Representation of Gene Regulatory Networks. In *Pacific Symposium on Biocomputing (PSB'00)*, pages 341–352. World Scientific Press, 2000.

[50] Paolo Milazzo. *Qualitative and Quantitative Formal Modeling of Biological Systems.* PhD thesis, University of Pisa, 2007.

[51] Nicolas Oury and Gordon Plotkin. Multi-Level Modelling via Stochastic Multi-Level Multiset Rewriting. *Mathematical Structures in Computer Science.*, 2012. To appear.

[52] Terence Parr et al. *ANTLR website.* http://www.antlr.org/.

[53] Gheorghe Păun. *Membrane Computing. An Introduction.* Springer, 2002.

[54] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *The Oxford Handbook of Membrane Computing.* Oxford University Press, 2010.

[55] Michael Pedersen and Gordon D. Plotkin. A Language for Biochemical Systems: Design and Formal Specification. *Transactions on Computational Systems Biology*, 12:77–145, 2010.

[56] Dario Pescini, Daniela Besozzi, Giancarlo Mauri, and Claudio Zandron. Dynamical Probabilistic P Systems. *International Journal of Foundations of Computer Science*, 17(1):183–204, 2006.

[57] Corrado Priami and Paola Quaglia. Beta-binders for Biological Interactions. In *Computational Methods in Systems Biology (CMSB'04)*, volume 3082 of *LNCS*, pages 20–33, 2005.

[58] Corrado Priami, Aviv Regev, William Silverman, and Ehud Shapiro. Application of Stochastic Name-passing Calculus to Representation and Simulation of Molecular Processes. *Infomation Processing Letters*, 80(1):25–31, 2001.

[59] Aviv Regev, Ekaterina M. Panina, William Silverman, Luca Cardelli, and Ehud Shapiro. BioAmbients: An Abstraction for Biological Compartments. *Theoretical Computer Science*, 325:141–167, 2004.

[60] Aviv Regev and Ehud Shapiro. Cells as Computation. *Nature*, 419(6905):343, September 2002.

[61] Aviv Regev and Ehud Shapiro. The $\pi$-calculus as an Abstraction for Biomolecular Systems. *Modelling in Molecular Biology*, pages 219–266, 2004.

[62] Nitzan Rosenfeld, Jonathan W. Young, Uri Alon, Peter S. Swain, and Michael B. Elowitz. Gene Regulation at the Single-cell Level. *Science*, 307(5717):1962–1965, 2007.

[63] Antoine Spicher, Olivier Michel, and Jean-Louis Giavitto. Interaction-Based Simulations for Integrative Spatial Systems Biology. In *Understanding the Dynamics of Biological Systems*, pages 195–231. 2011.

[64] Apostolos Syropoulos. Mathematics of Multisets. In *Multiset Processing*, volume 2235 of *LNCS*, pages 347–358, 2001.

[65] Lincoln Taiz and Eduardo Zeiger. *Plant Physiology, Fourth Edition.* Sinauer Associated Inc., 2006.

[66] Ruiqi Wang, Chunguang Li, Luonan Chen, and Kazuyuki Aihara. Modeling and Analyzing Biological Oscillations in Molecular Networks. *Proceedings of The IEEE – PIEEE*, 96(8):1361–1385, 2008.

[67] J. B. Wells. The Essence of Principal Typings. In *Intenational Colloquium on Automata, Languages and Programming (ICALP'02)*, volume 2380 of *LNCS*, pages 913–925, 2002.

[68] Darren Wilkinson. *Stochastic Modelling for Systems Biology.* Chapman & Hall/CRC, 2006.

[69] Patrick Wong, Stephanie Gladney, and Jay D. Keasling. Mathematical Model of the Lac Operon: Inducer Exclusion, Catabolite Repression, and Diauxic Growth on Glucose and Lactose. *Biotechnology Progress*, 13:132–143, 1997.