Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis

Qualitative and Quantitative Formal Modeling of Biological Systems

Paolo Milazzo

Supervisor
Prof. Roberto Barbuti

Supervisor
Prof. Andrea Maggiolo–Schettini

April 30, 2007

Abstract

Cell Biology, the study of the morphological and functional organization of cells, is now an established field in biochemical research. Computer Science can help the research in Cell Biology in several ways. For instance, it can provide biologists with models and formalisms able to describe and analyze complex systems such as cells. In the last few years many formalisms, originally developed by computer scientists to model systems of interacting components, have been applied to Biology. Among these, there are Petri Nets, Hybrid Systems, and the π-calculus. Moreover, formalisms such as P Systems, originally developed to study new computational paradigms inspired by Biology, have recently found application to the description of biological phenomena. Finally, some new formalisms have been proposed to describe biomolecular and membrane interactions.

The first advantage of using formal models to describe biological systems is that they avoid ambiguities. In fact, ambiguity is often a problem of the notations used by biologists. Moreover, the formal modeling of biological systems allows the development of simulators, which can be used to understand how the described system behaves in normal conditions, and how it reacts to changes in the environment and to alterations of some of its components. Furthermore, formal models allow the verification of properties of the described systems, by means of tools (such as model checkers) which are well established and widely used in other application fields of Computer Science, but unknown to biologists.

In this thesis we develop a formalism for the description of biological systems, called Calculus of Looping Sequences (CLS), based on term rewriting and including some typical features of process calculi for concurrency. What we want to achieve is a formalism that allows describing proteins, DNA fragments, membranes and other macromolecules, without ignoring the physical structure of these elements, and by keeping the syntax and the semantics of the formalism as simple as possible.

CLS terms are constructed from an alphabet of basic symbols (representing simple molecules) and include operators for the creation of sequences (representing proteins and DNA fragments), of closed sequences which may contain something (representing membranes), and of multisets of all these elements (representing juxtaposition). A CLS term describes the structure of the system under study, and its evolution is given by the application of rewrite rules describing the events that may occur in the system, and how the system changes after the occurrence of one of these events. We equip CLS with an operational semantics describing the possible evolutions of the system by means of application of given rewrite rules, and we show that other formalisms for the description of membranes can be encoded into CLS in a sound and complete way.

We propose bisimilarity as a tool to verify properties of the described systems. Bisimilarity is widely accepted as the finest extensional behavioral equivalence one may want to impose on systems. It may be used to verify a property of a system by assessing the

bisimilarity of the considered system with a system one knows to enjoy that property. To define bisimilarity of systems, these must have semantics based on labeled transition relations capturing potential external interactions between systems and their environment. A labeled transition semantics for CLS is derived from rewrite rules by using as labels contexts that would allow rules to be applied. We define bisimulation relations upon this semantics, and we show them to be congruences with respect to the operators on terms.

In order to model quantitative aspects of biological systems, such as the the frequency of a biological event, we develop a stochastic extension of CLS, called Stochastic CLS. Rates are associated with rewrite rules in order to model the speeds of the described activities. Therefore, transitions derived in Stochastic CLS are driven by exponential distributions, whose rates are obtained from the rates of the applied rewrite rules and characterize the stochastic behavior of the transitions. The choice of the next rule to be applied and of the time of its application is based on the classical Gillespie's algorithm for simulation of chemical reactions.

Stochastic CLS can be used as a formal foundation for a stochastic simulator, but also to build models to be given as an input to model checking tools. In fact, the transition system obtained by the semantics of Stochastic CLS can be easily transformed into a Continuous Time Markov Chain (CTMC). If the set of states of the CTMC is finite (namely, if the set of reachable CLS terms is finite) a standard probabilistic model checker (such as PRISM) can be used to verify properties of the described system.

Finally, we propose a translation of Kohn Molecular Interaction Maps (MIMs), a compact graphical notation for biomolecular systems, into Stochastic CLS. By means of our translation, a simulator of systems described with Stochastic CLS can be used to simulate also systems described by using MIMs.

Acknowledgments

I would have unable to complete this thesis without the support and guidance of my family, friends and colleagues. Please accept my thank for your assistance and patience.

I am particularly grateful to my supervisors Roberto Barbuti and Andrea Maggiolo Schettini who guided me through the technical hurdles of my work and helped shaping my approach to research.

Most of the material in this thesis is the result of joint work with Angelo Troina and Paolo Tiberi. Working with them was at the same time fruitful and enjoyable.

I would also thank Gheorghe Păun and Vincent Danos, the referees of this thesis, and Roberto Grossi and Umberto Mura, the thesis committee members, for their precious comments and suggestions.

Finally, I am grateful to my fiancée Lia for having continuously encouraged me, and my family for its continuous support.

Contents

Chapter 1

Introduction

1.1 Motivation

Biochemistry, often conveniently described as the study of the chemistry of life, is a multi-faceted science that includes the study of all forms of life and that utilizes basic concepts derived from Biology, Chemistry, Physics and Mathematics to achieve its goals. Biochemical research, which arose in the last century with the isolation and chemical characterization of organic compounds occurring in nature, is today an integral component of most modern biological research.

Most biological phenomena of concern to biochemists occur within small, living cells. In addition to understanding the chemical structure and function of the biomolecules that can be found in cells, it is equally important to comprehend the organizational structure and function of the membrane–limited aqueous environments called cells. Attempts to do the latter are now more common than in previous decades. Where biochemical processes take place in a cell and how these systems function in a coordinated manner are vital aspects of life that cannot be ignored in a meaningful study of biochemistry. Cell biology, the study of the morphological and functional organization of cells, is now an established field in biochemical research.

Computer Science can help the research in cell biology in several ways. For instance, it can provide biologists with models and formalisms able to describe and analyze complex systems such as cells. In the last few years many formalisms originally developed by computer scientists to model systems of interacting components have been applied to Biology. Among these, there are Petri Nets [51], Hybrid Systems [2], and the π-calculus [20, 69]. Moreover, some new formalisms have been proposed to describe biomolecular and membrane interactions [3, 13, 16, 23, 63, 66]. Others, such as P Systems [58, 59, 60], have been proposed as new biologically inspired computational models and have been later applied to the description of biological systems.

The π–calculus and new calculi based on it [63, 66] have been particularly successful in the description of biological systems, as they allow describing systems in a compositional manner. Interactions of biological components are modeled as communications on channels whose names can be passed. Sharing names of private channels allows describing biological compartments. However, these calculi offer very low–level interaction primitives, and this causes models to become very large and difficult to be read. Calculi such as those proposed in [13, 16, 23] give a more abstract description of systems and offer special

biologically motivated operators. However, they are often specialized to the description of some particular kinds of phenomena such as membrane interactions or protein interactions. Finally, P Systems have a simple notation and are not specialized to the description of a particular class of systems, but they are still not completely general. For instance, it is possible to describe biological membranes and the movement of molecules across membranes, and there are some variants able to describe also more complex membrane activities. However, the formalism is not flexible enough to allow describing easily new activities observed on membranes without defining new extensions of it.

From this discussion we conclude that there is a need of a formalism having a simple notation, having the ability to describe biological systems at different levels of abstraction, having some notions of compositionality and being flexible enough to allow describing new kinds of phenomena as they are discovered, without being specialized to the description of a particular class of systems. The aim of this thesis is to study a new formalism which could represent a step towards the satisfaction of all these requirements.

Both the qualitative and the quantitative aspects of biological systems are interesting: the former are related to *state dependent properties*, such as reachability of states or existence of equilibria and stable states; the latter are related to *time and probability dependent properties*, like the time needed to reach a certain state and the probability of reaching a certain state in a given time or in any time. In this thesis we shall develop an extension of our formalism to take into account also quantitative aspects of biological systems.

1.2 Contributions

In this thesis we present a new calculus based on term rewriting and called Calculus of Looping Sequences (CLS). We describe several variants of CLS and we choose among them the one which is expressive enough to describe the biological systems of interest and having the simplest semantics. The terms of CLS are constructed by starting from basic constituent elements and composing them by means of operators of sequencing, looping, containment and parallel composition. Looping allows tying up the ends of a sequence, thus creating a circular sequence of the constituent elements. We assume that the elements of a circular sequence can rotate, and this motivates the terminology of looping sequence. A looping sequence can represent a membrane and the containment operator allows representing that some element is inside the membrane.

In order to show that CLS is suitable to describe biological systems and their evolutions we give some guidelines for the modeling of such systems in CLS, and we show some CLS models of real biological systems. Moreover, we show how other well–established formalsisms for the description of biological systems can be translated into CLS.

Bisimilarity is widely accepted as the finest extensional behavioral equivalence one may want to impose on systems. It may be used to verify a property of a system by assessing the bisimilarity of the considered system with a system one knows to enjoy that property. The notion of congruence is very important for a compositional account of behavioral equivalence. This is true, in particular, for complex systems such as biological ones.

To define bisimilarity of systems, these must have semantics based on labeled transition relations capturing potential external interactions between systems and their environment. A labeled transition semantics for CLS is derived from rewrite rules by using as labels

contexts in which rules can be applied, in the style of Sewell [72] and Leifer and Milner [49]. We define bisimilarity relations and we show them to be congruences with respect to the operators on terms.

Biologists usually describe quantitative aspects of a biological system by giving a set of differential equations. Each equation gives the transformation rate of one of the components of the described system. Hence, simulation of the system can be performed by using a computer tool for solving differential equations (as, for example, [26] and [52]).

An alternative approach to the simulation of biological systems is the use of stochastic simulators. This kind of tools are usually based on simulation algorithms proved to be correct with respect to the kinetic theory of chemical reactions. The most used and well–established of such algorithms is the one introduced by Gillespie in [31]. Other examples are [6] and the one used in the StochSim simulator [73].

In his paper, Gillespie shows that the quantity of time spent between the occurrence of two chemical reactions is exponentially distributed, with the sum of the kinetic rates of the possible reactions as the parameter of the exponential distribution. This allows him to give a very simple and exact stochastic algorithm for simulating chemical reactions.

Exponential distribution is a probability distribution for which some very useful properties hold. The most important one is the memoryless property, that allows forgetting the history of the simulation in the choice of the time that will be spent by the next reaction. These properties motivated the proliferation of a number of stochastic models with exponentially distributed variables. From the mathematical point of view, the most famous of such models are *Continuous Time Markov Chains (CTMCs)*, while, from the computer science point of view, most of these models fall into the category of *Stochastic Process Algebras* (as, for example, [33, 36, 62]).

Exponential distribution is the *trait–d'union* between simulation of biological systems and stochastic process algebras, and permitted the latter to be easily applied to the description of biological systems. In particular, the *Stochastic π–Calculus* [62] has been successfully applied to the (quantitative) modeling of biological systems, becoming at the moment one of the most used compositional formalisms [11, 45, 64] in the new field of *Systems Biology* [39, 40].

In order to model quantitative aspects of biological systems, we develop a stochastic extension of CLS. Rates are associated with rewrite rules in order to model the speeds of the described activities. Therefore, transitions derived in Stochastic CLS are driven by a rate which 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 [31].

The transition system obtained by the semantics of Stochastic CLS can be easily transformed into a Continuous Time Markov Chain (CTMC). If the set of states of the CTMC is finite (namely, if the set of reachable CLS terms is finite) a standard probabilistic model checker (such as PRISM [46]) can be used to verify properties of the described system.

Since the most used technique for studying biological systems is simulation, we have developed a simulator for Stochastic CLS. In order to show the expressiveness of our formalism, we model and simulate some real examples of biological systems.

1.3 Related Work

We briefly describe some notable examples of formalisms that have been used in the last few years for modeling biological systems. Some of them have been defined with the specific purpose of describing biochemical networks and activity of membranes inside cells. Moreover, some of them have been inspired by the π–calculus process algebra of Milner [55], which is a standard foundational language for concurrency theory.

One of the oldest formalisms are Lindenmayer systems (or L Systems) [65]. An L system is a formal grammar most famously used to model the growth processes of plant development.

In the tradition of automata and formal language theory, a more recent formalism are P Systems, introduced by Păun [58, 59, 60]. P Systems introduce the idea of membrane computing in the subject of natural computing. They represent a new computational paradigm which allow solving NP-complete problem in polynomial time (but in exponential space), they originated a very big mass of work and recently they have been also applied to the description of biological systems (see [74] for a complete list of references).

A pioneering formalism in the description of biological systems is the κ–calculus of Danos and Laneve [23]. It is a formal language for protein interactions, it is enriched with a very intuitive visual notation and it has been encoded into the π–calculus. The κ–calculus idealizes protein-protein interactions, essentially as a particular restricted kind of graph–rewriting operating on graphs with sites. A formal protein is a node with a fixed number of sites, and a complex (i.e. a bundle of proteins connected together by low energy bounds) is a connected graph built over such nodes, in which connections are established between sites. The κ–calculus has been recently extended to model also membranes [47].

An example of direct application of a model for concurrency to biochemical systems has been introduced by Regev and Shapiro in [69, 67]. Their idea is to describe metabolic pathways as π–calculus processes and in [64] they showed how the stochastic variant of the model, defined by Priami in [62], can be used to represents both qualitative and quantitative aspects of the systems described. Moreover, Regev, Panina, Silverman, Cardelli and Shapiro in [66] defined the BioAmbients calculus, a model inspired by both the π–calculus and the Mobile Ambients calculus [14], which can be used to describe biochemical systems with a notion of compartments (as, for instance, membranes). More details of membrane interactions have been considered by Cardelli in the definition of Brane Calculi [13], which are elegant formalisms for describing intricate biological processes involving membranes. Moreover, a refinement of Brane Calculi have been introduced by Danos and Pradalier in [24].

We conclude by mentioning some works by Harel [35][38], in which the challenging idea is introduced of modelling a full multi–cellular animal as a reactive system. The multi–cellular animal should be, specifically, the C. elegans nematode worm [12], which is complex, but well defined in terms of anatomy and genetics. Moreover, Harel proposes to use the languages of Statecharts [34] and Live Sequence Charts (LSC) [21], which are visual notations with a formal semantics commonly adopted in the specification of software projects. Harel applies the same formalisms also to cellular and multi–cellular systems related to the immune systems of living organisms in [37] and [27].

1.4 Structure of the Thesis

The thesis is structured as follows.

- In Chapter 2 we recall some background notions of Biology, probability theory and Computer Science that will be assumed in the rest of the thesis.

In Chapters 3, 4 and 5 we introduce qualitative models of biological systems and their relationships with other well–established formalisms.

- In Chapter 3 we present a family of calculi based on term–rewriting and called Calculi of Looping Sequences. The family consists of three formalisms: the first is Full–CLS, in which terms are constructed by using operators of sequencing, parallel composition, looping and containment without any syntactical constraint. The second calculus of the family is CLS, and it differs from Full–CLS in the presence of some syntactical constraints which make its semantics very simple, without loosing too much from the viewpoint of expressiveness. The third calculus is called LCLS, and it is an extension of CLS which can be used to model protein interaction at the domain level, as it allows creating links (bindings) between individual elements of different sequences, modeling different proteins. The increased expressiveness of LCLS causes the need of a more complex semantics able to preserve a notion of well–formedness in order to ensure that links are not established between more than two elements, and they are established only between elements in the same membrane–delimited compartment. For each of the three calculi we give an application to the modeling of a real biological phenomenon.

- In Chapter 4 we give some guidelines for the modeling of biological systems with CLS. We choose CLS among the formalisms of the family of Calculi of Looping Sequences as it is the best compromise between expressiveness and simplicity. In this chapter, we also consider another variant of CLS, called CLS+, in which a form of commutativity can be introduced on looping sequences, as it could allow modeling membranes in a more natural way. We show that CLS+ can be translated into CLS.

- In Chapter 5 we compare CLS with two of the formalisms most related with it, namely with Brane Calculi [13] and P Systems [58, 59, 60]. We show that both Brane Calculi and P Systems can be translated into CLS, in particular we show the encodings into CLS of the PEP calculus, the simplest of Brane Calculi, and of transition P Systems, the most common variant of P Systems. In the case of the PEP calculus we can give a formally defined sound and complete translation of PEP systems into CLS terms. By applying the CLS rewrite rules associated with the encoding it is possible to obtain a semantic model from the term obtained by the translation which is equivalent to the semantic model of the original PEP system. In the case of P Systems, instead, we face in particular the problem of translating their maximal parallelism into an interleaving (sequential) model as CLS is. This is the main problem to be faced (the translation of Sequential P Systems [22] would be quite easy) and in order to solve it, we define a simulation algorithm for P Systems and we show how it can be "implemented" into CLS. We do not provide the translations of CLS into Brane Calculi and P Systems as the complete absence of constraints in the definition of CLS rewrite rules would make the work practically intractable.

In Chapters 6 and 7 we propose bisimulations as formal tools for the verification of properties of biological systems

- In Chapter 6 we develop a labeled semantics and bisimulation relations for CLS. The labeled semantics is defined by using as labels the context in which the term would permit the application of a rewrite rule. The main results of these chapters are that the bisimilarity relations defined on CLS terms are congruences. Moreover, we give bisimulation relations on systems, namely we allow comparing terms which may evolve by means of application of rewrite rules from two different sets. In this case the bisimulation relations are not congruences, however, as we show in an example, they can be used to verify interesting properties of the described systems, such as causality relationships between events.

- In Chapter 7 we develop a labeled semantics and bisimulation relations for the simplest of Brane Calculi, namely for the PEP calculus. As far as we know, this has never been done for such a calculus. Consequently, we compare the bisimulations of the PEP calculus with those of CLS defined in Chapter 6 by using the encoding of the PEP calculus into CLS defined in Chapter 5.

In Chapters 8 and 9 we study an extension of CLS for describing quantitative aspects of biological systems.

- In Chapter 8 we develop a stochastic extension of CLS, called Stochastic CLS, suitable to describe quantitative aspects of biological systems such as the frequencies and the probabilities of events. The extension is obtained by allowing rate constants to be specified in rewrite rules of CLS, and by incorporating the stochastic framework of the Gillespie algorithm [31] in the semantics of the formalism. This is the standard way of extending a formalism to model quantitative aspects of biological systems, but, as we shall see, this is not a trivial exercise in the case of CLS. From the semantics of a Stochastic CLS model it is possible to derive a Continuous Time Markov Chain, and this allows simulating and analyzing the system. We have developed a prototype simulator for Stochastic CLS, and we show the result of simulation of a real example of biological system.

- In Chapter 9 we show how Kohn's Molecular Interaction Maps (MIMs) [1, 43] can be translated into Stochastic CLS in order to allow simulating them. MIMs are a graphical notations for the description of biological pathways which can be used to describe a wide variety of interactions between cellular entities. Unfortunately, MIMs have not a formal syntax and semantics, hence we will describe their translation into Stochastic CLS by showing relevant examples. The translation of MIMs into Stochastic CLS allows simulating systems described by using MIMs, and also allows using them as a graphical user interface for a simulator based on Stochastic CLS.

Finally, we give some conclusions and discuss further work in Chapter 10.

1.5 Published Material

Part of the material presented in this thesis has appeared in some publications or has been submitted for publication, in particular:

- The definitions of Full–CLS presented in Section 3.1 and of the encoding of the PEP Calculus into CLS presented in Section 5.1 have appeared in [7].
- The definition of CLS presented in Section 3.3, the labeled semantics and the bisimulation relations presented in Chapter 6 have appeared in [8]. Moreover, an extended version of [8] that includes also the labeled semantics and the bisimulation relations for Brane Calculi presented in Chapter 7 has been submitted for publication [9].
- The definition of LCLS presented in Section 3.6 has appeared in [4].
- The definition of Stochastic CLS has been submitted for publication [5].
- The chemical reactions describing the activity of the Sorbitol Dehydrogenase enzyme simulated in Section 8.1.4 have been studied before in [3, 6].
- The main results of our work on CLS will be published as an invited contribution in the proceedings of the 8th Workshop on Membrane Computing [10].

All the published material is presented in this thesis in revised and extended form.

Chapter 2

Background

2.1 Notions of Biochemistry and Cell Biology

There are two basic classifications of cell: *procaryotic* and *eucaryotic*. Traditionally, the distinguishing feature between the two types is that a eucaryotic cell possesses a membrane–enclosed nucleus and a procaryotic cell does not. Procaryotic cells are usually small and relatively simple, and they are considered representative of the first types of cell to arise in biological evolution. Procaryotes include, for instance, almost all bacteria. Eucaryotic cells, on the other hand, are generally larger and more complex, reflecting an advanced evolution, and include multicellular plants and animals.

In eucaryotic cells, different biological functions are segregated in discrete regions within the cell, often in membrane–limited structures. Subcellular structures which have distinct organizational features are called *organelles*. As an organelle, for example, the *nucleus* contains chromosomal DNA and the enzymatic machinery for its expression and replication, and the *nuclear membrane* separates it from the rest of the cell, which is called *cytoplasm*. There are organelles within the cytoplasm, e.g. *mitochondria*, sites of respiration, and (in some cells) *chloroplasts*, sites of photosynthesis. In contrast, procaryotic cells have only a single cellular membrane and thus no membranous organelles. One molecular difference between the two types of cells is apparent in their genetic material. Procaryotes have a single chromosome (possibly present in more than one copy), while eucaryotes possess more than one chromosome.

Proteins

A eucaryotic or procaryotic cell contains thousands of different proteins, the most abundant class of biomolecules in cells. The genetic information contained in chromosomes determines the protein composition of an organism. As is true of many biomolecules, proteins exhibit functional versatility and are therefore utilized in a variety of biological roles. A few examples of biological functions of proteins are enzymatic activity (catalysis of chemical reactions), transport, storage and cellular structure.

Although biologically active proteins are macromolecules that may be very different in size and in shape, all are polymers composed by amino acids that form a chain. The number, chemical nature, and sequential order of amino acids in a protein chain determine the distinctive structure and characteristic chemical behavior of each protein. The native conformation of a protein is determined by interactions between the protein itself and

its aqueous environment, in which it reaches an energetically stable three–dimensional structure, most often the conformation requiring the least amount of energy to maintain. In this three dimensional structure, often very complex and involving more than one chain of amino acids, it is sometimes possible to identify places where chemical interaction with other molecules can occur. This places are called *interaction sites*, and are usually the basic entities in the abstract description of the behavior of a protein.

Nucleic Acids (DNA and RNA)

Similarly to proteins, nucleic acids are polymers, more precisely they are chains of nucleotides. Two types of nucleic acid exist: the *deoxyribonucleic acid* (DNA) and the *ribonucleic acid* (RNA). The former contains the genetic instructions for the biological development of a cellular form of life. In eucaryotic cells, it is placed in the nucleus and it is shaped as a double helix, while in procaryotic cells it is placed directly in the cytoplasm and it is circular. DNA contains the genetic information, that is inherited by the offspring of an organism. A strand of DNA contains genes, areas that regulate genes, and areas that either have no function, or a function yet unknown. Genes are the units of heredity and can be loosely viewed as the organism's "cookbook".

Like DNA, most biologically active RNAs are chains of nucleotides forming double stranded helices. Unlike DNA, this structure is not just limited to long double-stranded helices but rather collections of short helices packed together into structures akin to proteins. Various types of RNA exist, among these we mention the *Messenger RNA* (mRNA), that carries information from DNA to sites of protein synthesis in the cell, and the *Transfer RNA* (tRNA), that transfers a specific amino acid to a growing protein chain.

The Central Dogma of Molecular Biology

The description of proteins and nucleic acids we have given suggests a route for the flow of biological information in cells. In fact, we have seen that DNA contains instructions for the biological development of a cellular form of life, RNA carries information from DNA to sites of protein synthesis in the cell and provides amino acids for the development of new proteins, and proteins perform activities of several kinds in the cell. Schematically we have this flux of information:

$$\text{DNA} \xrightarrow{\text{transcription}} \text{RNA} \xrightarrow{\text{translation}} \text{Protein}$$

in which *transcription* and *translation* are the activities of performing a "copy" of a portion of DNA into a mRNA molecule, and of building a new protein by following the information found on the mRNA and by using the amino acids provided by tRNA molecules. This process is known as the *Central Dogma of Molecular Biology.*

Enzymes

Enzymes are proteins that behave as very effective catalysts, and are responsible for the thousands of coordinated chemical reactions involved in biological processes of living systems. Like any catalyst, an enzyme accelerates the rate of a reaction by lowering the energy of activation required for the reaction to occur. Moreover, as a catalyst, an enzyme is not destroyed in the reaction and therefore remains unchanged and is reusable.

The reactants of the chemical reaction catalyzed by an enzyme are called *substrate*. Substances that specifically decrease the rate of enzymatic activity are called *inhibitors*, and, in enzymology, inhibitory phenomena are studied because of their importance to many different areas of research. Inhibitors can be classified mainly in two types, either *competitive* or *noncompetitive*. The former are substances almost always structurally similar to the natural enzyme substrates and they bind to the enzyme at the interaction site where the substrates usually bind to. The latter are substances that bear no structural relationship to the substrates and that cannot interact at the active site of the enzyme, but must bind to some other portion of an enzyme.

Enzymes perform many important activities in cells. For example, DNA transcription and RNA translation are performed by enzymes, and in the external membrane of the cell there are enzymes responsible for transporting some molecules from the outside to the inside of the cell or vice–verse.

2.2 Notions of Probability Theory

A *probability distribution* is a function which assigns to every interval of the real numbers a probability $P(I)$, so that Kolmogorov axioms are satisfied, namely:

- for any interval I it holds $P(I) \geq 0$
- $P(\mathbb{R}) = 1$
- for any set of pairwise disjoint intervals $I_1, I_2, \ldots$ it holds $P(I_1 \cup I_2 \cup \ldots) = \sum P(I_i)$

A random variable on a real domain is a variable whose value is randomly determined. Every random variable gives rise to a probability distribution, and this distribution contains most of the important information about the variable. If X is a random variable, the corresponding probability distribution assigns to the interval $[a, b]$ the probability $P(a \leq X \leq b)$, i.e. the probability that the variable X will take a value in the interval $[a, b]$. The probability distribution of the variable X can be uniquely described by its *cumulative distribution function* $F(x)$, which is defined by

$$F(x) = P(X \leq x)$$

for any $x \in \mathbb{R}$.

A distribution is called *discrete* if its cumulative distribution function consists of a sequence of finite jumps, which means that it belongs to a discrete random variable X: a variable which can only attain values from a certain finite or countable set.

A distribution is called *continuous* if its cumulative distribution function is continuous, which means that it belongs to a random variable X for which $P(X = x) = 0$ for all $x \in R$.

Most of the continuous distribution functions can be expressed by a probability density function: a non-negative Lebesgue integrable function f defined on the real numbers such that

$$P(a \leq X \leq b) = \int_a^b f(x)\, dx$$

for all a and b.

The *support* of a distribution is the smallest closed set whose complement has probability zero.

An important continuous probability distribution function is the *exponential distribution*, which is often used to model the time between independent events that happen at a constant average rate. The distribution is supported on the interval $[0, \infty)$. The probability density function of an exponential distribution has the form

$$f(x, \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where $\lambda > 0$ is a parameter of the distribution, often called the rate parameter.

The cumulative distribution function, instead, is given by

$$F(x, \lambda) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

The exponential distribution is used to model Poisson processes, which are situations in which an object initially in state A can change to state B with constant probability per unit time λ. The time at which the state actually changes is described by an exponential random variable with parameter λ. Therefore, the integral from 0 to T over f is the probability that the object is in state B at time T.

In real-world scenarios, the assumption of a constant rate (or probability per unit time) is rarely satisfied. For example, the rate of incoming phone calls differs according to the time of day. But if we focus on a time interval during which the rate is roughly constant, such as from 2 to 4 p.m. during work days, the exponential distribution can be used as a good approximate model for the time until the next phone call arrives.

The *mean* or *expected value* of an exponentially distributed random variable X with rate parameter λ is given by

$$\mathrm{E}[X] = \frac{1}{\lambda}$$

In light of the example given above, this makes sense: if you receive phone calls at an average rate of 2 per hour, then you can expect to wait half an hour for every call.

Exponential distributions are at the base of *Continuous Time Markov Chains (CTMCs)*. A CTMC is a family of random variables $\{X(t) | t \geq 0\}$, where $X(t)$ is an observation made at time instant t and t varies over non–negative reals. The state space, namely the set of all possible values taken by $X(t)$, is a discrete set. Moreover, a CTMC must satisfy the *Markov (memoryless) property*: for any integer $k \geq 0$, sequence of time instances $t_0 < t_1 < \cdots < t_k$ and states $s_0, \ldots, s_k$ it holds

$$P(X(t_k) = s_k \,|\, X(t_{k-1}) = s_{k-1}, \ldots, X(t_1) = s_1) = P(X(t_k) = s_k \,|\, X(t_{k-1}) = s_{k-1})$$

where $P(E_1|E_2)$ denotes the probability of event E_1 when it is known that event E_2 happens (this is called *conditional probability*).

Intuitively, the memoryless property means that the probability of making a transition to a particular state at a particular time depends only on the current state, not the previous history of states passed through. The exponential distribution is the only continuous probability distribution which exhibits this memoryless property, hence it is the only one that can be used in the definition of CTMCs.

Formally, a CTMC is defined as follows.

Definition 2.1 (Continuous Time Markov Chain)**.** *A CTMC is a triple* $\langle \mathsf{S}, \mathsf{R}, \pi \rangle$*, where*

- S *is the set of states,*
- $\mathsf{R} : \mathsf{S} \times \mathsf{S} \mapsto \mathbb{R}^{\geq 0}$ *is the* transition function*,*
- $\pi : \mathsf{S} \mapsto [0, 1]$ *is the* starting distribution*.*

The system is assumed to pass from a configuration modeled by a state s to another one modeled by a state s' by consuming an exponentially distributed quantity of time, in which the parameter of the exponential distribution is $\mathsf{R}(s, s')$. The summation $\sum_{s' \in \mathsf{S}} \mathsf{R}(s, s')$ is called the *exit rate* of state s. Finally, the system is assumed to start from a configuration modeled by a state $s \in \mathsf{S}$ with probability $\pi(s)$, and $\sum_{s \in \mathsf{S}} \pi(s) = 1$. If the set of states of the CTMC is finite ($\mathsf{S} = \{s_1, \ldots, s_n\}$), then the transition function R can be represented as a square matrix of size n in which the element at position (i, j) is equal to $\mathsf{R}(s_i, s_j)$.

2.3 Stochastic Simulation of Chemical Reactions

The fundamental empirical law governing reaction rates in biochemistry is the *law of mass action*. This states that for a reaction in a homogeneous medium, the reaction rate will be proportional to the concentrations of the individual reactants involved. A chemical reaction is usually represented by the following notation:

$$\ell_1 S_1 + \ell_2 S_2 \underset{k_{-1}}{\overset{k}{\rightleftharpoons}} \ell_3 S_3 + \ell_4 S_4$$

where $S_1, \ldots, S_4$ are molecules, $\ell_1, \ldots, \ell_4$ are their stoichiometric coefficients, and k, k_{-1} are the kinetic constants. We denote with L the sum of the stoichiometric coefficients, that is the total number of reactant molecules. The use of the symbol $\rightleftharpoons$ denotes that the reaction is *reversible* (i.e. it can occur in both directions). Irreversible reactions are denoted by the single arrow $\rightarrow$.

For example, given the simple reaction

$$2A \underset{k_{-1}}{\overset{k}{\rightleftharpoons}} B$$

the rate of the production of molecule B for the law of mass action is:

$$\frac{dB_+}{dt} = k[A]^2$$

and the rate of destruction of B is:

$$\frac{dB_-}{dt} = k_{-1}[B]$$

where $[A], [B]$ are the *concentrations* (i.e. moles over volume unit) of the respective molecules. In general, the rate of a reaction is:

$$k[S_1]^{\ell_1} \cdots [S_\rho]^{\ell_\rho}$$

where $S_1, \ldots, S_\rho$ are all the distinct molecular reactants of the reaction.

The rate of a reaction is usually expressed in $moles \cdot s^{-1}$ (it is a speed), therefore the measure unit of the kinetic constant is $moles^{-(L-1)} \cdot s^{-1}$.

In [31] Gillespie gives a stochastic formulation of chemical kinetics that is based on the theory of collisions and that assumes a stochastic reaction constant c_μ for each considered chemical reaction R_μ. The reaction constant c_μ is such that $c_\mu dt$ is the probability that a particular combination of reactant molecules of R_μ will react in an infinitesimal time interval dt, and can be derived with some approximations from the kinetic constant of the chemical reaction.

The probability that a reaction R_μ will occur in the whole solution in the time interval dt is given by $c_\mu dt$ multiplied by the number of distinct R_μ molecular reactant combinations. For instance, the reaction

$$R_1 : S_1 + S_2 \rightarrow 2S_1 \tag{2.1}$$

will occur in a solution with X_1 molecules S_1 and X_2 molecules S_2 with probability $X_1 X_2 c_1 dt$. Instead, the inverse reaction

$$R_2 : 2S_1 \rightarrow S_1 + S_2 \tag{2.2}$$

will occur with probability $\frac{X_1(X_1-1)}{2!}c_2 dt$. The number of distinct R_μ molecular reactant combinations is denoted by Gillespie with h_μ, hence, the probability of R_μ to occur in dt (denoted with $a_\mu dt$) is

$$a_\mu dt = h_\mu c_\mu dt \ .$$

Now, assuming that $S_1, \ldots, S_n$ are the only molecules that may appear in a chemical solution, a *state* of the simulation is a tuple $(X_1, \ldots, X_n)$ representing a solution containing X_i molecules S_i for each i in $1, \ldots, n$. Given a state $(X_1, \ldots, X_n)$, a set of reactions $R_1, \ldots, R_M$, and a value t representing the current time, the algorithm of Gillespie performs two steps:

1. The time $t + \tau$ at which the next reaction will occur is randomly chosen with τ exponentially distributed with parameter $\sum_{\nu=1}^{M} a_\nu$;
2. The reaction R_μ that has to occur at time $t+\tau$ is randomly chosen with probability $a_\mu dt$.

The function $P_g(\tau, \mu)dt$ represents the probability that the next reaction will occur in the solution in the infinitesimal time interval $(t+\tau, t+\tau+dt)$ and will be R_μ. The two steps of the algorithm imply

$$P_g(\tau, \mu)dt = P_g^0(\tau) \cdot a_\mu dt$$

where $P_g^0(\tau)$ corresponds to the probability that no reaction occurs in the time interval $(t, t+\tau)$. Since $P_g^0(\tau)$ is defined as

$$P_g^0(\tau) = \exp\left(-\sum_{\nu=1}^{M} a_\nu \tau\right)$$

we have, for $0 \leq \tau < \infty$,

$$P_g(\tau, \mu)dt = \exp\left(-\sum_{\nu=1}^{M} a_\nu \tau\right) \cdot a_\mu dt \ .$$

Finally, the two steps of the algorithm can be implemented in accordance with $P_g(\tau, \mu)$ by choosing τ and μ as follows:

$$\tau = \left(\frac{1}{\sum_{\nu=1}^{M} a_\nu}\right) \ln\left(\frac{1}{r_1}\right) \qquad \mu = \text{the integer for which } \sum_{\nu=1}^{\mu-1} a_\nu < r_2 \sum_{\nu=1}^{M} a_\nu \leq \sum_{\nu=1}^{\mu} a_\nu$$

where $r_1, r_2 \in [0, 1]$ are two real values generated by a random number generator. After the execution of the two steps, the clock has to be updated to $t + \tau$ and the state has to be modified by subtracting the molecular reactants and adding the molecular products of R_μ.

2.4 Transition Systems and Bisimulations

In this section we present some basic notions of process description language theory that are needed in the remainder of the thesis. In particular we recall the definitions of *Transition System* (TS), *Labeled Transition System* (LTS) and *bisimulation relation* over LTSs, and we show how a LTS can be specified by means of inference rules.

A TS is a mathematical model describing something having a notion of *state* (or *configuration*) which may evolve by performing steps from one state to another. A TS is formally defined as follows.

Definition 2.2 (Transition System)**.** *A* Transition System (TS) *is a pair* $(S, \rightarrow)$ *where* S *is the set of* states *ranged over by* $s, s_0, s_1, \ldots$*, and* $\rightarrow \subseteq S \times S$ *is the* transition relation*. We write* $s_i \rightarrow s_j$ *when* $(s_i, s_j) \in \rightarrow$*.*

In a TS, the nature of the elements of S usually depends on what the TS describes. For instance, if the TS is used to describe the execution of programs written in some imperative programming language, its states will be pairs $\langle C, \sigma \rangle$ where C is a program and σ is its store. Instead, if the TS is used to describe the evolution of chemical solution in which reactions may occur, its states will be multisets M describing the multitude of molecules that are present in the chemical solution. The transition relation, instead, represents the steps that can be performed by the system from one state to another one. In fact, $s_0 \rightarrow s_1$ means that a system in state s_0 in one step can change its state to s_1. In the example of the imperative programming language one step corresponds to the execution of a single command of the program, and in the chemical example one step corresponds to one occurrence of a chemical reaction in the chemical solution.

In a TS, a state s is *reachable* from another one s_0 if a system in state s_0 can perform a finite (and possibly empty) sequence of transition at the end of which the state of the system is s. More precisely, s is reachable from s_0 if either $s_0 = s$, or there exist $s_1, \ldots, s_n \in S$ such that $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_n \rightarrow s$. We write $s_0 \Rightarrow s$ if s is reachable from s_0. We denote with $Reach(s_0) \subseteq S$ the set of all states that are reachable from s_0.

A LTS is a TS in which transitions are enriched with labels.

Definition 2.3 (Labeled Transition System)**.** *A* Labeled Transition System (LTS) *is a triple* $(S, L, \rightarrow)$ *where* S *is the set of* states *(or* configurations*) ranged over by* $s, s_0, s_1, \ldots$*,* L *is a set of* labels *ranged over by* $l, l_0, l_1, \ldots$ *and* $\rightarrow \subseteq S \times L \times S$ *is the* labeled transition relation*. We write* $s_0 \xrightarrow{l} s_1$ *when* $(s_0, l, s_1) \in \rightarrow$*.*

In a LTS the label of a transition usually denotes the event that has caused the transition. For instance, the operational semantics of CCS [54] is a LTS. CCS is a formalism describing concurrent processes that are able to interact by synchronizing on channels. A synchronization is obtained by two processes performing one input and one output actions, respectively, on the same channel. In the LTS of CCS, a label $\overline{a}$ denotes an output action on channel a, while a label a denotes an input on the same channel. An internal synchronization is represented by a transition labeled with τ.

Often, the set of labels L of a LTS contains a special label denoting an hidden action. In CCS, for example, label τ denotes this kind of actions, and we use the same notation in this section. We denote with $s_0 \Rightarrow s_n$ a finite (and possibly empty) sequence of τ–labeled transitions from s_0 to s_n, namely $s_0 \Rightarrow s_n$ if either $s_0 = s_n$, or there exist $s_1, \ldots, s_{n-1} \in S$ such that $s_0 \xrightarrow{\tau} s_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} s_{n-1} \xrightarrow{\tau} s_n$. Moreover, if $l \neq \tau$ we denote with $s_0 \stackrel{l}{\Longrightarrow} s_n$ a finite (and non empty) sequence of transitions from s_0 to s_3 such that there exist $s_1, s_2 \in S$ such that $s_0 \Rightarrow s_1 \xrightarrow{l} s_2 \Rightarrow s_3$. Finally, we denote with $\stackrel{\overline{l}}{\Longrightarrow}$ the relations corresponding either to $\Rightarrow$ if $l = \tau$, or to $\stackrel{l}{\Longrightarrow}$ if $l \neq \tau$.

LTSs may describe the behavior of the modeled system in great detail. Relations on states of a LTS can be defined to compare the behavior of two modeled systems. In particular, behavioral equivalences are reflexive, transitive and symmetric relations that relate systems that are not distinguished by any external observer, according to a given notion of observation. We recall here the notion of (strong) bisimulation equivalence which relates two states in a LTS when they are step by step able to perform transitions with the same lables.

Definition 2.4 (Strong Bisimulation)**.** *Given an LTS $(S, L, \rightarrow)$, a relation $R \subseteq S \times S$ is a* strong bisimulation *if whenever $(s_0, s_2) \in R$ the following two conditions hold:*

$s_0 \rightarrow s_1 \implies \exists s_3 \in S$ *such that* $s_2 \rightarrow s_3$ *and* $(s_1, s_3) \in R$*;*

$s_2 \rightarrow s_3 \implies \exists s_1 \in S$ *such that* $s_0 \rightarrow s_1$ *and* $(s_1, s_3) \in R$*.*

The strong bisimilarity $\sim$ *is the largest of such relations.*

In comparing the behavior of two systems, most of the time hidden actions can be ignored. For this reason a different notion of bisimulation equivalence, called weak bisimulation, is often considered.

Definition 2.5 (Weak Bisimulation)**.** *Given an LTS $(S, L, \rightarrow)$, a relation $R \subseteq S \times S$ is a* weak bisimulation *if whenever $(s_0, s_2) \in R$ the following two conditions hold:*

$s_0 \xrightarrow{l} s_1 \implies \exists s_3 \in S$ *such that* $s_2 \stackrel{\overline{l}}{\Longrightarrow} s_3$ *and* $(s_1, s_3) \in R$*;*

$s_2 \xrightarrow{l} s_3 \implies \exists s_1 \in S$ *such that* $s_0 \stackrel{\overline{l}}{\Longrightarrow} s_1$ *and* $(s_1, s_3) \in R$*.*

The weak bisimilarity $\approx$ *is the largest of such relations.*

Following the Structural Operational Semantics (SOS) approach [61], LTSs in which states are terms built over some signature are usually specified by means of a set of inference rules. Before discussing this point, let us recall some preliminary notions.

Let us consider a countably infinite set of *variables* V, ranged over by $x, y, z, \ldots$. A *signature* consists of a set of *function symbols*, disjoint from V, together with an *arity* mapping that assigns a natural number $ar(f)$ to each function symbol f. Functions of arity zero are usually called *constants*, while function of arity greather than zero are usually called *operators*. Given a constant f we write f for $f()$.

Definition 2.6 (Open Terms)**.** *The set of* open terms $T(\Sigma)$ *over a signature* Σ *is the least set such that: (i)* $V \subseteq T(\Sigma)$*, and (ii) given a function symbol* f *and* $t_1, \ldots, t_{ar(f)} \in T(\Sigma)$ *it holds* $f(t_1, \ldots, t_{ar(f)}) \in T(\Sigma)$*. The set* $T(\Sigma)$ *is ranged over by* $t, u, v, \ldots$*.*

Terms that does not contain variables are usually called *closed terms* (or *ground terms*). The set of closed terms is denoted by $T_g(\Sigma)$. In the rest of the thesis we will use also the terminology of pattern and term to denote open and closed terms, respectively.

The set of closed terms over Σ gives the *term algebra* of Σ. We recall that, given a signature Σ, a Σ–algebra is a pair (A, Σ_A), where A is a set called *carrier* and Σ_A is a set of functions $\{f_A : A^n \mapsto A | f \in \Sigma \text{ and } ar(f) = n\}$. Essentially, (A, Σ_A) is an interpretation of Σ. Now, the term algebra of Σ is the Σ–algebra having $T_g(\Sigma)$ as carrier, and, for each $f \in \Sigma$ with $ar(f) = n$, a function mapping closed terms $t_1, \ldots, t_n$ to term $f(t_1, \ldots, t_n)$.

A *substitution* is a mapping $\sigma : V \mapsto T(\Sigma)$. A substitution can be extended trivially to a mapping from terms to terms, namely, $\sigma(t)$ is the term obtained by replacing all the variables occurring in t by $\sigma(x)$. A substitution is called *instantiation* (or *closed substitution*) if it maps variables to closed terms.

A *context* $C[x_1, \ldots . x_n]$ denotes an open term in which at most the distinct variables $x_1, \ldots, x_n$ may appear. The term $C[t_1, \ldots, t_n]$ is obtained by replacing all occurrences of variables x_i in $C[x_1, \ldots, x_n]$ by t_i, for $1 \leq i \leq n$.

An LTS whose states are terms built over some signature can be specified by means of a set of inference rules. An inference rule for the specification of an LTS (a *transition rule*) is a logical rule having the form

$$\frac{t_1 \xrightarrow{l_1} t'_1 \quad \cdots \quad t_n \xrightarrow{l_n} t'_n}{t \xrightarrow{l} t'}$$

where $t_i \xrightarrow{l_i} t'_i$, for $1 \leq i \leq n$, are the *premises* and $t \xrightarrow{l} t'$ is the *conclusion*. A transition rule states that whenever the premises are transitions of the LTS, then also the conclusion is a transition of the LTS. Side conditions can be associated to a transition rule with the effect of imposing that the conclusion of the rule is a transition of the LTS whenever both the premises and the side conditions are satisfied. A transition rule without premises is called an *axiom*, and a (non empty and possibly infinite) LTS can be specified by providing a set of transition rules with at least one axiom.

Part I

Qualitative Modeling of Biological Systems

Chapter 3

Calculi of Looping Sequences

Process calculi, in particular the π–calculus, allow modeling cellular components by describing their interaction capabilities as input/output actions on communication channels representing chemical reactions. This kind of abstractions favors semantic compositionality as, in principle, the behavior of a cellular component can be described as a labeled transition system, and the behavior of a system of cellular components can be obtained by appropriately merging the labeled transition systems of its components.

Compositionality is an extremely useful property of a formalism, and it is one of the main motivations for the application of process calculi to the description of biological systems. Moreover, the lack of compositionality is the typical criticism on models of biological systems based on rewrite rules. On the other hand, rewrite systems often allow describing biological systems with a notation which is much more readable than the one of process calculi, as they separate the description of the states of the system from the description of the reactions that may occur. Moreover, rewrite systems often allow a more detailed description of the physical structure of the modeled biological components, and usually are more general than process calculi. With generality we mean the ability of describing new kinds of interactions when needed. This is often allowed in rewrite systems by the fact that interactions are described by rewrite rules, which are part of the specification of a system, while in process calculi they are described by applications of pre–defined operators, hence the possible kinds of interactions are determined a priori.

In this chapter we develop a formalism for the description of biological systems based on term rewriting and including some typical features of process calculi for concurrency. What we want to achieve is a formalism that allows describing (at least) proteins, DNA fragments, membranes and macromolecules in general, without ignoring the physical structure of these elements, and by keeping the syntax and the semantics of the formalism as simple as possible.

The kind of structures that most frequently appear in cellular components is probably the sequence. A DNA fragment, for instance, is a sequence of nucleic acids, and it can be seen, at a higher level of abstraction, also as a sequence of genes. Proteins are sequences of amino acids, and they can be seen also as sequences of interaction sites. Membrane, instead, are essentially closed surfaces interspersed with proteins and molecules of various kinds, hence we can see them abstractly as closed circular sequences whose elements or subsequences describe the entities that are placed in the membrane surface. Finally, there are usually many components in a biological system, some of which may be contained

in some membranes, and membranes may be nested in various ways, thus forming a hierarchical structure that may change over time.

By following the viewpoint of cellular systems just presented, in order to model these systems we should be able to describe the evolution of sequences, which may be circular and which may contain something, for instance other sequences. In the rest of the chapter we develop a formalism based on term rewriting which tries to fulfill these requirements. The formalism is called *Calculus of Looping Sequences* (CLS for short) and it is presented in three variants: the first one, called Full–CLS, is defined simply by considering a signature for terms in which the operators can be used without syntactical constraints and can be used to describe biological systems quite easily, as we show in an example of bacteriophage replication and bacterial sporulation. The second variant, that we actually call CLS, contains a restriction on the syntax of terms which simplifies the semantics of the formalism. This restriction reduces the expressiveness of the model, but we claim that the expressiveness of CLS is anyway sufficient to describe the biological systems of interest. To this aim, we give a real example of gene regulation in E.coli. Finally, the third variant, called LCLS, is an extension of CLS in which links can be established between elements of different sequences. These links allow modeling protein interaction at the domain level and are inspired by the way of modeling protein interactions introduced in the seminal work by Danos and Laneve [23].

3.1 Definition of Full–CLS

In this section we introduce the *Full Calculus of Looping Sequences* (Full–CLS). As already said before, we have to define terms able to describe (i) sequences, (ii) which may be closed, (iii) which may contain something, and (iv) which may be juxtaposed to other sequences in the system. For each of these four points we define an operator in the grammar of terms. Moreover, we assume a possibly infinite alphabet of elements $\mathcal{E}$ ranged over by $a, b, c, \ldots$ to be used as the building blocks of terms, and a neutral element ϵ representing the empty term. Terms of the calculus are defined as follows.

Definition 3.1 (Terms)**.** Terms T *of Full–CLS are given by the following grammar:*

$$T \;::=\; a \;\;\big|\;\; \epsilon \;\;\big|\;\; T \cdot T \;\;\big|\;\; \big(T\big)^L \;\;\big|\;\; T \rfloor T \;\;\big|\;\; T \,|\, T$$

where a *is a generic element of* $\mathcal{E}$*. We denote with* $\mathcal{T}$ *the infinite set of terms.*

Terms include the elements in the alphabet $\mathcal{E}$ and the empty term ϵ. Moreover, the following operators can be used to build more complex terms:

- Sequencing (or concatenation) $\_ \cdot \_$: creates a sequence whose elements are the two terms to which it is applied.

- Looping $\big(\_\big)^L$: creates a closed circular sequence of the term to which it is applied. The operator is called looping because, as we shall see, it is always possible to rotate the representation of the circular sequence.

- Containment $\_ \rfloor \_$: represents the containment of the second term to which it is applied into the first one.

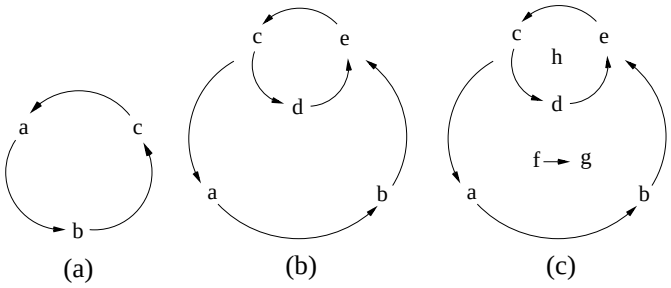


Figure 3.1: Examples of CLS terms

- Parallel composition $\_ \mid \_$: represents the juxtaposition of the two terms to which it is applied.

Brackets can be used to indicate the order of application of the operators in a term. We assume the $\cdot$ operator to have the highest precedence and the $\rfloor$ operator to have the precedence over the $\mid$ operator. Therefore $T_1 \rfloor T_2 \mid T$ stands for $(T_1 \rfloor T_2) \mid T$. Moreover, we assume $\rfloor$ to be right–associative, therefore with $T_1 \rfloor T_2 \rfloor T$ we denote the term $T_1 \rfloor (T_2 \rfloor T)$.

Some simple examples of terms are depicted in Figure 3.1. In the figure, example (a) shows the simple term $\left(a \cdot b \cdot c\right)^L$ representing a looping sequence composed by elements a, b and c. Example (b) shows the term $\left(a \cdot b \cdot \left(c \cdot d \cdot e\right)^L\right)^L$, that is similar to the term of example (a) but with the c element replaced by another looping sequence whose elements are c, d and e. Note that the small looping sequence $\left(c \cdot d \cdot e\right)^L$ is not contained into the bigger one, but it is one of the elements that compose it. Finally, example (c) shows the term $\left(a \cdot b \cdot (\left(c \cdot d \cdot e\right)^L \rfloor h)\right)^L \rfloor f \cdot g$. In this case we have the same looping sequences of example (b), but they are not empty, namely the smaller one contains element h, by the application of the containment operator to $\left(c \cdot d \cdot e\right)^L$, and the bigger one contains the sequence $f \cdot g$, by the other application of the containment operator.

The syntax of terms is single–sorted, hence operators can be applied freely. This causes some ambiguous situations that must be discussed. Let us consider the following examples of terms:

$$(T_1 \mid T_2) \cdot T_3 \qquad \left(T_1 \mid T_2\right)^L \qquad (T_1 \mid T_2) \rfloor T_3$$

In all these three examples we have an operator applied to the parallel composition of T_1 and T_2. In the first case sequencing is applied, in the second case looping and in the third case containment. Now, consider the first example: parallel composition represents juxtaposition, hence the two components can be close to each other, but they are not connected. Sequential composition, instead, denotes a physical connection between its components, hence in this case T_3 what is connected to? It cannot be connected to both T_1 and T_2, otherwise this would create a connection between them that we do not want, hence it must be connected either only to T_1 or only to T_2. The same situation occurs in the other two examples: in the second one, since we want a looping sequence to be a single completely connected component as it must model closed surfaces, we cannot allow it to be formed by a parallel composition. In the third example, again, we have that term

T_3 cannot be contained in both T_1 and T_2, because they represent two separated entities.

Another ambiguous situation regards containment. The use of the containment operator makes sense only if its first operand is a looping sequence, which represents something closed, and therefore able to contain something. Hence, what a use of containment as in $a \cdot b \cdot c \rfloor T$ should mean?

All these ambiguities can be removed by appropriately defining a *structural congruence* relation. The notion of structural congruence is very common in process calculi: it is a relation used to consider as equal syntactically different terms representing the same process. A notion similar to structural congruence exists also in term rewriting systems, and it is the additional relation used in *class rewriting* [75] (or *rewriting modulo a congruence*). We define a structural congruence relation on Full–CLS terms as follows.

Definition 3.2 (Structural Congruence)**.** *The structural congruence $\equiv$ is the least congruence relation on terms satisfying the following axioms:*

$A1.$	$(T_1 \mid T_2) \cdot T \equiv (T_1 \cdot T) \mid T_2$	$A8.$	$\big(T_1 \cdot T_2\big)^L \equiv \big(T_2 \cdot T_1\big)^L$
$A2.$	$T \cdot (T_1 \mid T_2) \equiv (T \cdot T_1) \mid T_2$	$A9.$	$(T_1 \cdot T_2) \cdot T_3 \equiv T_1 \cdot (T_2 \cdot T_3)$
$A3.$	$\big(T \mid T_1\big)^L \equiv \big(T\big)^L \mid T_1$	$A10.$	$(T_1 \mid T_2) \mid T_3 \equiv T_1 \mid (T_2 \mid T_3)$
$A4.$	$(T_1 \mid T_2) \rfloor T \equiv (T_1 \rfloor T) \mid T_2$	$A11.$	$T \mid T_1 \mid T_2 \equiv T \mid T_2 \mid T_1$
$A5.$	$a \rfloor T \equiv a \mid T$	$A12.$	$T \mid \epsilon \equiv T \rfloor \epsilon \equiv T$
$A6.$	$(T_1 \cdot T_2) \rfloor T \equiv (T_1 \cdot T_2) \mid T$	$A13.$	$T \cdot \epsilon \equiv \epsilon \cdot T \equiv T$
$A7.$	$(T_1 \rfloor T_2) \rfloor T_3 \equiv T_1 \rfloor (T_2 \mid T_3)$	$A14.$	$\big(\epsilon\big)^L \equiv \epsilon$

Axioms A1, A2, A3 and A4 deal with the ambiguity of the parallel composition described above, and state that if we apply either sequential composition, containment or looping to a parallel composition of terms, these operators act upon the first term of the parallel composition.

Axioms A5, A6 and A7, instead, deal with the ambiguity related to the containment operator, and state that when containment is applied to something that is not a looping sequence, it can be replaced by parallel composition.

Another very important axiom, which motivates the terminology of looping sequence, is A8. This axiom states that a sequence having a looping operator applied to it can be rotated freely.

A structural congruence relation usually states associativity and commutativity of operators. Here, we want sequencing and parallel composition to be associative, and this is expressed by axioms A9 and A10, respectively. Moreover, we want parallel composition to be commutative. However, since the first term of a parallel composition plays the special role described by axioms A1, A2, A3 and A4, we cannot allow full commutativity. To explain the problem, let us assume for a moment that $T_1 \mid T_2 \equiv T_2 \mid T_1$ is an axiom of the structural congruence, and consider the term $a \cdot b \mid c$. By applying axiom A1, then the full commutativity axiom just introduced, and then axiom A1 again, we obtain the following sequence of equalities:

$$a \cdot b \mid c \equiv (a \mid c) \cdot b \equiv (c \mid a) \cdot b \equiv c \cdot b \mid a$$

hence the initial term would be considered equivalent to a term in which c takes the place of a in the sequence. In order to avoid this kind of mistakes, we forbid commutativity

of the first element of a parallel composition, as stated by axiom A11. However, in what follows we will show that the full commutativity can be derived by the other axioms of the structural congruence in all safe cases.

The last three axioms, namely axioms A12, A13 and A14, describe the neutral role of ϵ and $\left(\epsilon\right)^L$ with respect to the operators of the calculus. We remark that in axiom A2 the neutral term ϵ is placed on the right hand side of the $\mid$ operator, otherwise ϵ could be inserted at the left hand of a series of parallel compositions and its first term would lose its privileged role.

We want to remark that assigning a special role to an element of a parallel composition is not unusual. For instance, in [29, 32] the last element in a series of parallel compositions has the special role of giving the result of the computation of the whole series. Thus, it cannot be commuted.

Proposition 3.3. $T \rfloor (T_1 \mid T_2) \equiv T \rfloor (T_2 \mid T_1)$.

Proof. The equivalence can be derived as follows: $T \rfloor (T_1 \mid T_2) \overset{A12}{\equiv} (T \rfloor \epsilon) \rfloor (T_1 \mid T_2) \overset{A7}{\equiv} T \rfloor (\epsilon \mid T_1 \mid T_2) \overset{A11}{\equiv} T \rfloor (\epsilon \mid T_2 \mid T_1) \overset{A7}{\equiv} (T \rfloor \epsilon) \rfloor (T_2 \mid T_1) \overset{A12}{\equiv} T \rfloor (T_2 \mid T_1)$. □

The proposition shows that the first element of a series of parallel compositions can be commuted when the whole series is contained inside another term. As a consequence, to have unrestricted commutativity of a parallel composition at the top level of a term, one can insert the term into the term $\left(\epsilon\right)^L$ by using the containment operator. In this way we forbid the first element of a series of parallel compositions to commute only when the whole series is an element of a sequence. Standard commutativity holds otherwise. In what follows we will always assume that Full–CLS terms are contained at top–level into $\left(\epsilon\right)^L$, hence we will always assume full–commutativity of the parallel composition operator at top–level.

Now we define rewrite rules, which can be used to describe the evolution of terms. Roughly, a rewrite rule is a triple consisting of two terms and one condition to be satisfied. The two terms describe what term the rule can be applied to and the term obtained after the application of the rule, respectively, and the condition must be satisfied before applying the rule.

In order to allow a rule to be applied to a wider range of terms, we introduce variables in the terms of a rule. We assume a set $\mathcal{V}$ of variables ranged over by $X, Y, Z, \ldots$, and we call *patterns* terms enriched with variables. The syntax of patterns is therefore as follows.

Definition 3.4 (Patterns)**.** Patterns P *of Full–CLS are given by the following grammar:*

$$P \ ::= \ a \ \mid \ \epsilon \ \mid \ P \cdot P \ \mid \ \left(P\right)^L \ \mid \ P \rfloor P \ \mid \ P \mid P \ \mid \ X$$

where a is a generic element of $\mathcal{E}$, and X is a generic element of $\mathcal{V}$. We denote with $\mathcal{P}$ the infinite set of patterns.

We assume the structural congruence relation to be trivially extended to patterns. An *instantiation* is a partial function $\sigma : \mathcal{V} \to \mathcal{T}$. Given $P \in \mathcal{P}$, with $P\sigma$ we denote the term obtained by replacing each occurrence of each variable $X \in \mathcal{V}$ appearing in P with the corresponding term $\sigma(X)$. With Σ we denote the set of all the possible instantiations and, given $P \in \mathcal{P}$, with $Var(P)$ we denote the set of variables appearing in P. Note that

if $Var(P) = \varnothing$, then $P \in \mathcal{T}$. Finally, we define a function $occ : \mathcal{E} \times \mathcal{T} \to \mathbb{N}$ such that $occ(a, T)$ returns the number of the elements a syntactically occurring in the term T. Now we can define rewrite rules.

Definition 3.5 (Rewrite Rules)**.** *A* rewrite rule *is a triple* (P_1, P_2, Σ') *such that* $P_1, P_2 \in \mathcal{P}$, $P_1 \not\equiv \epsilon$, $Var(P_2) \subseteq Var(P_1)$, $\Sigma' \subseteq \Sigma$ *and, for all* $\sigma \in \Sigma'$, $Var(P_1) \subseteq Dom(\sigma)$. *We denote with* $\Re$ *the infinite set of all the possible rewrite rules. We say that a rewrite rule is* ground *if* $Var(P_1) = Var(P_2) = \varnothing$, *and a set of rewrite rules* $\mathcal{R} \in \Re$ *is* ground *if all the rewrite rules it contains are ground.*

A rewrite rule (P_1, P_2, Σ') states that a term $P_1\sigma$, obtained by instantiating variables in P_1 by an instantiation function $\sigma \in \Sigma'$, can be transformed into the term $P_2\sigma$. Note that we assume $Var(P_2) \subseteq Var(P_1) \subseteq Dom(\sigma)$, hence all the variables of P_1 and P_2 are instantiated by σ. A rule can be applied to all the terms which can be obtained by instantiating the variables in P_1 with any of the instantiations in Σ'. For instance, if $\Sigma' = \{\sigma \in \Sigma | occ(a, \sigma(X)) = 0\}$, then a rule $(b \cdot X \cdot b, c \cdot X \cdot c, \Sigma')$ can be applied to $b \cdot c \cdot b$ (obtaining $c \cdot c \cdot c$) and to $b \cdot c \cdot c \cdot b$ (obtaining $c \cdot c \cdot c \cdot c$), but not to $b \cdot a \cdot b$.

In what follows, we shall often write a rewrite rule as $T \mapsto T'$ $[\mathcal{C}]$ instead of $(T, T', \Sigma' = \{\sigma \in \Sigma \mid \mathcal{C}\sigma\})$, where $\mathcal{C}$ is a condition, and we shall omit Σ' when $\Sigma' = \Sigma$ and write $T \mapsto T'$. For instance, with $b \cdot X \cdot b \mapsto c \cdot X \cdot c$ $[occ(a, X) = 0]$ we denote $(b \cdot X \cdot b, c \cdot X \cdot c, \Sigma' = \{\sigma \in \Sigma | occ(a, \sigma(X)) = 0\})$.

The association of rewrite rules with conditions to be satisfied before each application is quite usual in term rewriting, in particular it is typical of *conditional rewriting* [75].

Now we define the semantics of Full–CLS as a transition system. States of the transition system are terms, and transitions corresponds to rule applications. Given an initial term one can use the transition relation to compute all the possible evolutions caused by applications of rewrite rules to its subterms.

Definition 3.6 (Semantics)**.** *Given a set of rewrite rules* $\mathcal{R} \subseteq \Re$, *the* semantics *of Full–CLS is the least transition relation* $\to$ *on terms closed under* $\equiv$, $\_|\_$, $\_\rfloor\_$, $\_\cdot\_$, $(\_)^L$ *and satisfying the following inference rule:*

$$\frac{(P_1, P_2, \Sigma') \in \mathcal{R} \qquad P_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma'}{P_1\sigma \to P_2\sigma}$$

A *model* in Full–CLS is given by a term describing the initial state of the modeled system and by a set of rewrite rules describing all the possible events that may occur in the system. We now give two simple examples of Full–CLS models of biological phenomena. The examples aim at showing some peculiarities of the formalism and the use of the semantics to study the possible evolutions of the described systems.

Example 3.7. We describe a very simple interaction between two membranes, one inside the other, in which the inner one contains a molecule. (Think for example of a vesicle containing a molecule inside the cellular membrane.) To make the example a bit more complete we assume that the two membranes can increase their size, until they reach some precise boundaries. When the inner membrane becomes greather than a certain size, it could break and leave the contained molecule in the environment. Moreover, at any time, but before breaking, the inner membrane can join the outer one.

The system can be modeled as follows. We model the outer membrane as a looping sequence composed by a elements, and the inner one as a looping sequence composed by

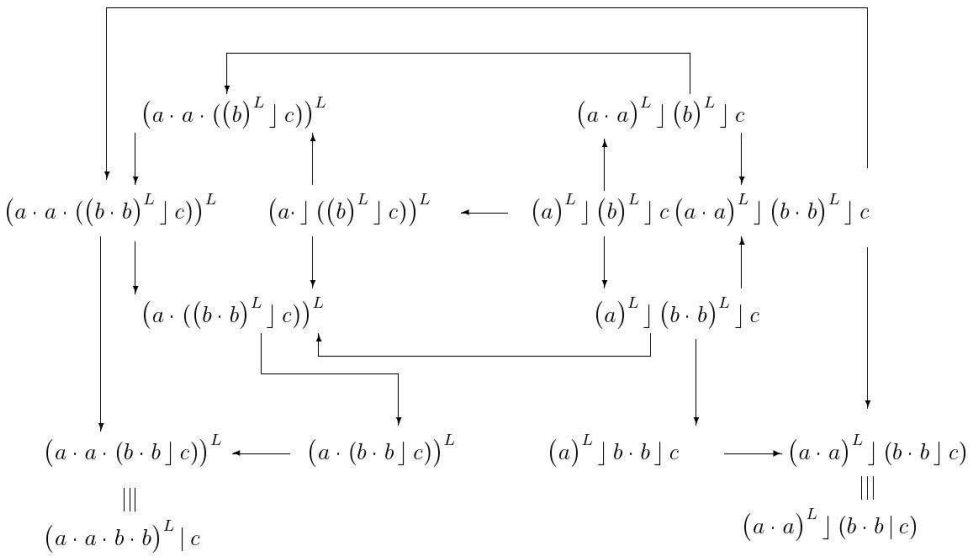


Figure 3.2: The transition system of the example.

b elements. Moreover, c is the molecule contained in the inner membrane. We model the size of a membrane as the number of elements composing it, and we assume n, m and k to be the maximum size of the outer membrane, the maximum size of the inner one, and the size after which the inner membrane can break, respectively. The rewrite rules describing the possible events occurring in the system are the following:

1. $\quad (a \cdot X)^L \mapsto (a \cdot a \cdot X)^L \qquad [occ(a, \sigma(X)) < n - 1]$
2. $\quad (b \cdot X)^L \mapsto (b \cdot b \cdot X)^L \qquad [occ(b, \sigma(X)) < m - 1]$
3. $\quad (b \cdot X)^L \mapsto b \cdot b \cdot X \qquad [occ(b, \sigma(X)) \geq k]$
4. $\quad (a \cdot X)^L \rfloor (b \cdot Y)^L \rfloor Z \mapsto (a \cdot X \cdot ((b \cdot Y)^L \rfloor Z))^L$

The four rules describe growth of the outer membrane, growth of the inner membrane, breaking of the inner membrane and joining of the two membranes, respectively. We model the initial state of the system as the term

$$(a)^L \rfloor (b)^L \rfloor c$$

and we show in Figure 3.2 the transition system obtained from this term when $n = m = 2$ and $k = 1$.

First of all, note that the set of states of the transition system is finite (unfortunately, this happens rarely in models of real systems). Moreover, note that the system may reach two different final states: the first, on the bottom right of the figure, is the state in which the inner membrane has broken before joining the outer one, the second, on the bottom left, is the state in which the inner membrane has broken after joining the outer one. It is worth noticing that in the latter case the content of the inner membrane is freed in the environment, and not inside the outer membrane (see Figure 3.3 for a graphical representation of this phenomenon). This is caused by axiom A3 of the structural

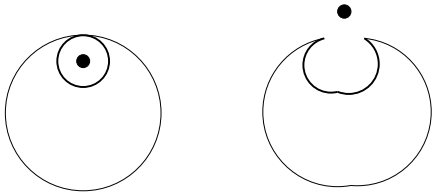


Figure 3.3: The effect of opening a looping sequence that is an element of another one.

congruence relation. The opposite default behavior could be obtained by replacing axiom A3 by $\left(T \,|\, T_1\right)^L \equiv \left(T\right)^L \rfloor\, T_1$.

Example 3.8. We describe the first few steps of the epidermal growth factor receptor (EGFR) signaling pathway to show the power of the structural congruence. The EGFR is a transmembrane protein that binds to an EGF protein on its extracellular domain, then forms a dimer with another EGFR protein in the same state, and then, after a phosphorylation, binds to a protein called ShC on its intracellular domain.

The system can be modeled as follows. We model the cell membrane as a looping sequence composed by R elements representing EGFR proteins. We denote with E an EGF protein, with RE a receptor bound to an EGF protein, and with $R2P$ the dimerization of the complex, assumed to be phosphorylated. Finally, we denote with ShC an ShC protein, and with $R2S$ the complex formed by $R2P$ and ShC. The rules describing the evolution of the system are the following:

$$\begin{aligned} R \,|\, E &\mapsto RE \\ RE \cdot X \cdot RE &\mapsto R2P \cdot X \\ \left(R2P \cdot X\right)^L \rfloor\, ShC &\mapsto \left(R2S \cdot X\right)^L \end{aligned}$$

The three rules describe the formation of the EGFR/EGF complex, the formation of the phosphorylated dimer, and its binding to the ShC protein, respectively. We model the initial state of the system with a few instances of each protein as the term

$$\left(R \cdot R \cdot R \cdot R \cdot R \cdot R\right)^L \rfloor\, (ShC \,|\, ShC) \,|\, E \,|\, E \,|\, E$$

and we show the following sequence of transitions as an example of possible evolution:

$$\begin{aligned} & \left(R \cdot R \cdot R \cdot R \cdot R \cdot R\right)^L \rfloor\, (ShC \,|\, ShC \,|\, ShC) \,|\, E \,|\, E \,|\, E \\ \equiv\ & \left(R \cdot R \cdot (R \,|\, E) \cdot R \cdot (R \,|\, E) \cdot R\right)^L \rfloor\, (ShC \,|\, ShC \,|\, ShC) \,|\, E \\ \to\ & \left(R \cdot R \cdot RE \cdot R \cdot (R \,|\, E) \cdot R\right)^L \rfloor\, (ShC \,|\, ShC \,|\, ShC) \,|\, E \\ \to\ & \left(R \cdot R \cdot RE \cdot R \cdot RE \cdot R\right)^L \rfloor\, (ShC \,|\, ShC \,|\, ShC) \,|\, E \\ \to\ & \left(R \cdot R \cdot R2 \cdot R \cdot R\right)^L \rfloor\, (ShC \,|\, ShC \,|\, ShC) \,|\, E \\ \equiv\ & \left(R2 \cdot R \cdot R \cdot R \cdot R\right)^L \rfloor\, (ShC \,|\, ShC \,|\, ShC) \,|\, E \end{aligned}$$

$$\rightarrow \ \left(R2S \cdot R \cdot R \cdot R \cdot R\right)^{L} \rfloor \left(ShC \mid ShC\right) \mid E$$
$$\rightarrow \ \ldots$$

In this sequence of transitions the structural congruence relation has been applied twice. The first time it has been used to permit the application of the first rewrite rule when R is an element of a looping sequence and E is outside the looping sequence. The second time it has been used to rotate the looping sequence and hence to allow the application of the third rewrite rule. A powerful structural congruence relation allows defining simpler rewrite rules.

To conclude the presentation of Full–CLS, we give a result on its expressiveness.

Theorem 3.9 (Turing Completeness)**.** *The class of Full–CLS models is Turing complete.*

Proof. We adapt the proof for rewrite systems in [25] to Full–CLS. Turing machines can be simulated by Full–CLS models. Each state symbol q and tape symbol $a, b, \ldots$ of the machine will be a symbol in the alphabet $\mathcal{E}$ of the Full–CLS model. The tape of the machine will be represented by a sequence $l \cdot a_1 \cdot \cdots \cdot a_{i-1} \cdot h \cdot a_i \cdot \cdots \cdot a_n \cdot r$, with $l, r, h \in \mathcal{E}$. In this sequence, l and r denote the left and right ends of the tape, and h the position of the read head. The symbol that is being scanned is a_i, and the left portion of the tape cannot be blank. The state q of the machine will be represented by the sequence $s \cdot q$ with $s \in \mathcal{E}$. We assume l, r, h and s to differ from any state symbol and tape symbol of the machine.

A transition of the machine will be encoded into a sequence of one of the following forms:

1. $t \cdot b \cdot q \cdot a \cdot s' \cdot q' \cdot b \cdot a' \cdot t$
2. $t \cdot l \cdot q \cdot a \cdot l \cdot s' \cdot q' \cdot \# \cdot a' \cdot t$
3. $t \cdot b \cdot q \cdot a \cdot b \cdot a' \cdot s' \cdot q' \cdot t$
4. $t \cdot b \cdot q \cdot r \cdot b \cdot a' \cdot s' \cdot q' \cdot r \cdot t$

where $\#$ is the blank symbol of the machine, and $t, s' \in \mathcal{E}$ are assumed to differ from any state symbol and tape symbol of the machine.

Symbol t is used to specify that the sequence describes a transition of the machine. In all the four forms of transitions we have that the three symbols that follow the first t in the sequence represent the configuration of the machine in which the transition can occur. In particular: in 1 and 3, $b \cdot q \cdot a$ denotes a machine in state q in which the symbol being scanned is a and the symbol immediately to the left of a is b; in 2, $l \cdot q \cdot a$ denotes a machine in state q in which the symbol being scanned is a and it is at the left end of the tape; in 4, $b \cdot q \cdot r$ denotes a machine in state q in which the read head is at the right end of the tape and the last symbol of the tape was b. In all the four forms of transitions, the rest of the sequence denotes how the configuration of the machine changes after the occurrence of the transition. The symbol s' is used in transition to mark the position where the read head will be placed after the occurrence of the transition.

Now, for each left–moving instruction of the form "if in state q reading a, write a', move left, and go into state q'", in the CLS term there must be sequences of the form

$$t \cdot b \cdot q \cdot a \cdot s' \cdot q' \cdot b \cdot a' \cdot t$$

for every tape symbol b, as well as an extra sequence of the form

$$t \cdot l \cdot q \cdot a \cdot l \cdot s' \cdot q' \cdot \# \cdot a' \cdot t$$

to handle the left end of the tape. For each right–moving instruction of the form "if in state q reading a, write a', move right, and go into state q'", there must be sequences of the form

$$t \cdot b \cdot q \cdot a \cdot b \cdot a' \cdot s' \cdot q' \cdot t$$

for every type symbol b, as well as an extra sequence of the form

$$t \cdot b \cdot q \cdot r \cdot b \cdot a' \cdot s' \cdot q' \cdot r \cdot t$$

when the symbol being scanned is #, to handle the right end of the tape. The parallel composition of all these transition sequences, together with a sequence $l \cdot a_1 \cdot \cdots \cdot a_{i-1} \cdot h \cdot a_i \cdot \cdots \cdot a_n \cdot r$, and with a sequence $s \cdot q$, is the Full–CLS term corresponding to a machine in state q with tape $a_1 \cdots a_n$ in which the tape symbol being scanned is a_i. Summing up, such a term is the following:

$$s \cdot q \,|\, l \cdot a_1 \cdot \cdots \cdot a_{i-1} \cdot h \cdot a_i \cdot \cdots \cdot a_n \cdot r \,|\, t \cdots t \,|\, \ldots \,|\, t \cdots t$$

Finally, the set of rewrite rules that must be included in the model contains only the following rule, and it is the same for all machines.

$$Y \cdot h \cdot Y' \,|\, s' \cdot X \,|\, t \cdot Y \cdot X \cdot Y' \cdot Z \cdot s' \cdot X' \cdot Z' \cdot t \;\mapsto\; Z \cdot h \cdot Z' \,|\, s' \cdot X' \,|\, t \cdot Y \cdot X \cdot Y' \cdot Z \cdot s' \cdot X' \cdot Z' \cdot t$$

$$[X, X', Y, Y' \in \mathcal{E}].$$

□

3.2 Bacteria Sporulation and Bacteriophage Viruses in Full–CLS

In this section we show how Full–CLS can be used to describe some aspects of the reproduction of bacteria and of bacteriophage viruses. For the sake of our study we can assume that a bacterium consists of a cellular membrane containing its DNA. In particular, as regards bacteria reproduction, we consider the sporulation mechanism, which allows producing inactive and very resistant forms, called spores. A spore can germinate and then produce a new bacterium.

Schematically, the sporulation process (shown in Fig. 3.4) proceeds as follows:

1. the DNA inside the bacterium is duplicated (duplication);
2. inside the bacterium a new membrane is formed containing the copy of the DNA (prespore);
3. around the prespore a second membrane layer is formed (coat);
4. eventually, the spore passes through the bacterium membrane and becomes a free spore (release).

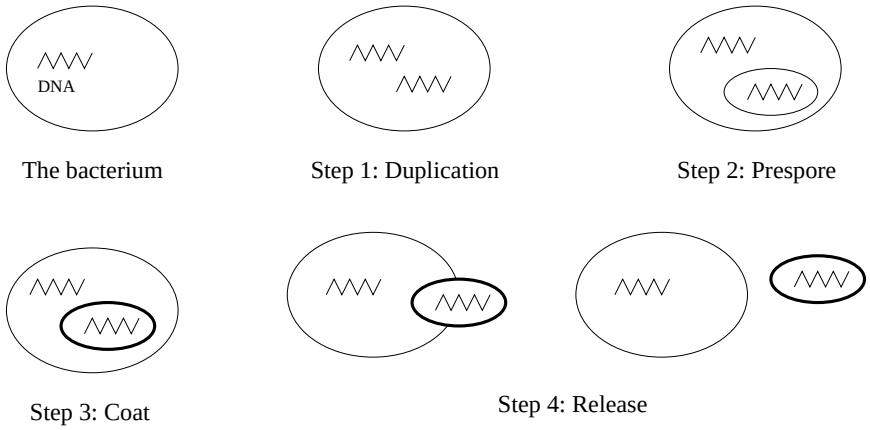


Figure 3.4: The Sporulation Process

For the sake of clarity, before giving the rules for the process, let us introduce some denotations for terms which occur very often:

$$PRESPORE \ ::= \ \big(m\big)^L \rfloor DNA_b$$

$$SPORE_1 \ ::= \ \big(c\big)^L \rfloor PRESPORE \qquad SPORE_2 \ ::= \ \big(d\big)^L \rfloor PRESPORE$$

Now, the rewrite rules for describing the steps of the process are the following:

$S1.$ $\big(m \cdot m\big)^L \rfloor (DNA_b \,|\, X) \ \mapsto \ \big(m \cdot m\big)^L \rfloor (DNA_b \,|\, DNA_b \,|\, X) \qquad [occ(DNA_b, X) = 0]$

$S2.$ $\big(m \cdot m\big)^L \rfloor (DNA_b \,|\, DNA_b \,|\, X) \ \mapsto \ \big(m \cdot m\big)^L \rfloor (DNA_b \,|\, PRESPORE \,|\, X)$

$S3.$ $\big(m \cdot m\big)^L \rfloor (X \,|\, PRESPORE \,|\, Y) \ \mapsto \ \big(m \cdot m\big)^L \rfloor (X \,|\, SPORE_1 \,|\, Y)$

$S4.$ $\big(m \cdot m\big)^L \rfloor (X \,|\, SPORE_1 \,|\, Y) \ \mapsto \ \big(SPORE_1 \cdot m \cdot m\big)^L \rfloor (X \,|\, Y)$

$S5.$ $\big(SPORE_1 \cdot m \cdot m\big)^L \rfloor X \ \mapsto \ (\big(m \cdot m\big)^L \rfloor X) \,|\, SPORE_2$

$S6.$ $SPORE_2 \ \mapsto \ d \,|\, \big(m \cdot m\big)^L \rfloor DNA_b$

Rule S1 describes DNA duplication inside a bacterium (step 1 of the process). The bacterium membrane is represented by a looping of two membrane elements m; element DNA_b represents the bacterium DNA and the term variable X represents any other element inside the bacterium membrane. The condition that DNA_b does not appear in the term X means that a sporulation process must terminate before starting a second one (no more than one copy of DNA inside the bacterium at one time).

Rule S2 models the forming of a prespore (step 2). Conventionally, we assume that the number of membrane elements of a prespore is one, hence the size of a prespore is roughly a half the size of a bacterium.

Rule S3 models the forming of the spore coat (step 3), where c represents the elements of the outer coat. The double layer of the spore is represented by two looping terms, one inside the other:

$$\big(c\big)^L \rfloor (\big(m\big)^L \rfloor DNA_b).$$

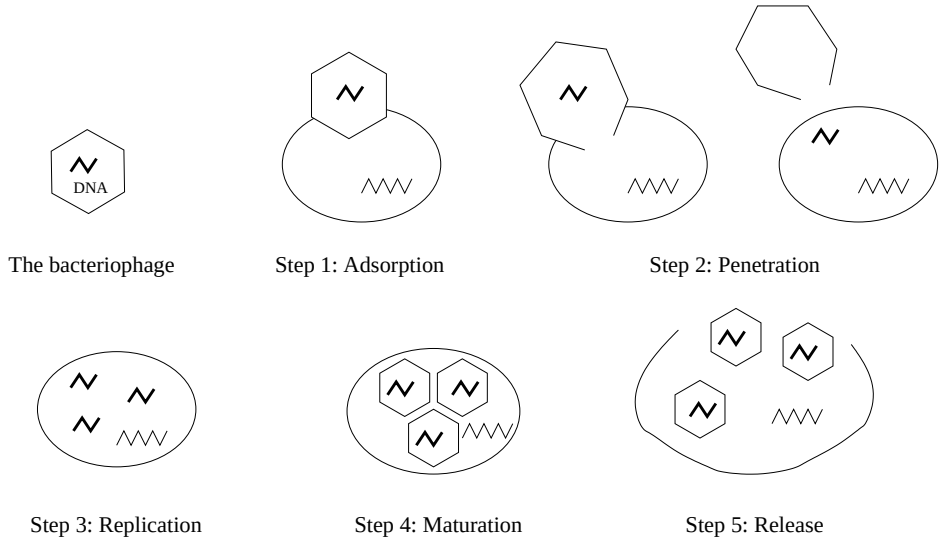


Figure 3.5: The Bacteriophage Replication Process

Rules S4 and S5 model the exiting of the spore from the bacterium (step 4). In a first phase (rule S4) the spore adheres to the bacterium membrane, becoming one element of the looping representing it. Note that the spore is represented in the rule as first element of the looping, but it can be shifted to any position by using the congruence rules. In a second phase (rule S5) the spore becomes free. In this phase, in order to distinguish a free spore from a spore inside the bacterium, the outer coat of the spore changes its elements from c to d.

A free spore may germinate by loosing its coat, which becomes an open membrane, and by growing to a normal size of two membrane elements (rule S6).

Bacteriophage viruses (or phages) exploit the enzymes of the bacteria for duplicating their DNA. In particular, they behave according to the following pattern (depicted in Figure 3.5):

1. the phage joins with the bacterium membrane (adsorption);

2. the phage releases its DNA inside the bacterium (penetration);

3. the DNA of the phage replicates itself using bacterium enzymes (replication);

4. each copy of the phage DNA forms a new phage inside the bacterium membrane (maturation);

5. when the number of new phages inside the bacterium reaches a certain number, the membrane breaks and the new phages become free (release).

As before, we introduce a denotation for a term which occurs quite often:

$$VIRUS \ ::= \ \big(v\big)^L \rfloor DNA_v$$

The rewrite rules for describing the steps of the process are the following:

$$V1.\quad VIRUS \,|\, \big(m \cdot m\big)^L \rfloor X \;\mapsto\; \big(VIRUS \cdot m \cdot m\big)^L \rfloor X$$

$$V2.\quad \big(VIRUS \cdot m \cdot m\big)^L \rfloor X \;\mapsto\; \big(m \cdot m\big)^L \rfloor (X \,|\, DNA_v) \,|\, v$$

$$V3.\quad \big(m \cdot m\big)^L \rfloor (X \,|\, DNA_v) \;\mapsto\; \big(m \cdot m\big)^L \rfloor (X \,|\, DNA_v \,|\, DNA_v) \qquad [occ(DNA_v, X) < max - 1]$$

$$V4.\quad \big(m \cdot m\big)^L \rfloor (X \,|\, DNA_v) \;\mapsto\; \big(m \cdot m\big)^L \rfloor (X \,|\, VIRUS)$$

$$V5.\quad \big(m \cdot m\big)^L \rfloor X \;\mapsto\; m \cdot m \,|\, X \qquad [occ(VIRUS, X) > max - s]$$

Rule V1 describes the joining of phage with the bacterium membrane (step 1 of the process). The phage membrane is represented by a looping of one element v; DNA_v represents the phage DNA. The application of the rule causes the phage to become part of the bacterium membrane. Namely, the looping representing the phage becomes an element of the looping representing the bacterium membrane.

Rule V2 models the releasing of phage DNA inside the bacterium. The phage membrane becomes a free open membrane (step 2).

Rule V3 describes the replication of phage DNA inside the bacterium (step 3). We assume that the replication happens only if the occurrences of DNA_v inside the bacterium are less than a number max.

Rule V4 describes the formation of a membrane around a phage DNA inside the bacterium (step 4).

Rule V5 models the breaking of the bacterium membrane when the number of phages inside it reaches a value close enough to max (the distance is less then a value $s > 0$). The bacterium membrane becomes a free open membrane, and everything contained in it (variable X) is released (step 5).

Note that we have assumed that bacteria and phages cannot die a natural death. In particular, bacteria can die only if parasitized by viruses, and viruses die only when inoculating their DNA inside the bacterium.

Given a Full–CLS model of a biological system, it is possible to verify properties of reachability of particular states by computing all the possible evolutions of the model. A model checker would allow verifying these kinds of properties automatically, under the condition that the transition system representing all the possible evolutions has a small or simple state space. This condition is often not satisfied, but usually one can simplify the verification by performing approximations. The easiest of such approximations is verifying bounded reachability (reachability after a limited number of transition) of the states of interest.

Example 3.10. Assume $max = 2$ and $s = 0$, namely that no replication of DNA_v can occur in a bacterium already containing two or more copies of DNA_v, and that the bacterium membrane can break when at least two viruses are inside. Consider the initial configuration in which there is one bacterium and three phages. This is represented by the term:

$$\big(\big(m \cdot m\big)^L \rfloor DNA_b\big) \,|\, VIRUS \,|\, VIRUS \,|\, VIRUS.$$

We can prove that, in a possible evolution, we can reach the configuration:

$$\big(m \cdot m\big)^L \rfloor (DNA_b \,|\, DNA_v \,|\, DNA_v \,|\, DNA_v \,|\, DNA_v) \,|\, v \,|\, v \,|\, v.$$

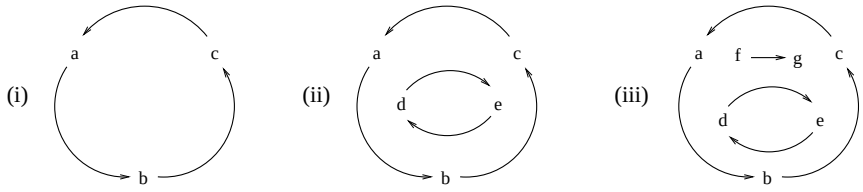


Figure 3.6: (i) represents $\big(a \cdot b \cdot c\big)^L$; (ii) represents $\big(a \cdot b \cdot c\big)^L \rfloor \big(d \cdot e\big)^L$; (iii) represents $\big(a \cdot b \cdot c\big)^L \rfloor \big(\big(d \cdot e\big)^L \mid f \cdot g\big)$.

The configuration represents a situation in which the bacterium contains a number of copies of virus DNA greater than max.

Actually, the steps to reach the configuration are the following: one virus infects the bacterium and its DNA is replicated inside the bacterium membrane (by application of rules V1, V2 and V3, in the order). Then the other two phages infect the bacterium (rule V1) and inoculate their DNA in it (rule V2).

3.3 Definition of CLS

In Section 3.1 we have seen that the structural congruence relation of Full–CLS is quite complex, because it has to handle some ambiguities that may arise in terms. These ambiguities are caused by the combined use of an operator representing juxtaposition (which implies disconnectedness) and of another one representing physical connection (such as sequencing). Another cause of ambiguity is the non–combined use of the looping operator and of containment.

In this section we introduce the Calculus of Looping Sequences (CLS). Its main difference with respect to Full–CLS is that it assumes restrictions on the syntax of terms aiming at avoiding ambiguities. In particular, in CLS terms we have that sequences can be composed only by elements of the alphabet $\mathcal{E}$, and the containment operator can be applied only to looping sequences. The alphabet $\mathcal{E}$ and the neutral term ϵ are assumed as in Full–CLS.

Definition 3.11 (Terms)**.** Terms T *and* Sequences S *of CLS are given by the following grammar:*

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

where a is a generic element of $\mathcal{E}$. We denote with $\mathcal{T}$ the infinite set of terms, and with $\mathcal{S}$ the infinite set of sequences.

As in Full–CLS, we have a sequencing operator $\_ \cdot \_$, a looping operator $\big(\_\big)^L$, a parallel composition operator $\_ \mid \_$, and a containment operator $\_ \rfloor \_$. Sequencing can be used only to compose elements of the alphabet $\mathcal{E}$, as it is used in an independent syntactic category S. A term can be a sequence, or a looping sequence containing a term, or the parallel

composition of two terms. By the definition of terms, we have that looping and containment are always applied together, hence we can consider them as a single binary operator $\left(\_\right)^L \rfloor \_$ which applies to one sequence and one term.

Brackets can be used to indicate the order of application of the operators, and we assume $\left(\_\right)^L \rfloor \_$ to have the precedence over $\_ \mid \_$. In Figure 3.6 we show some examples of CLS terms and their visual representation.

The constraints on the syntax imposed in CLS simplify the definition of the structural congruence relation. Since we have different syntactic categories, we define two different relations, one on sequences and one on terms.

Definition 3.12 (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:*

$$S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3 \qquad S \cdot \epsilon \equiv_S \epsilon \cdot S \equiv_S S$$

$$S_1 \equiv_S S_2 \text{ implies } S_1 \equiv_T S_2 \text{ and } \left(S_1\right)^L \rfloor T \equiv_T \left(S_2\right)^L \rfloor T$$
$$T_1 \mid T_2 \equiv_T T_2 \mid T_1 \qquad T_1 \mid (T_2 \mid T_3) \equiv_T (T_1 \mid T_2) \mid T_3 \qquad T \mid \epsilon \equiv_T T$$
$$\left(\epsilon\right)^L \rfloor \epsilon \equiv \epsilon \qquad \left(S_1 \cdot S_2\right)^L \rfloor T \equiv_T \left(S_2 \cdot S_1\right)^L \rfloor T$$

Rules of the structural congruence state the associativity of $\cdot$ and $\mid$, the commutativity of the latter and the neutral role of ϵ. Moreover, axiom $\left(S_1 \cdot S_2\right)^L \rfloor T \equiv_T \left(S_2 \cdot S_1\right)^L \rfloor T$ says that elementary sequences in a looping can rotate. We remark that, differently from Full–CLS, we have $\left(\epsilon\right)^L \rfloor T \not\equiv T$ if $T \not\equiv \epsilon$, hence $\left(\epsilon\right)^L$ does not play a neutral role if it is not empty. In the following, for simplicity, we will use $\equiv$ in place of $\equiv_T$.

Patterns in CLS include three different types of variables: two 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 TV ranged over by $X, Y, Z, \ldots$, a set of sequence variables 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, $\mathcal{V} = TV \cup SV \cup \mathcal{X}$.

Definition 3.13 (Patterns)**.** Patterns P *and* sequence patterns SP *of CLS are given by the following grammar:*

$$\begin{array}{rcl} P & ::= & SP \mid \left(SP\right)^L \rfloor P \mid P \mid P \mid X \\ SP & ::= & \epsilon \mid a \mid SP \cdot SP \mid \widetilde{x} \mid x \end{array}$$

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

We assume the structural congruence relation to be trivially extended to patterns. As in Full–CLS, an *instantiation* is a partial function $\sigma : \mathcal{V} \to \mathcal{T}$. An instantiation must preserve the type of variables, thus for $X \in TV, \widetilde{x} \in SV$ and $x \in \mathcal{X}$ we have $\sigma(X) \in \mathcal{T}, \sigma(\widetilde{x}) \in \mathcal{S}$ and $\sigma(x) \in \mathcal{E}$, respectively. Given $P \in \mathcal{P}$, with $P\sigma$ we denote the term obtained by replacing each occurrence of each variable $X \in \mathcal{V}$ appearing in P with the corresponding term $\sigma(X)$. With Σ we denote the set of all the possible instantiations and, given $P \in \mathcal{P}$, with $Var(P)$ we denote the set of variables appearing in P.

Now we define rewrite rules. For the sake of simplicity, differently from Full–CLS we do not allow application conditions to be included in rules. However, an extension of CLS with these conditions could be defined without problems by following the Full–CLS approach.

Definition 3.14 (Rewrite Rules)**.** *A rewrite rule is a pair of patterns* (P_1, P_2)*, denoted with* $P_1 \mapsto P_2$*, where* $P_1, P_2 \in PP$*,* $P_1 \not\equiv \epsilon$ *and such that* $Var(P_2) \subseteq Var(P_1)$*. We denote with* $\Re$ *the infinite set of all the possible rewrite rules. We say that a rewrite rule is* ground *if* $Var(P_1) = Var(P_2) = \varnothing$*, and a set of rewrite rules* $\mathcal{R} \in Re$ *is* ground *if all the rewrite rules it contains are ground.*

A rewrite rule (P_1, P_2) states that a term $P_1\sigma$, obtained by instantiating variables in P_1 by some instantiation function σ, can be transformed into the ground term $P_2\sigma$. Rule application is the mechanism of evolution of CLS terms. We define the semantics of CLS as a transition system, in which states corresponds to terms, and transitions corresponds to rule applications.

Definition 3.15 (Semantics)**.** *Given a set of rewrite rules* $\mathcal{R} \subseteq \Re$*, the* semantics *of CLS is the least transition relation* $\rightarrow$ *on terms closed under* $\equiv$*, and satisfying the following inference rules:*

$$\frac{(P_1, P_2) \in \mathcal{R} \qquad P_1\sigma \not\equiv \epsilon \qquad \sigma \in \Sigma}{P_1\sigma \rightarrow P_2\sigma} \qquad \frac{T_1 \rightarrow T_2}{T \,|\, T_1 \rightarrow T \,|\, T_2} \qquad \frac{T_1 \rightarrow T_2}{(S)^L \rfloor T_1 \rightarrow (S)^L \rfloor T_2}$$

where the symmetric rule for the parallel composition is omitted.

A *model* in CLS is given by a term describing the initial state of the modeled system and by a set of rewrite rules describing all the possible events that may occur in the system.

Finally, as for Full–CLS, Turing–completeness holds for CLS.

Theorem 3.16 (Turing Completeness)**.** *The class of CLS models is Turing complete.*

Proof. The proof is essentially the same as the one of Theorem 3.9, but with a difference in the forms of the sequences representing transitions of the simulated Turing machine, and in the rewrite rule that allow the CLS model to evolve. We show here only the differences with respect to the other proof. A transition of the machine will be encoded into a sequence of one of the following forms:

1. $t \cdot b \cdot q \cdot a \cdot s \cdot q' \cdot b \cdot a'$
2. $t \cdot l \cdot q \cdot a \cdot l \cdot s \cdot q' \cdot \# \cdot a'$
3. $t \cdot b \cdot q \cdot a \cdot b \cdot a' \cdot s \cdot q'$
4. $t \cdot b \cdot q \cdot r \cdot b \cdot a' \cdot s \cdot q' \cdot r$

where # is the blank symbol of the machine, and $t \in \mathcal{E}$ is assumed to differ from any state symbol and tape symbol of the machine.

Differently from the proof of Theorem 3.9, in a sequence describing a transition we do not need the t symbol at the right end, and we can reuse symbol s instead of introducing symbol s'.

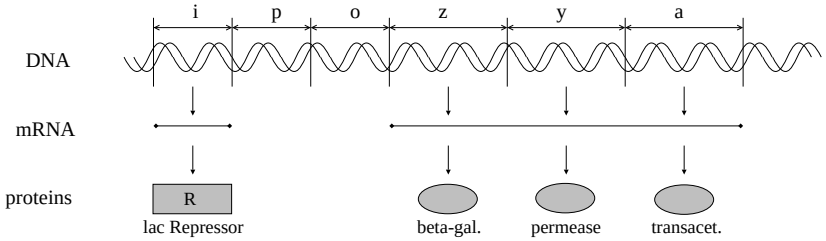


Figure 3.7: The lactose operon.

Now, for each left–moving instruction of the form "if in state q reading a, write a', move left, and go into state q'", in the CLS term there must be sequences of the form

$$t \cdot b \cdot q \cdot a \cdot s \cdot q' \cdot b \cdot a'$$

for every tape symbol b, as well as an extra sequence of the form

$$t \cdot l \cdot q \cdot a \cdot l \cdot s \cdot q' \cdot \# \cdot a'$$

to handle the left end of the tape. For each right–moving instruction of the form "if in state q reading a, write a', move right, and go into state q'", there must be sequences of the form

$$t \cdot b \cdot q \cdot a \cdot b \cdot a' \cdot s \cdot q'$$

for every type symbol b, as well as an extra sequence of the form

$$t \cdot b \cdot q \cdot r \cdot b \cdot a' \cdot s \cdot q' \cdot r$$

when the symbol being scanned is #, to handle the right end of the tape.

A machine in state q with tape $a_1 \cdots a_n$ in which the tape symbol being scanned is a_i, is encoded into the following term:

$$s \cdot q \,|\, l \cdot a_1 \cdot \cdots \cdot a_{i-1} \cdot h \cdot a_i \cdot \cdots \cdot a_n \cdot r \,|\, t \cdot \cdots \,|\, \ldots \,|\, t \cdot \cdots$$

and the set of rewrite rule that must be included in the model contains only the following rule:

$$s \cdot x \,|\, \widetilde{y} \cdot y \cdot h \cdot z \cdot \widetilde{z} \,|\, t \cdot y \cdot x \cdot z \cdot \widetilde{u} \cdot s \cdot k \cdot \widetilde{w} \;\mapsto\; s \cdot k \,|\, \widetilde{y} \cdot \widetilde{u} \cdot h \cdot \widetilde{w} \cdot \widetilde{z} \,|\, t \cdot y \cdot x \cdot z \cdot \widetilde{u} \cdot s \cdot k \cdot \widetilde{w}$$

where $x, y, z, k \in \mathcal{X}$ and $\widetilde{y}, \widetilde{u}, \widetilde{w}, \widetilde{z} \in SV$. □

3.4 Modeling Gene Regulation in E.Coli with CLS

In this section we develop a CLS model of the regulation process of the lactose operon in E. coli (Escherichia coli). E. coli is a bacterium often present in the intestine of many animals. As most bacteria, it is often exposed to a constantly changing physical and chemical environment, and reacts to changes in its environment through changes in the kinds of proteins it produces.

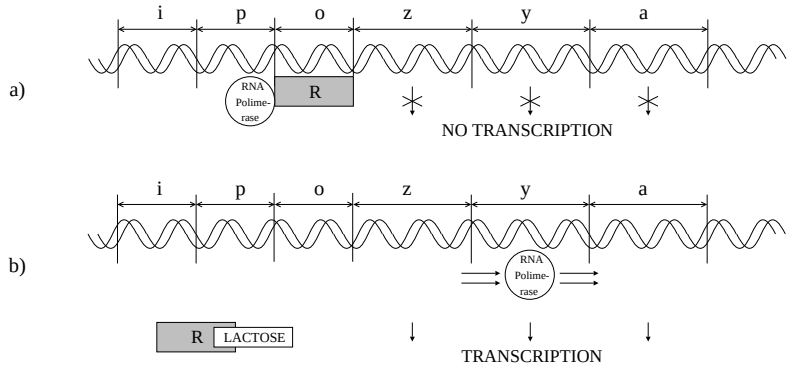


Figure 3.8: The regulation process. In the absence of lactose (case a) the lac Repressor binds to gene o and precludes the RNA polymerase from transcribing genes z,y and a. When lactose is present (case b) it binds to and inactivates the lac Repressor.

In general, 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 phenomenon is called *enzyme induction* or, more generally, *gene regulation* since it is obtained by controlling the transcription of some genes into the corresponding proteins.

Let us consider the lactose degradation example in E. coli. Two enzymes are required to start the breaking process: the *lactose permease*, which is incorporated in the membrane of the bacterium and actively transports the sugar into the cell (without this enzyme lactose can enter the bacterium anyway, but much more slowly), and the *beta galactosidase*, which splits lactose into glucose and galactose. The bacterium produces also the *transacetylase* enzyme, whose function is marginal.

The sequence of genes in the DNA of E. coli which produces the described enzymes, is known as the *lactose operon* (see Fig. 3.7). It is composed by six genes: the first three (i, p, o) regulate the production of the enzymes, and the last three (z, y, 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 Fig. 3.8): 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 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 any more the activity of the RNA polymerase. In this case transcription is performed and the three enzymes for lactose degradation are synthesized.

Now we describe how to model the gene regulation process with CLS. For the sake of simplicity we give a partial model, in the sense that we describe how the transcription of the structural genes is activated when the lactose is in the environment, but we do not describe how the transcription of such genes is stopped when the lactose disappears. Moreover, in

order to simplify the example, we assume that genes are transcribed directly into proteins (thus avoiding the modeling of the mRNA), that the lac Repressor is transcribed from gene i without the need of the RNA polymerase and that it can be produced only once. Finally, we assume that one RNA polymerase is present inside the bacterium.

We model the membrane of the bacterium as the looping sequence $(m)^L$, where the elementary constituent 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 element corresponds to a gene, and we replace $lacO$ with RO in the sequence when the lac Repressor is bound to gene o. When the lac Repressor is unbound, it is modeled by the elementary constituent $repr$. Finally, we model the RNA polymerase as the elementary constituent $polym$, a molecule of lactose as the elementary constituent $LACT$, and beta galactose, lactose permease and transacetylase enzymes as elementary constituents $betagal, perm$ and $transac$, respectively.

When no lactose is present the bacterium is modeled by the following term:

$$Ecoli \ ::= \ (m)^L \rfloor (lacI \cdot lacP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA \,|\, polym)$$

The transcription of the DNA is modeled by the following set of rules:

$$\begin{aligned}
lacI \cdot \widetilde{x} &\mapsto lacI' \cdot \widetilde{x} \,|\, repr && \text{(R1)}\\
polym \,|\, \widetilde{x} \cdot lacP \cdot \widetilde{y} &\mapsto \widetilde{x} \cdot PP \cdot \widetilde{y} && \text{(R2)}\\
\widetilde{x} \cdot PP \cdot lacO \cdot \widetilde{y} &\mapsto \widetilde{x} \cdot lacP \cdot PO \cdot \widetilde{y} && \text{(R3)}\\
\widetilde{x} \cdot PO \cdot lacZ \cdot \widetilde{y} &\mapsto \widetilde{x} \cdot lacO \cdot PZ \cdot \widetilde{y} && \text{(R4)}\\
\widetilde{x} \cdot PZ \cdot lacY \cdot \widetilde{y} &\mapsto \widetilde{x} \cdot lacZ \cdot PY \cdot \widetilde{y} \,|\, betagal && \text{(R5)}\\
\widetilde{x} \cdot PY \cdot lacA &\mapsto \widetilde{x} \cdot lacY \cdot PA \,|\, perm && \text{(R6)}\\
\widetilde{x} \cdot PA &\mapsto \widetilde{x} \cdot A \,|\, transac \,|\, polym && \text{(R7)}
\end{aligned}$$

Rule (R1) describes the transcription of gene i into the lac Repressor. After transcription $lacI$ becomes $lacI'$ to avoid further productions of the lac Repressor. Rule (R2) describes the binding of the RNA polymerase to gene p. We denote the complex formed by the binding RNA polymerase to a gene $lac\_$ with the elementary constituent $P\_$. Rules (R3)–(R6) describe the scanning of the DNA performed by the RNA polymerase and the consequent production of enzymes. Rule (R3) can be applied (and the scanning can be performed) only when the sequence contains $lacO$ instead of RO, that is when the lac Repressor is not bound to gene o. Finally, in rule (R7) the RNA polymerase terminates the scanning and releases the sequence.

The following rules describe the binding of the lac Repressor to gene o, and what happens when lactose is present in the environment of the bacterium:

$$\begin{aligned}
repr \,|\, \widetilde{x} \cdot lacO \cdot \widetilde{y} &\mapsto \widetilde{x} \cdot RO \cdot \widetilde{y} && \text{(R8)}\\
LACT \,|\, (m \cdot \widetilde{x})^L \rfloor X &\mapsto (m \cdot \widetilde{x})^L \rfloor (X \,|\, LACT) && \text{(R9)}\\
\widetilde{x} \cdot RO \cdot \widetilde{y} \,|\, LACT &\mapsto \widetilde{x} \cdot lacO \cdot \widetilde{y} \,|\, RLACT && \text{(R10)}
\end{aligned}$$

Rule (R8) describes the binding of the lac Repressor to gene o, rule (R9) models the passage of the lactose through the membrane of the bacterium and rule (R10) the removal of the lac Repressor from gene o operated by the lactose. In this rule the elementary constituent $RLACT$ denotes the binding of the lactose to the lac Repressor.

Finally, the following rules describe the behavior of the enzymes synthesized when lactose is present, and their degradation:

$$\left(\widetilde{x}\right)^L \rfloor (perm \,|\, X) \;\mapsto\; \left(perm \cdot \widetilde{x}\right)^L \rfloor X \qquad \text{(R11)}$$

$$LACT \,|\, \left(perm \cdot \widetilde{x}\right)^L \rfloor X \;\mapsto\; \left(perm \cdot \widetilde{x}\right)^L \rfloor (LACT \,|\, X) \qquad \text{(R12)}$$

$$betagal \,|\, LACT \;\mapsto\; betagal \,|\, GLU \,|\, GAL \qquad \text{(R13)}$$

$$perm \;\mapsto\; \epsilon \qquad \text{(R14)}$$

$$betagal \;\mapsto\; \epsilon \qquad \text{(R15)}$$

$$transac \;\mapsto\; \epsilon \qquad \text{(R16)}$$

Rule (R11) describes the incorporation of the lactose permease in the membrane of the bacterium, rule (R12) the transportation of lactose from the environment to the interior performed by the lactose permease, and rule (R13) the decomposition of the lactose into glucose (denoted GLU) and galactose (denoted GAL) performed by the beta galactosidase. Finally, rules (R14),(R15) and (R16) describe degradation of the lactose permease, the beta galactosidase and the transacetylase enzymes, respectively.

Let us denote the set of rewrite rules $\{(\text{R1}), \ldots, (\text{R16})\}$ as $\mathcal{R}_{lac}$, and the lactose operon $lacI' \cdot lacP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA$, after the production of the lac Repressor, as $lacI'$–A. An example of possible sequence of transitions which can be performed by the term $Ecoli$ by applying rules in $\mathcal{R}_{lac}$ when there are two molecules of lactose in the environment is the following (where $\rightarrow^*$ denotes a sequence of $\rightarrow$ steps):

$$\begin{aligned}
& Ecoli \,|\, LACT \,|\, LACT \\
\rightarrow^* \quad & \left(m\right)^L \rfloor (lacI' \cdot lacP \cdot RO \cdot lacZ \cdot lacY \cdot lacA \,|\, polym) \,|\, LACT \,|\, LACT \\
\rightarrow^* \quad & \left(m\right)^L \rfloor (lacI'\text{–}A|polym|RLACT) \,|\, LACT \\
\rightarrow^* \quad & \left(perm \cdot m\right)^L \rfloor (lacI'\text{–}A|betagal|transac|polym|RLACT) \,|\, LACT \\
\rightarrow^* \quad & \left(perm \cdot m\right)^L \rfloor (lacI'\text{–}A|betagal|transac|polym|RLACT|GLU|GAL)
\end{aligned}$$

In the example, by applying rules (R1) and (R8), $Ecoli$ produces the lac Repressor, which binds to gene o in the lactose operon. Then, the bacterium interacts with a molecule of lactose in the environment: by applying rule (R9) the lactose enters the membrane of the bacterium and by applying rule (R10) it binds to the lac Repressor. Then, a sequence of internal transitions are performed by applying rules (R2)–(R7) and (R11): the result is the transcription of the structural genes and the incorporation of the lactose permease in the membrane of the bacterium. Finally, the term interacts the other molecule of lactose in the environment, which enters the bacterium and is decomposed into GLU and GAL. The rules applied in this phase are (R12) and (R13).

Note that, if one starts from $Ecoli$, every time (R11) can be applied, also (R9) can be applied and the same results are obtained. Therefore, rule (R11) seems to be redundant. Nevertheless, rule (R11) describes a precise phenomenon, namely the action performed by the lactose permease, which is modeled by no other rule. The difference between rules (R9) and (R11) is that the latter describes a much faster event. However, since quantitative aspects are not considered in the calculus, the difference between the two rules does not appear.

3.5 Quasi–termination in CLS

By the semantics of CLS we have that, given a set of rewrite rules $\mathcal{R}$, a semantic model of a term T is a transition relation $\rightarrow$. Let us denote with $\rightarrow^*$ the reflexive and transitive closure of $\rightarrow$. We say that a term T' is *reachable* from T if $T \rightarrow^* T'$. We denote with $Reach(T, \mathcal{R})$ the set of all the terms reachable from T by applying rules in $\mathcal{R}$.

Reachability of particular terms (in general, reachability of particular states) is a property of interest in the study of many kinds of systems. In fact, it is often the case that, after building a model of a system, the modeler tests on it the non reachability of some error state in order to prove that the described system never fails. More complex properties than reachability can be verified by model checking. Properties on the possible evolutions of a system are described as logical formulas whose truth is tested by applying some verification algorithm. In order to use any of these verification techniques (reachability of states and model checking) it is preferable to have finite state semantic models. It is easy to see that in CLS this is not always the case: for instance, if we have $\mathcal{R} = \{a \mapsto a \,|\, b\}$ and $T = a$ we obtain $T = a \rightarrow a \,|\, b \rightarrow a \,|\, b \,|\, b \rightarrow \ldots$. Hence, it would be useful to be able to determine whether semantic models are finite or not for any initial term T by simply testing some property on the given set of rewrite rules $\mathcal{R}$.

The problem of finiteness of the set of reachable terms in CLS is similar to the problem of termination in term rewriting systems (see [25] for a survey on the topic). More precisely, it coincides with the problem of quasi–termination in such systems. Termination means that all evolutions of the system are finite. Quasi–termination, instead, does not forbid cyclic evolutions, namely infinite evolutions in which the same states are reached periodically. The set of reachable terms in a term rewriting system is finite if and only if such a system is quasi–terminating.

The termination and quasi–termination problems are, in general, undecidable. However, it has been proved that (i) a system is terminating if and only if there exists a well–founded monotonic ordering $\succ$ on terms such that the left–hand side of each rewrite rule is bigger (with respect to $\succ$) than the corresponding right–hand side; and (ii) a system is quasi–terminating if and only if there exists a well–founded monotonic quasi–ordering $\succsim$ on terms, whose equivalence relation $\approx$ admits only finite equivalence classes, and such that the left–hand side of each rewrite rule is bigger than or equivalent to (with respect to $\succsim$) the corresponding right–hand side. Note that, a monotonic ordering (quasi–ordering) is such that reducing a subterm reduces any superterm containing it. These properties are independent from the choice of the initial term.

Now we apply the same approach to CLS. The main differences between CLS and term rewriting systems we have to take into account are the presence of different syntactical categories for terms and sequences in CLS, and the closure with respect to the structural congruence $\equiv$ of the transition relation.

Theorem 3.17 (Finiteness of the Set of Reachable Terms)**.** *Given a set of CLS rules $\mathcal{R}$, if a well–founded monotonic quasi–ordering $\succsim$ on $\mathcal{T}$ exists whose equivalent relation $\approx$ admits only finite equivalence classes and such that:*

$$\forall T \mapsto T' \in \mathcal{R}.\, \forall \sigma \in \Sigma.\, T \succsim T'$$

then $\forall T \in \mathcal{T}.Reach(T, \mathcal{R})$ is finite.

Proof. We reduce this problem to the quasi–termination problem in term rewriting systems (TRSs), by showing how terms and rewrite rules of CLS can be translated into a term rewriting system (see [75] for an introduction to TRSs). For the sake of simplicity, without loss of generality, we omit the looping operator $\left(\cdot\right)^L$. Moreover, we assume the both the left– and right–hand sides of each rule in $\mathcal{R}$ are minimal terms. We consider a set of function symbols $\mathcal{F}$ composed by the constant ϵ, by one constant symbol for each element in $\mathcal{E}$, by the binary symbols $|, \rfloor, \cdot$, and by the unary symbol $\mathtt{t}$. Moreover, we allow variables in $\mathcal{V}$ to occur in TRS rules.
We define the encoding of CLS terms into TRS terms as $\{\!|T|\!\} = \mathtt{t}([\![T]\!])$, where the auxiliary encoding $[\![\cdot]\!]$ is recursively defined as follows:

$$\begin{array}{lll} [\![\epsilon]\!] = \epsilon & [\![x]\!] = x \quad \forall x \in \mathcal{V} & [\![T_1 \mid T_2]\!] = |\,(\{\!|T_1|\!\}, \{\!|T_1|\!\}) \\ [\![a]\!] = a \quad \forall a \in \mathcal{E} & [\![S_1 \cdot S_2]\!] = \cdot\,([\![S_1]\!], [\![S_1]\!]) & [\![S \rfloor T]\!] = \rfloor\,([\![S]\!], \{\!|T|\!\}) \end{array}$$

We extend the encoding $\{\!|\cdot|\!\}$ to sets of rewrite rules in the obvious way. Note that the symbol $\mathtt{t}$ is used to take into account the syntactical constraints of CLS. Moreover, we translate the structural congruence relation $\equiv$ into the following set of TRS rules:

$$\begin{array}{rl} \mathcal{R}_{\equiv} = \{ & \cdot\,(\cdot(x,y),z) \to \cdot(x,\cdot(y,z)), \quad \cdot(x,\cdot(y,z)) \to \cdot(\cdot(x,y),z), \\ & \cdot\,(x,\epsilon) \to \cdot(\epsilon,x), \quad \cdot(\epsilon,x) \to \cdot(x,\epsilon), \quad \cdot(\epsilon,x) \to x, \\ & |(|(x,y),z) \to |(x,|(y,z)), \quad |(x,|(y,z)) \to |(|(x,y),z), \\ & |(x,y) = |(y,x), \quad |(\epsilon,x) \to x, \quad \rfloor(\cdot(x,y),z) \to \rfloor(\cdot(y,x),z)\} \end{array}$$

Now, given a set of CLS rules $\mathcal{R}$, the corresponding term rewriting system is $\{\!|\mathcal{R}|\!\} \cup \mathcal{R}_{\equiv}$. Let $\Rightarrow$ be the transition relation of term rewriting systems, and $\Rightarrow^*$ be its reflexive and transitive closure. It holds that $T \to^* T'$ if and only if $\{\!|T|\!\} \Rightarrow^* \{\!|T''|\!\}$, with $T' \equiv T''$, and that the conditions on $\mathcal{R}$ imposed by the theorem are satisfied if and only if the conditions for quasi–termination on the term rewriting systems $\{\!|\mathcal{R}|\!\} \cup \mathcal{R}_{\equiv}$ are satisfied. □

As for termination and quasi–termination in term rewriting systems, to prove the finiteness of the set of reachable terms in CLS it could be convenient to separate the quasi–ordering $\succsim$ into (i) a mapping from terms $\mathcal{P}$ to multivariate integer polynomials, and (ii) the standard relation $\geq$ on integers, as described in [25]. This would allow reducing the finiteness problem to the problem of solving a system of inequalities.

3.6 Definition of LCLS

A formalism for modeling protein interactions at the domain level was developed in the seminal paper by Danos and Laneve [23], and extended in [47]. This formalism allows expressing proteins by a node with a fixed number of domains; binding between domains allow complexating proteins. In this section we extend CLS to represent protein interaction at the domain level. Such an extension, called Calculus of Linked Looping Sequences (LCLS), is obtained by labelling elementary components of sequences. Two elements with the same label are considered to be linked.

To model a protein at the domain level in CLS it would be natural to use a sequence with one symbol for each domain. However, the binding between two domains of two different proteins, that is the linking between two elements of two different sequences, cannot be expressed in CLS. To represent this, we extend CLS by labels on basic symbols. If in a term two symbols appear having the same label, we intend that they represent

domains which are bound to each other. If in a term there is a symbol with a label and no other symbol with the same label, we intend that the term represents only a part of a system we model, and that the symbol will be linked to some other symbol in another part of the term representing the full model.

As membranes create compartments, elements inside a looping sequence cannot be linked to elements outside. Elements inside a membrane can be linked either to other elements inside the membrane or to elements of the membrane itself. An element can be linked at most to another element.

The syntax of terms of the Calculus of Linked Looping Sequences (LCLS) is defined as follows. We use as labels natural numbers.

Definition 3.18 (Terms)**.** Terms T *and* Sequences S *of LCLS are given by the following grammar:*

$$\begin{array}{rcl} T & ::= & S \mid \left(S\right)^L \rfloor T \mid T \,|\, T \\ S & ::= & \epsilon \mid a \mid a^n \mid S \cdot S \end{array}$$

where a is a generic element of $\mathcal{E}$, and n is a natural number. We denote with $\mathcal{T}$ the infinite set of terms, and with $\mathcal{S}$ the infinite set of sequences.

The structural congruence relation is the same as for CLS.

Definition 3.19 (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:*

$$S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3 \qquad S \cdot \epsilon \equiv_S \epsilon \cdot S \equiv_S S$$

$$S_1 \equiv_S S_2 \text{ implies } S_1 \equiv_T S_2 \text{ and } \left(S_1\right)^L \rfloor T \equiv_T \left(S_2\right)^L \rfloor T$$
$$T_1 \,|\, T_2 \equiv_T T_2 \,|\, T_1 \qquad T_1 \,|\, (T_2 \,|\, T_3) \equiv_T (T_1 \,|\, T_2) \,|\, T_3 \qquad T \,|\, \epsilon \equiv_T T$$
$$\left(\epsilon\right)^L \rfloor \epsilon \equiv \epsilon \qquad \left(S_1 \cdot S_2\right)^L \rfloor T \equiv_T \left(S_2 \cdot S_1\right)^L \rfloor T$$

Patterns of LCLS are similar to the ones of CLS, with the addition of the labels.

Definition 3.20 (Patterns)**.** Patterns P *and* sequence patterns SP *of LCLS are given by the following grammar:*

$$\begin{array}{rcl} P & ::= & SP \mid \left(SP\right)^L \rfloor P \mid P \,|\, P \mid X \\ SP & ::= & \epsilon \mid a \mid a^n \mid SP \cdot SP \mid \widetilde{x} \mid x \mid x^n \end{array}$$

where a is a generic element of $\mathcal{E}$, n is a natural number and $X, \widetilde{x}$ and x are generic elements of TV, SV and $\mathcal{X}$, respectively. We denote with $\mathcal{P}$ the infinite set of patterns.

Note that, as for CLS, a LCLS term is also a LCLS pattern; everything we will define for patterns will be immediately defined also for terms. Moreover, in what follows we will often use the notions of *compartment* and of *top–level compartment* of a pattern. A compartment is a subpattern that is the content of a looping sequence and in which the contents of inner looping sequences are not considered. The top–level compartment is the

portion of the pattern that is not inside any looping sequence. For instance, the top–level compartment of a pattern $P = a \mid \left(b\right)^L \rfloor\, c \mid \left(d\right)^L \rfloor\, (X \mid \left(e\right)^L \rfloor\, f)$ is $a \mid \left(b\right)^L \rfloor\, \epsilon \mid \left(d\right)^L \rfloor\, \epsilon$. Other compartments in P are c, $X \mid \left(e\right)^L \rfloor\, \epsilon$, and f.

An LCLS pattern is well–formed if and only if a label occurs no more than twice, and two occurrences of a label are always in the same compartment. The following type system will be used for deriving the well–formedness of patterns.

In each inference rule the conclusion has the form $(N, N') \models P$, where N and N' are sets of natural numbers with N the set of labels used twice and N' the set of labels used only once in the top–level compartment of P.

Definition 3.21 (Type System)**.** *The typing algorithm for LCLS patterns is defined by the following inference rules:*

$$1.\ \left(\varnothing, \varnothing\right) \models \epsilon \qquad 2.\ \left(\varnothing, \varnothing\right) \models a \qquad 3.\ \left(\varnothing, \{n\}\right) \models a^n$$

$$4.\ \left(\varnothing, \varnothing\right) \models x \qquad 5.\ \left(\varnothing, \{n\}\right) \models x^n \qquad 6.\ \left(\varnothing, \varnothing\right) \models \widetilde{x} \qquad 7.\ \left(\varnothing, \varnothing\right) \models X$$

$$8.\ \frac{\left(N_1, N_1'\right) \models SP_1 \ \ \left(N_2, N_2'\right) \models SP_2 \ \ N_1 \cap N_2 = N_1' \cap N_2 = N_1 \cap N_2' = \varnothing}{\left(N_1 \cup N_2 \cup (N_1' \cap N_2'), (N_1' \cup N_2') \setminus (N_1' \cap N_2')\right) \models SP_1 \cdot SP_2}$$

$$9.\ \frac{\left(N_1, N_1'\right) \models P_1 \ \ \left(N_2, N_2'\right) \models P_2 \ \ N_1 \cap N_2 = N_1' \cap N_2 = N_1 \cap N_2' = \varnothing}{\left(N_1 \cup N_2 \cup (N_1' \cap N_2'), (N_1' \cup N_2') \setminus (N_1' \cap N_2')\right) \models P_1 \mid P_2}$$

$$10.\ \frac{\left(N_1, N_1'\right) \models SP \ \ \left(N_2, N_2'\right) \models P \ \ N_1 \cap N_2 = N_1' \cap N_2 = N_1 \cap N_2' = \varnothing \ \ N_2' \subseteq N_1'}{\left(N_1 \cup N_2', N_1' \setminus N_2'\right) \models \left(SP\right)^L \rfloor\, P}$$

where a is a generic element of $\mathcal{E}$, n is a natural number, and $X, \widetilde{x}$ and x are generic elements of TV, SV and $\mathcal{X}$, respectively. We write $\models P$ if there exist $N, N' \subset \mathbb{N}$ such that $(N, N') \models P$, and $\not\models P$ otherwise.

Rules 1–7 are self explanatory. Rule 8 states that a sequence pattern $SP_1 \cdot SP_2$ is well–typed if there are no labels which occur either four times ($N_1 \cap N_2 = \varnothing$) or three times ($N_1' \cap N_2 = N_1 \cap N_2' = \varnothing$). Labels occurring twice in $SP_1 \cdot SP_2$ are those which occur twice either in SP_1 or in SP_2 together with labels occurring once both in SP_1 and in SP_2. Rule 9 for the parallel composition is analogous to rule 8. Rule 10 states that the only labels which can be used for typing $\left(SP\right)^L \rfloor\, P$ must be different from those used for typing P. Moreover the labels used once in P must be used once in SP, that is these labels are used to bind elements inside the membrane to elements on the membrane itself.

The following lemma states some simple properties of the type system.

Lemma 3.22. *Given $N, N' \subset \mathbb{N}$, and $P \in \mathcal{P}$, then $(N, N') \models P$ implies:*
(i) *both N and N' are finite;*
(ii) *$N \cap N' = \varnothing$.*

Proof. It is easy to see that the typing algorithm always terminates, because it is recursively defined on the structure of patterns, which is always finite.

Both properties can be proved by induction on the structure of P.

As regards property (i), the base cases are the axioms of the type system. In these cases we have that N is empty and N' may contain at most one element. In the inductive

cases obtained by the sequential and parallel compositions and by containment it is easy to see that the elements in the set $N \cup N'$ are at most as many as those in $N_1 \cup N_1' \cup N_2 \cup N_2'$, which is, by induction hypothesis, a finite set.

As regards property (ii), the base cases are the axioms in which $N = \varnothing$. In the two induction cases of the sequential and parallel composition we have, by induction hypothesis, that $N_i \cap N_i' = \varnothing$ and, by the premise of the rule, that $N_1 \cap N_2 = N_1 \cap N_2' = N_2 \cap N_1' = \varnothing$. It follows that $(N_1 \cup N_2) \cap (N_1' \cup N_2') = \varnothing$, and consequently $((N_1 \cup N_2) \cup (N_1' \cup N_2')) \cap ((N_1' \cup N_2') \setminus (N_1' \cup N_2')) = \varnothing$. The induction step of the rule for containment is a trivial application of the induction hypothesis: we know that $N_1 \cap N_1' = \varnothing$, and hence also $N_1 \cap (N_1' \setminus N_2') = \varnothing$. □

The type system can be used to introduce a concept of well–formedness of patterns.

Definition 3.23 (Well–Formedness of Patterns)**.** *A pattern P is* well–formed *if and only if $\models P$ holds.*

Now we give two lemmas. The first relates the well–formedness of a pattern with the well–formedness of its subpatterns. The second states that well–formedness is preserved by structural congruence.

Lemma 3.24. *Given $P \in \mathcal{P}$ and P' a subpattern of P, then $\models P$ implies $\models P'$.*

Proof. We prove that $\not\models P'$ implies $\not\models P$. This can be done by induction on the structure of P. The base case is when $P = P'$, and the implication holds trivially. The inductive cases regards sequential composition, parallel composition and containment. In all these cases in the premises of the inference rules in the type systems there is the requirement that the subpatterns are typable patterns. □

Lemma 3.25. *Given $P_1, P_2 \in \mathcal{P}$, $\models P_1$ and $P_1 \equiv P_2$ imply $\models P_2$.*

Proof. Trivial structural induction. □

The use of labels to represent links is not new. In [23] well–formedness of terms is given by a concept of graph–likeness. We notice that in our case membranes, which are not present in the formalism of [23], make the treatment more complicated. In [47], where the concept of membrane is introduced, well–formedness of terms is given intuitively and not formally defined.

We say that a well–formed pattern P is *closed* if and only if $(N, \varnothing) \models P$ for some $N \subset \mathbb{N}$, and that it is *open* otherwise. Moreover, we say that P is *link–free* if and only if $(\varnothing, \varnothing) \models P$. Since patterns include terms, we use the same terminology also for terms. For example, $a \cdot b \cdot c \,|\, d \cdot x$ is a link–free pattern, $a \cdot b^1 \cdot c \,|\, d \cdot x^1$ is a closed pattern, and $a \cdot b^1 \cdot c^2 \,|\, d \cdot x^1$ is an open pattern.

In the following we shall use a notion of set of links of a pattern, namely the set of labels that occur twice in the top–level compartment of the pattern.

Definition 3.26. *The* set of links *of a pattern P is:*

$$L(P) = \{n | \#(n, L_M(P)) = 2\}$$

where $L_M(P)$ *is the* multiset of labels *of* P*, recursively defined as follows:*

$$L_M(\epsilon) = \varnothing \qquad L_M(\nu) = \varnothing \qquad L_M(\nu^n) = \{n\} \qquad L_M(\widetilde{x}) = \varnothing$$
$$L_M(SP_1 \cdot SP_2) = L_M(SP_1) \cup L_M(SP_2) \qquad L_M(P_1 \,|\, P_2) = L_M(P_1) \cup L_M(P_2)$$
$$L_M(\big(SP\big)^L \rfloor P) = L_M(SP) \cup (L_M(SP) \cap L_M(P)) \qquad L_M(X) = \varnothing$$

where $\nu \in \mathcal{E} \cup EV$*,* $n \in \mathbb{N}$*,* P_1, P_2 *are any pattern,* SP *is any sequence pattern.*

If P is a well–formed term, there exists $N \subset \mathbb{N}$ such that $(L(P), N) \models P$.

Let $\mathcal{A}$ be the set of all total injective functions $\alpha : \mathbb{N} \to \mathbb{N}$. Given $\alpha \in \mathcal{A}$, the α–*renaming* of a LCLS pattern P is the pattern $P\alpha$ obtained by replacing every label n in P by $\alpha(n)$. It holds that α–renaming preserves well–formedness.

Lemma 3.27. *Given* $P \in \mathcal{P}$*,* $\forall \alpha \in \mathcal{A}$ *it holds* $\models P \iff \models P\alpha$

Proof. If $(N, N') \models P$ then $(N_\alpha, N'_\alpha) \models P\alpha$, where N_α and N'_α are obtained by N and N', respectively, by replacing links in accordance with α, and vice–versa. □

Links in a term are placeholders: the natural number used in the two labels of a link has not a particular meaning. Hence, we can consider as equivalent patterns which are different only in the values of their links.

Definition 3.28 (α–equivalence)**.** *The* α*–equivalence relation* $=_\alpha$ *on LCLS patterns is the least equivalence relation which satisfies the following rules:*

$$\nu^{n1} \,|\, \mu^{n1} =_\alpha \nu^{n2} \,|\, \mu^{n2} \qquad \frac{P_1 \,|\, P_2 =_\alpha P_3}{P_2 \,|\, P_1 =_\alpha P_3} \qquad \frac{SP_1 \,|\, SP_2 =_\alpha P_3}{SP_1 \cdot SP_2 =_\alpha P_3}$$
$$\frac{P_1 =_\alpha P_2 \quad P_3 =_\alpha P_4 \quad L(P_1) \cap L(P_3) = L(P_2) \cap L(P_4) = \varnothing}{P_1 \,|\, P_3 =_\alpha P_2 \,|\, P_4}$$
$$\frac{SP_1 =_\alpha SP_2 \quad P_1 =_\alpha P_2 \quad L(SP_1) \cap L(P_1) = L(SP_2) \cap L(P_2) = \varnothing}{\big(SP_1\big)^L \rfloor P_1 =_\alpha \big(SP_2\big)^L \rfloor P_2}$$
$$\frac{\big(SP_1 \cdot SP'_1\big)^L \rfloor P_1 =_\alpha \big(SP_2 \cdot SP'_2\big)^L \rfloor P_2 \quad ni \notin L_M(SP_i \cdot SP'_i) \cup L_M(P_i)}{\big(SP_1 \cdot \nu^{n1} \cdot SP'_1\big)^L \rfloor (\mu^{n1} \,|\, P_1) =_\alpha \big(SP_2 \cdot \nu^{n2} \cdot SP'_2\big)^L \rfloor (\mu^{n2} \,|\, P_2)}$$
$$\frac{\big(SP_1 \cdot SP'_1\big)^L \rfloor P_1 =_\alpha \big(SP_2 \cdot SP'_2\big)^L \rfloor P_2 \quad ni \notin L_M(SP_i \cdot SP'_i) \cup L_M(P_i)}{\nu^{n1} \,|\, \big(SP_1 \cdot \mu^{n1} \cdot SP'_1\big)^L \rfloor P_1 =_\alpha \nu^{n2} \,|\, \big(SP_2 \cdot \mu^{n2} \cdot SP'_2\big)^L \rfloor P_2}$$

where $\nu, \mu \in \mathcal{E} \cup EV$*,* $n1, n2 \in \mathbb{N}$*,* P_1, P_2, P_3, P_4 *are any pattern,* SP_1, SP_2, SP_3, SP_4 *are any sequence pattern.*

It is easy to see that α–equivalence preserves well–formedness of patterns.

Lemma 3.29. *Given* $P_1, P_2 \in \mathcal{P}$*,* $\models P_1$ *and* $P_1 =_\alpha P_2$ *imply* $\models P_2$*.*

Proof. Trivial structural induction. □

Note that the labels which occur only once in a pattern P are not renamed by the α–equivalence relation. Instead, the application of an α–renaming function to P may change these labels. Moreover, labels which occur twice in more than one compartment of the pattern can be renamed differently in each compartment by the α–equivalence relation, while they are all renamed by the same value by applying some α–renaming function.

We say that an instantiation function σ is well–formed if it maps variables into well–formed closed terms and sequences. We denote with Σ_{wf} the set of all well–formed instantiation functions. Differently from CLS, the application of an instantiation function to a pattern does not correspond to the substitution of every variable in the pattern with the corresponding term given by the instantiation function, because this could lead to not well–formed terms. As an example, consider the well-formed pattern $P = a \cdot \widetilde{x} \,|\, X$ and a well–formed instantiation function σ such that $\sigma(\widetilde{x}) = b^1 \cdot c^1$ and $\sigma(X) = d^1 \,|\, e^1$. The application of σ to P would produce the term $P\sigma = a \cdot b^1 \cdot c^1 \,|\, d^1 \,|\, e^1$, which is not well–formed. Similarly, consider the well–formed pattern $P = a \cdot \widetilde{x} \cdot \widetilde{x}$ and the same well-formed instantiation function. We obtain $P\sigma = a \cdot b^1 \cdot c^1 \cdot b^1 \cdot c^1$, which is not well–formed. To avoid these situations, we define application of an instantiation function to a LCLS pattern in a way such that the links in the instantiations of all occurrences of all variables are renamed if necessary.

Definition 3.30 (Pattern Instantiation)**.** *Given a pattern $P \in \mathcal{P}$ and an instantiation function $\sigma \in \Sigma$, the application of σ to P is a LCLS term $P\sigma$ given by the following inductive definition:*

$$\epsilon\sigma = \epsilon \quad a\sigma = a \quad a^n\sigma = a^n \quad \widetilde{x}\sigma = \sigma(\widetilde{x}) \quad x\sigma = \sigma(x) \quad x^n\sigma = \sigma(x)^n \quad X\sigma = \sigma(X)$$

$$\frac{SP_i\sigma =_\alpha S_i \quad L(S_1) \cap L(S_2) = \varnothing}{SP_1 \cdot SP_2\ \sigma = S_1 \cdot S_2} \qquad \frac{P_i\sigma =_\alpha T_i \quad L(T_1) \cap L(T_2) = \varnothing}{P_1 \,|\, P_2\ \sigma = T_1 \,|\, T_2}$$

$$\frac{SP\sigma =_\alpha S \quad P\sigma =_\alpha T \quad L(S) \cap L(T) = \varnothing}{\big(SP\big)^L \rfloor P\ \sigma = \big(S\big)^L \rfloor T}$$

where P_1, P_2, P are any pattern, SP_1, SP_2, SP are any sequence pattern.

Now, by applying a well–formed instantiation function to a well–formed pattern, we obtain a well–formed term.

Lemma 3.31. *Given $P \in \mathcal{P}, \sigma \in \Sigma_{wf}$, it holds that $\models P$ implies $\models P\sigma$.*

Proof. By induction on the structure of P we prove that $(N, N') \models P$ implies $\exists N''.(N \cup N'', N') \models P\sigma$. Base cases correspond to the axioms of the type system. Among them, the only interesting cases are $(\varnothing, \varnothing) \models \widetilde{x}$ and $(\varnothing, \varnothing) \models X$. Since σ is well–formed in both cases we have that there exists N such that $(N, \varnothing) \models \sigma(\widetilde{x})$ or $(N, \varnothing) \models \sigma(X)$, hence, in both cases $\models P\sigma$ holds. In the inductive cases of the sequential and parallel composition and of containment the induction hypothesis can be applied trivially, with the help of Lemma 3.29. □

As in CLS, rewrite rules in LCLS are pairs of patterns.

Definition 3.32 (Rewrite Rules)**.** *A rewrite rule is a pair of patterns (P_1, P_2), denoted with $P_1 \mapsto P_2$, where $P_1, P_2 \in \mathcal{P}$, $P_1 \not\equiv \epsilon$ and such that $Var(P_2) \subseteq Var(P_1)$. We denote with $\Re$ the infinite set of all the possible rewrite rules.*

Our aim is to show that the application of a rewrite rule composed by well–formed patterns to a well–formed term produces another well–formed term. It is easy to see that, as a consequence of Lemma 3.31, this holds if variables of the rewrite rule are instantiated by a well–formed instantiation function. However, sometimes we would like to relax this constraint and allow a variable to be instantiated with an open term. For instance, we would permit the application of a rewrite rule $\widetilde{x} \cdot a \mapsto \widetilde{x} \cdot b$ to the term $c^1 \,|\, d^1 \cdot a$ (so to obtain $c^1 \,|\, d^1 \cdot b$), which requires that $\sigma(\widetilde{x}) = d^1$. Relaxing this constraint causes the introduction of constraints on the two patterns of the rewrite rules: they must not add or remove occurrences of variables, they cannot move variables from a compartment to another one, and they cannot add single occurrences of labels. To check these constraints we introduce a notion of compartment safety.

Definition 3.33 (Compartment Safety)**.** *The* compartment safety relation *cs on pairs of patterns is the least equivalence relation satisfying the following rules:*

$$cs(\epsilon,\epsilon) \qquad cs(\epsilon,\nu) \qquad cs(\nu^n,\mu^n) \qquad cs(\epsilon,\nu^n|\mu^n) \qquad cs(\widetilde{x},\widetilde{x}) \qquad cs(X,X)$$

$$\frac{cs(P_1,P_2) \quad cs(P_3,P_4)}{cs(P_1|P_3,P_2|P_4)} \qquad \frac{cs(P_1|P_2,P_3)}{cs(P_2|P_1,P_3)} \qquad \frac{cs(SP_1|SP_2,P_3)}{cs(SP_1 \cdot SP_2,P_3)}$$

$$\frac{cs(SP_1,SP_2) \quad cs(P_1,P_2)}{cs(\left(SP_1\right)^L \rfloor P_1, \left(SP_2\right)^L \rfloor P_2)} \qquad \frac{cs(\left(SP_1 \cdot SP_2\right)^L \rfloor P_1, \left(SP_3\right)^L \rfloor P_2)}{cs(\left(SP_2 \cdot SP_1\right)^L \rfloor P_1, \left(SP_3\right)^L \rfloor P_2)}$$

$$\frac{cs(\left(SP_1\right)^L \rfloor P_1, \left(SP_2\right)^L \rfloor P_2)}{cs(\left(SP_1\right)^L \rfloor P_1, \left(SP_2 \cdot \nu^n\right)^L \rfloor (\mu^n \,|\, P_2))} \qquad \frac{cs(\left(SP_1\right)^L \rfloor P_1, \left(SP_2\right)^L \rfloor P_2)}{cs(\left(SP_1\right)^L \rfloor P_1, \nu^n \,|\, \left(SP_2 \cdot \mu^n\right)^L \rfloor P_2)}$$

where $\nu, \mu \in \mathcal{E} \cup EV$*,* $n \in \mathbb{N}$*,* P_1, P_2, P_3, P_4 *are any pattern,* SP_1, SP_2, SP_3 *are any sequence pattern.*

Definition 3.34 (Compartment Safe Rewrite Rule)**.** *A rewrite rule* (P_1, P_2) *is* compartment safe (CS) *if* $cs(P_1, P_2)$ *holds. It is* compartment unsafe (CU) *otherwise. We denote with* $\Re^{CS} \subset \Re$ *the infinite set of CS rewrite rules, and with* $\Re^{CU} \subset \Re$ *the infinite set of CU rewrite rules.*

Now, we can introduce well–formedness also for rewrite rules.

Definition 3.35 (Well–Formedness of Rewrite Rules)**.** *A rewrite rule* $(P_1, P_2) \in \Re$ *is* well–formed *if* P_1 *and* P_2 *are well–formed patterns, and either* $(P_1, P_2) \in \Re^{CS}$ *or both* P_1 *and* P_2 *are closed patterns.*

The application of a well–formed rule satisfying compartment safety to a well–formed term preserves the well–formedness of the term even if variables are instantiated by a non well–formed instantiation function.

Lemma 3.36. *Given* $\sigma \in \Sigma$ *and a well–formed rewrite rule* (P_1, P_2) *such that* $(P_1, P_2) \in \Re^{CS}$*, it holds that* $\models P_1\sigma$ *implies* $\models P_2\sigma$*.*

Proof. By Definition 3.35 we know that there exist $N_1, N_2, N_1', N_2' \subset \mathbb{N}$ such that $(N_1, N_1') \models P_1$ and $(N_2, N_2') \models P_2$. Moreover, by definition of $\models$ we know that there exist $N_3, N_3' \subset \mathbb{N}$ such that $(N_3, N_3') \models P_1\sigma$. By Definition 3.33, we have that P_1 and P_2 have a similar

structure, they have the same variables placed in the same positions, and, as regards links P_2 may be different from P_1 only because it contains a different set of connected links (those links which appear twice). By definition of $P\sigma$ we have that those links do not interfere with the links of the instantiation of the variables, hence, we have that there exists $N_4 \subset \mathbb{N}$ such that $(N_4, N_3') \models P\sigma$. □

Now, we can define the semantics of LCLS.

Definition 3.37 (Semantics)**.** *Given a set of rewrite rules $\mathcal{R} \subseteq \Re$, such that $\mathcal{R} = \mathcal{R}^{CS} \cup \mathcal{R}^{CU}$ with $\mathcal{R}^{CS} \subset \Re^{CS}$ and $\mathcal{R}^{CU} \subset \Re^{CU}$, the* semantics *of LCLS is the least transition relation $\rightarrow$ on terms closed under $\equiv$ and $=_\alpha$, and satisfying the following inference rules:*

$$\textit{(appCS)} \quad \frac{(P_1, P_2) \in \mathcal{R}^{CS} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma \quad \alpha \in \mathcal{A}}{P_1\alpha\sigma \rightarrow P_2\alpha\sigma}$$

$$\textit{(appCU)} \quad \frac{(P_1, P_2) \in \mathcal{R}^{CU} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma_{wf} \quad \alpha \in \mathcal{A}}{P_1\alpha\sigma \rightarrow P_2\alpha\sigma}$$

$$\textit{(par)} \quad \frac{T_1 \rightarrow T_1' \qquad L(T_1) \cap L(T_2) = \{n_1, \ldots, n_M\} \qquad n_1', \ldots, n_M' \textit{ fresh}}{T_1 \,|\, T_2 \rightarrow T_1'\{^{n_1', \ldots, n_M'}/_{n_1, \ldots, n_M}\} \,|\, T_2}$$

$$\textit{(cont)} \quad \frac{T \rightarrow T' \qquad L(S) \cap L(T') = \{n_1, \ldots, n_M\} \qquad n_1', \ldots, n_M' \textit{ fresh}}{\big(S\big)^L \rfloor T \rightarrow \big(S\big)^L \rfloor T'\{^{n_1', \ldots, n_M'}/_{n_1, \ldots, n_M}\}}$$

where the symmetric rule for the parallel composition is omitted.

Rules *(appCS)* and *(appCU)* describe the application of compartment safe and compartment unsafe rewrite rules, respectively. In the latter case we require that the instantiation function used to apply the rule is well–formed. In both cases, an α–renaming function is used to rename the labels in the pattern, in particular those appearing only once in the top–level compartment. The *(par)* and *(cont)* rules propagate the effect of a rewrite rule application to contexts by resolving conflicts in the use of labels.

Finally, we can give a theorem which states that the application of well–formed rewrite rules to well–formed terms produces new well–formed terms.

Theorem 3.38 (Subject Reduction)**.** *Given a set of well–formed rewrite rules $\mathcal{R}$ and $T \in \mathcal{T}$, it holds that $\models T$ and $T \rightarrow T'$ imply $\models T'$.*

Proof. We know by Lemma 3.25 that the closure under $\equiv$ of the semantics preserves types, and by Lemma 3.29 that this holds also for the closure under $=_\alpha$, hence we prove the theorem by induction on the derivation of $T \rightarrow T'$.

The first base case is the application of rule *(appCS)*. In this case, one of the rewrite rules in $\mathcal{R}^{CS}$ has been applied, say (P_1, P_2), and $T = P_1\alpha\sigma$ for some $\sigma \in \Sigma$ and $\alpha \in \mathcal{A}$. It is easy to see that $(P_1, P_2) \in \Re^{CS}$ implies $(P_1\alpha, P_2\alpha) \in \Re^{CS}$. Now, by Lemma 3.36 we have that $\models P_1\alpha\sigma$ implies $\models P_2\alpha\sigma$, that is $\models T'$.

The second base case is the application of rule *(appCU)*. Since $\mathcal{R}$ is well–formed, we have $\models P_1$ and $\models P_2$. Now, by Lemma 3.27 we have that $\models P_2$ implies $\models P_2\alpha$, and by Lemma 3.31 we have that $\models P_2\alpha$ implies $\models P_2\alpha\sigma$, that is $\models T'$.

The first inductive case is when *(par)* is the last applied rule in the derivation tree of $T \to T'$. In this case $T = T_1 \,|\, T_2$, and $T' = T_1'\{^{n_1', \ldots, n_M'}/_{n_1, \ldots, n_M}\} \,|\, T_2$. By Lemma 3.24 we have that $\models T_1 \,|\, T_2$ implies $\models T_1$ and $\models T_2$, and by the application of the induction hypothesis we obtain $\models T_1'$. Now, by Lemma 3.27 we have that $\models T_1'$ implies $\models T_1'\{^{n_1', \ldots, n_M'}/_{n_1, \ldots, n_M}\}$ and, by the definition of the type system, we obtain $\models T_1'\{^{n_1', \ldots, n_M'}/_{n_1, \ldots, n_M}\} \,|\, T_2'$. The second inductive case is when *(cont)* is the last applied rule in the derivation tree of $T \to T'$. In this case $T = \big(S\big)^L \rfloor T_1$ and $T' = \big(S\big)^L \rfloor T_1'\{^{n_1', \ldots, n_M'}/_{n_1, \ldots, n_M}\}$. By Lemma 3.24 we have that $\models \big(S\big)^L \rfloor T_1$ implies $\models T_1$, and, by the application of the induction hypothesis, we obtain $\models T_1'$. Now, by Lemma 3.27 we have that $\models T_1'$ implies $\models T_1'\{^{n_1', \ldots, n_M'}/_{n_1, \ldots, n_M}\}$ and, by the definition of the type system, we obtain $\models \big(S\big)^L \rfloor T_1'\{^{n_1', \ldots, n_M'}/_{n_1, \ldots, n_M}\}$. □

It is trivial to prove that also LCLS is Turing complete.

Theorem 3.39 (Turing Completeness)**.** *The class of LCLS models is Turing complete.*

Proof. The subclass of LCLS models in which rewrite rules contains only link–free patterns and in which the initial term is a link–free term corresponds to the class of CLS models. By Theorem 3.16 we have that Turing completeness holds for CLS models, hence it holds also for LCLS models. □

3.7 The EGF Signalling Pathway in LCLS

Cells are often exposed to constantly changing physical and chemical environments, and react to changes in their environments through changes in the kinds of proteins they produce. For example, cells do not synthesize degradative enzymes (which are proteins) unless the substrates for these enzymes are present in the environment, and they do not synthesize proteins which stimulate cell proliferation if these new cells are not needed in the environment. Protein synthesis is regulated by activating some proteins already present in the cell. These bind to DNA at specific locations, enable transcription of some genes into RNA strands, which are translated into new proteins.

A classical example of reaction to an external stimulus is the epidermal growth factor (EGF) signal transduction pathway (also known as the RTK pathway). If EGF proteins are present in the environment of a cell, they must be interpreted as a signal from the environment meaning that new cells are needed, and hence the cell should react by synthesizing proteins which stimulate its proliferation. A cell recognizes the EGF signal because it has on its membrane some EGF receptor proteins (EGFR), which are transmembrane proteins (they have some intra–cellular and some extra–cellular domains). One of the extra–cellular domains binds to one EGF protein in the environment, forming a signal–receptor complex on the membrane. This causes a conformational change on the receptor protein that enables it to bind to another one signal–receptor complex. The formation of the binding of the two signal–receptor complexes (called dimerization) causes the phosphorylation[1] of some intra–cellular domains of the dimer. This, in turn, causes the internal domains of the dimer to be recognized by a protein that is inside the cell (in

[1]Phosphorylation is the addition of some phosphate groups (PO_4) to a protein.

the cytoplasm), called SHC. The protein SHC binds to the dimer, enabling a long chain of protein–protein interactions, which finally activate some proteins, such as one called ERK, which bind to the DNA and stimulate synthesis of proteins for cell proliferation.

We model in LCLS the steps of the EGF pathway up to the binding of the protein SHC to the dimer. We model the EGFR protein as the sequence $R_{E1} \cdot R_{E2} \cdot R_{I1} \cdot R_{I2}$, where R_{E1} and R_{E2} are two extra–cellular domains and R_{I1} and R_{I2} are two intra–cellular domains. The membrane of the cell is modeled as a looping sequence which could contain EGFR proteins. Outside the looping sequence (i.e. in the environment) there could be EGF proteins, and inside (i.e. in the cytoplasm) there could be SHC proteins. Rewrite rules modeling the pathway are the following:

$$EGF \,|\, \left(R_{E1} \cdot \widetilde{x}\right)^L \rfloor X \;\mapsto\; \left(SR_{E1} \cdot \widetilde{x}\right)^L \rfloor X \tag{R1}$$

$$\begin{aligned}\left(SR_{E1} \cdot R_{E2} \cdot x \cdot y \cdot \widetilde{x} \cdot SR_{E1} \cdot R_{E2} \cdot z \cdot w \cdot \widetilde{y}\right)^L \rfloor X \;\mapsto\\ \left(SR_{E1} \cdot R^1_{E2} \cdot x \cdot y \cdot SR_{E1} \cdot R^1_{E2} \cdot z \cdot w \cdot \widetilde{x} \cdot \widetilde{y}\right)^L \rfloor X\end{aligned} \tag{R2}$$

$$\begin{aligned}\left(R^1_{E2} \cdot R_{I1} \cdot \widetilde{x} \cdot R^1_{E2} \cdot R_{I1} \cdot \widetilde{y}\right)^L \rfloor X \;\mapsto\\ \left(R^1_{E2} \cdot PR_{I1} \cdot \widetilde{x} \cdot R^1_{E2} \cdot R_{I1} \cdot \widetilde{y}\right)^L \rfloor X\end{aligned} \tag{R3}$$

$$\begin{aligned}\left(R^1_{E2} \cdot PR_{I1} \cdot \widetilde{x} \cdot R^1_{E2} \cdot R_{I1} \cdot \widetilde{y}\right)^L \rfloor X \;\mapsto\\ \left(R^1_{E2} \cdot PR_{I1} \cdot \widetilde{x} \cdot R^1_{E2} \cdot PR_{I1} \cdot \widetilde{y}\right)^L \rfloor X\end{aligned} \tag{R4}$$

$$\begin{aligned}\left(R^1_{E2} \cdot PR_{I1} \cdot R_{I2} \cdot \widetilde{x} \cdot R^1_{E2} \cdot PR_{I1} \cdot R_{I2} \cdot \widetilde{y}\right)^L \rfloor (SHC \,|\, X) \;\mapsto\\ \left(R^1_{E2} \cdot PR_{I1} \cdot R^2_{I2} \cdot \widetilde{x} \cdot R^1_{E2} \cdot PR_{I1} \cdot R_{I2} \cdot \widetilde{y}\right)^L \rfloor (SHC^2 \,|\, X)\end{aligned} \tag{R5}$$

Rule R1 represents the binding of the EGF protein to the receptor domain R_{E1} with SR_{E1} as a result. Rule R2 represents that when two EGFR proteins activated by proteins EGF occur on the membrane, they may bind to each other to form a dimer (shown by the link 1). Rule R3 represents the phosphorylation of one of the internal domains R_{I1} of the dimer, and rule R4 represents the phosphorylation of the other internal domain R_{I1} of the dimer. The result of each phosphorylation is PR_{I1}. Rule R5 represents the binding of the protein SHC in the cytoplasm to an internal domain R_{I2} of the dimer. Remark that the binding of SHC to the dimer is represented by the link 2, allowing the protein SHC to continue the interactions to stimulate cell proliferation.

Let us denote the $R_{E1}{\cdot}R_{E2}{\cdot}R_{I1}{\cdot}R_{I2}$ by EGFR. By starting from a cell with some EGFR proteins on its membrane, some SHC proteins in the cytoplasm and some EGF proteins in the environment, a possible evolution is the following (we write on each transition the name of the rewrite rule applied):

$$\begin{aligned}
& EGF \,|\, EGF \,|\, \left(EGFR{\cdot}EGFR{\cdot}EGFR{\cdot}EGFR\right)^L \rfloor (SHC \,|\, SHC)\\
\xrightarrow{(R1)}\; & EGF \,|\, \left(SR_{E1}{\cdot}R_{E2}{\cdot}R_{I1}{\cdot}R_{I2}{\cdot}EGFR{\cdot}EGFR{\cdot}EGFR\right)^L \rfloor (SHC \,|\, SHC)\\
\xrightarrow{(R1)}\; & \left(SR_{E1}{\cdot}R_{E2}{\cdot}R_{I1}{\cdot}R_{I2}{\cdot}EGFR{\cdot}SR_{E1}{\cdot}R_{E2}{\cdot}R_{I1}{\cdot}R_{I2}{\cdot}EGFR\right)^L \rfloor (SHC \,|\, SHC)\\
\xrightarrow{(R2)}\; & \left(SR_{E1}{\cdot}R^1_{E2}{\cdot}R_{I1}{\cdot}R_{I2}{\cdot}SR_{E1}{\cdot}R^1_{E2}{\cdot}R_{I1}{\cdot}R_{I2}{\cdot}EGFR{\cdot}EGFR\right)^L \rfloor (SHC \,|\, SHC)\\
\xrightarrow{(R3)}\; & \left(SR_{E1}{\cdot}R^1_{E2}{\cdot}PR_{I1}{\cdot}R_{I2}{\cdot}SR_{E1}{\cdot}R^1_{E2}{\cdot}R_{I1}{\cdot}R_{I2}{\cdot}EGFR{\cdot}EGFR\right)^L \rfloor (SHC \,|\, SHC)
\end{aligned}$$

$$\xrightarrow{(R4)} \quad \left(SR_{E1}\cdot R^1_{E2}\cdot PR_{I1}\cdot R_{I2}\cdot SR_{E1}\cdot R^1_{E2}\cdot PR_{I1}\cdot R_{I2}\cdot EGFR\cdot EGFR\right)^L \rfloor (SHC \,|\, SHC)$$

$$\xrightarrow{(R5)} \quad \left(SR_{E1}\cdot R^1_{E2}\cdot PR_{I1}\cdot R^2_{I2}\cdot SR_{E1}\cdot R^1_{E2}\cdot PR_{I1}\cdot R_{I2}\cdot EGFR\cdot EGFR\right)^L \rfloor (SHC^2 \,|\, SHC)$$

Chapter 4

CLS as an Abstraction for Biomolecular Systems

In the previous chapter we have presented three calculi for the description of biological systems based on term rewriting in which terms include sequences, looping sequences and operators of containment and parallel composition. Two of the three calculi, namely Full–CLS and LCLS, allow expressing complex structures such as looping sequences whose elements can be other looping sequences, and complexes in which links are established between elements of different sequences or looping sequences. The ability of describing complex structures caused the semantics of the two formalisms to become complex. In the case of Full–CLS we needed to define a complex structural congruence relation to resolve some ambiguities in the definition of terms. In LCLS, instead, we needed to develop a type system to guarantee that no ill–formed terms could be produced by the application of rewrite rules.

The simplest of the three calculi, namely CLS, has a simple semantics. However, it is expressive enough to describe interesting biological pathways (as the degradation of lactose in E.coli). For this reason, in the rest of the thesis we will concentrate on CLS, and we will use it to develop bisimulation relations and a stochastic extension.

In this chapter we give some guidelines for the modeling of biological systems in CLS. What could seem strange is the use of looping sequences for the description of membranes, as sequencing is not a commutative operation and this do not correspond to the usual fluid representation of membranes in which object can move freely. What one would expect is to have a multiset or a parallel composition of objects on a membrane. In the case of CLS, what could be used is a parallel composition of sequences. To address this problem, we define an extension of CLS, called CLS+, in which the looping operator can be applied to a parallel composition of sequences, and we show that if we add a slight restriction on the use of variables in CLS+, we can translate quite easily CLS+ models into CLS ones. CLS+ has a semantics which is more complicated than the semantics of CLS, hence in the rest of the thesis we shall continue using CLS, by knowing that we are not loosing too much expressiveness with respect to CLS+. This choice, will simplify in particular proofs of the theorems of the following chapters, as most of them are performed by induction on the derivation of the transitions obtained from the semantics of the calculus.

Biomolecular Entity	**CLS Term**
Elementary object (genes, domains, other molecules, etc...)	Alphabet symbol
DNA strand	Sequence of elements representing genes
RNA strand	Sequence of elements representing transcribed genes
Protein	Sequence of elements representing domains or single alphabet symbol
Molecular population	Parallel composition of molecules
Membrane	Looping sequence

Table 4.1: Guidelines for the abstraction of biomolecular entities into CLS.

4.1 CLS Modeling Guidelines

In this section we describe how CLS can be used to model biomolecular systems as Regev and Shapiro did in [68] for the π–calculus. An abstraction is a mapping from a real–world domain to a mathematical domain, that may allow highlighting some essential properties of a system while ignoring other, complicating, ones. In [68], Regev and Shapiro show how to abstract biomolecular systems as concurrent computation by identifying the biomolecular entities and events of interest and by associating them with concepts of concurrent computation such as concurrent processes and communications. In particular, they give some guidelines for the abstraction of biomolecular systems to the π–calculus, and give some simple examples.

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 rewriting rules. We already introduced the biological interpretation of CLS operators and we gave some example of models of biomolecular systems in chapter 3. Here, we want to give more general guidelines.

First of all, we select the biomolecular entities of interest. Since we want to describe cells, we consider molecular populations and membranes. Molecular populations are groups of molecules which are in the same compartment of the cell. Molecules can be of many types: we classify them 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 contain something (hence, create a compartment) and it may have molecules on its surface.

Now, we select the biomolecular events of interest. The simplest kind of event is the change of state of an elementary object. Then, we may have 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, we may have interactions between membranes and molecules: in

Biomolecular Event	**Examples of CLS Rewrite Rule**
State change	$a \;\mapsto\; b$ $\widetilde{x}\cdot a\cdot\widetilde{y} \;\mapsto\; \widetilde{x}\cdot b\cdot\widetilde{y}$
Complexation	$a\,|\,b \;\mapsto\; c$ $\widetilde{x}\cdot a\cdot\widetilde{y}\,|\,b \;\mapsto\; \widetilde{x}\cdot c\cdot\widetilde{y}$
Decomplexation	$c \;\mapsto\; a\,|\,b$ $\widetilde{x}\cdot c\cdot\widetilde{y} \;\mapsto\; \widetilde{x}\cdot a\cdot\widetilde{y}\,|\,b$
Catalysis	$c\,|\,P_1 \;\mapsto\; c\,|\,P_2$ where $P_1 \;\mapsto\; P_2$ is the catalyzed event
State change on membrane	$\left(a\cdot\widetilde{x}\right)^L\rfloor X \;\mapsto\; \left(b\cdot\widetilde{x}\right)^L\rfloor X$
Complexation on membrane	$\left(a\cdot\widetilde{x}\cdot b\cdot\widetilde{y}\right)^L\rfloor X \;\mapsto\; \left(c\cdot\widetilde{x}\cdot\widetilde{y}\right)^L\rfloor X$ $a\,|\,\left(b\cdot\widetilde{x}\right)^L\rfloor X \;\mapsto\; \left(c\cdot\widetilde{x}\right)^L\rfloor X$ $\left(b\cdot\widetilde{x}\right)^L\rfloor(a\,|\,X) \;\mapsto\; \left(c\cdot\widetilde{x}\right)^L\rfloor X$
Decomplexation on membrane	$\left(c\cdot\widetilde{x}\right)^L\rfloor X \;\mapsto\; \left(a\cdot b\cdot\widetilde{x}\right)^L\rfloor X$ $\left(c\cdot\widetilde{x}\right)^L\rfloor X \;\mapsto\; a\,|\,\left(b\cdot\widetilde{x}\right)^L\rfloor X$ $\left(c\cdot\widetilde{x}\right)^L\rfloor X \;\mapsto\; \left(b\cdot\widetilde{x}\right)^L\rfloor(a\,|\,X)$
Catalysis on membrane	$\left(c\cdot\widetilde{x}\cdot SP_1\cdot\widetilde{y}\right)^L \;\mapsto\; \left(c\cdot\widetilde{x}\cdot SP_2\cdot\widetilde{y}\right)^L$ where $SP_1 \;\mapsto\; SP_2$ is the catalyzed event
Membrane crossing	$a\,|\,\left(\widetilde{x}\right)^L\rfloor X \;\mapsto\; \left(\widetilde{x}\right)^L\rfloor(a\,|\,X)$ $\left(\widetilde{x}\right)^L\rfloor(a\,|\,X) \;\mapsto\; a\,|\,\left(\widetilde{x}\right)^L\rfloor X$ $\widetilde{x}\cdot a\cdot\widetilde{y}\,|\,\left(\widetilde{z}\right)^L\rfloor X \;\mapsto\; \left(\widetilde{z}\right)^L\rfloor(\widetilde{x}\cdot a\cdot\widetilde{y}\,|\,X)$ $\left(\widetilde{z}\right)^L\rfloor(\widetilde{x}\cdot a\cdot\widetilde{y}\,|\,X) \;\mapsto\; \widetilde{x}\cdot a\cdot\widetilde{y}\,|\,\left(\widetilde{z}\right)^L\rfloor X$
Catalyzed membrane crossing	$a\,|\,\left(b\cdot\widetilde{x}\right)^L\rfloor X \;\mapsto\; \left(b\cdot\widetilde{x}\right)^L\rfloor(a\,|\,X)$ $\left(b\cdot\widetilde{x}\right)^L\rfloor(a\,|\,X) \;\mapsto\; a\,|\,\left(b\cdot\widetilde{x}\right)^L\rfloor X$ $\widetilde{x}\cdot a\cdot\widetilde{y}\,|\,\left(b\cdot\widetilde{z}\right)^L\rfloor X \;\mapsto\; \left(b\cdot\widetilde{z}\right)^L\rfloor(\widetilde{x}\cdot a\cdot\widetilde{y}\,|\,X)$ $\left(b\cdot\widetilde{z}\right)^L\rfloor(\widetilde{x}\cdot a\cdot\widetilde{y}\,|\,X) \;\mapsto\; \widetilde{x}\cdot a\cdot\widetilde{y}\,|\,\left(b\cdot\widetilde{z}\right)^L\rfloor X$
Membrane joining	$\left(\widetilde{x}\right)^L\rfloor(a\,|\,X) \;\mapsto\; \left(a\cdot\widetilde{x}\right)^L\rfloor X$ $\left(\widetilde{x}\right)^L\rfloor(\widetilde{y}\cdot a\cdot\widetilde{z}\,|\,X) \;\mapsto\; \left(\widetilde{y}\cdot a\cdot\widetilde{z}\cdot\widetilde{x}\right)^L\rfloor X$
Catalyzed membrane joining	$\left(b\cdot\widetilde{x}\right)^L\rfloor(a\,|\,X) \;\mapsto\; \left(a\cdot b\cdot\widetilde{x}\right)^L\rfloor X$ $\left(\widetilde{x}\right)^L\rfloor(a\,|\,b\,|\,X) \;\mapsto\; \left(a\cdot\widetilde{x}\right)^L\rfloor(b\,|\,X)$ $\left(b\cdot\widetilde{x}\right)^L\rfloor(\widetilde{y}\cdot a\cdot\widetilde{z}\,|\,X) \;\mapsto\; \left(\widetilde{y}\cdot a\cdot\widetilde{z}\cdot\widetilde{x}\right)^L\rfloor X$ $\left(\widetilde{x}\right)^L\rfloor(\widetilde{y}\cdot a\cdot\widetilde{z}\,|\,b\,|\,X) \;\mapsto\; \left(\widetilde{y}\cdot a\cdot\widetilde{z}\cdot\widetilde{x}\right)^L\rfloor(b\,|\,X$
Membrane fusion	$\left(\widetilde{x}\right)^L\rfloor(X)\,|\,\left(\widetilde{y}\right)^L\rfloor(Y) \;\mapsto\; \left(\widetilde{x}\cdot\widetilde{y}\right)^L\rfloor(X\,|\,Y)$
Catalyzed membrane fusion	$\left(a\cdot\widetilde{x}\right)^L\rfloor(X)\,|\,\left(b\cdot\widetilde{y}\right)^L\rfloor(Y) \;\mapsto\; \left(a\cdot\widetilde{x}\cdot b\cdot\widetilde{y}\right)^L\rfloor(X\,|\,Y)$
Membrane division	$\left(\widetilde{x}\cdot\widetilde{y}\right)^L\rfloor(X\,|\,Y) \;\mapsto\; \left(\widetilde{x}\right)^L\rfloor(X)\,|\,\left(\widetilde{y}\right)^L\rfloor(Y)$
Catalyzed membrane division	$\left(a\cdot\widetilde{x}\cdot b\cdot\widetilde{y}\right)^L\rfloor(X\,|\,Y) \;\mapsto\; \left(a\cdot\widetilde{x}\right)^L\rfloor(X)\,|\,\left(b\cdot\widetilde{y}\right)^L\rfloor(Y)$

Table 4.2: Guidelines for the abstraction of biomolecular events into CLS.

particular a molecule may cross or join a membrane. Finally, we may have interactions between membranes: in this case there may be many kinds of interactions (fusions, divisions, phagocytosis, exocytosis, etc. . .).

The guidelines for the abstraction of biomolecular entities and events into CLS are given in Table 4.1 and Table 4.2, respectively. Entities are associated with CLS terms: elementary objects are modeled as alphabet symbols, non–elementary objects as CLS sequences and membranes as looping sequences. Note that proteins are associated also with alphabet symbols, and this will be very often the preferred representation for them. This choice is a consequence of the fact that protein interaction at the domain level cannot be modeled properly with CLS (for this reason we introduced LCLS in the previous chapter).

Biomolecular events are associated with CLS rewrite rules. We give some examples of rewrite rules for each type of event. The list of examples is not complete: one could define also rewrite rules for the description of complexation/decomplexation events involving more than two molecules, or catalysis event in which the catalyzing molecule is in on a membrane and the catalyzed event occurs in its content. Moreover, in the table we give only a few very simple examples of membrane interaction, but more complex and realistic kinds of interaction can be defined.

We remark that in the second example of rewrite rule associated with the complexation event we have that one of the two molecules which are involved should be either an elementary object or a protein modeled as a single alphabet symbol. As before, this is caused by the problem of modeling protein interaction at the domain level.

Now, what could be the object of criticism is the use of (looping) sequences as an abstraction for membranes. Sequencing is not a commutative operator, while membranes have a form of "natural" commutativity because they are fluid surfaces. Commutativity can be added to looping sequences by allowing the application of the looping operator to a parallel composition of sequences. We study this extension of the formalism in the following section.

4.2 Definition of CLS+

We define CLS+ as an extension of CLS in which the looping operator can be applied to a parallel composition of sequences. This would allow modeling membranes in a more natural way. However, as we shall see, this will require the definition of a more complex semantics.

Terms in CLS+ are defined as follows.

Definition 4.1 (Terms)**.** Terms T, Branes B, *and* Sequences S *of CLS+ are given by the following grammar:*

$$\begin{array}{lcl} T & ::= & S \;\mid\; \left(B\right)^{L} \rfloor T \;\mid\; T \,|\, T \\ B & ::= & S \;\mid\; S \,|\, S \\ S & ::= & \epsilon \;\mid\; a \;\mid\; S \cdot S \end{array}$$

where a is a generic element of $\mathcal{E}$. We denote with $\mathcal{T}$ the infinite set of terms, with $\mathcal{B}$ the infinite set of branes and with $\mathcal{S}$ the infinite set of sequences.

The structural congruence relation of CLS+ is a trivial extension of the one of CLS. The only difference is that commutativity of branes replaces rotation of looping sequences.

Definition 4.2 (Structural Congruence)**.** *The structural congruence relations $\equiv_S$, $\equiv_B$ and $\equiv_T$ are the least congruence relations on sequences, on branes and on terms, respectively, satisfying the following rules:*

$$S_1 \cdot (S_2 \cdot S_3) \equiv_S (S_1 \cdot S_2) \cdot S_3 \qquad S \cdot \epsilon \equiv_S \epsilon \cdot S \equiv_S S$$

$$S_1 \equiv_S S_2 \text{ implies } S_1 \equiv_B S_2$$
$$B_1 \,|\, B_2 \equiv_B B_2 \,|\, B_1 \qquad B_1 \,|\, (B_2 \,|\, B_3) \equiv_B (B_1 \,|\, B_2) \,|\, B_3 \qquad B \,|\, \epsilon \equiv_B B$$

$$S_1 \equiv_S S_2 \text{ implies } S_1 \equiv_T S_2$$
$$B_1 \equiv_B B_2 \text{ implies } \left(B_1\right)^L \rfloor T \equiv_T \left(B_2\right)^L \rfloor T$$
$$T_1 \,|\, T_2 \equiv_T T_2 \,|\, T_1 \qquad T_1 \,|\, (T_2 \,|\, T_3) \equiv_T (T_1 \,|\, T_2) \,|\, T_3 \qquad T \,|\, \epsilon \equiv_T T \qquad \left(\epsilon\right)^L \rfloor \epsilon \equiv \epsilon$$

Now, to define patterns in CLS+ we consider an additional type of variables with respect of CLS, namely brane variables. We assume a set of brane variables BV ranged over by $\overline{x}, \overline{y}, \overline{z}, \ldots$.

Definition 4.3 (Patterns)**.** Patterns P, brane patterns BP *and* sequence patterns SP *of CLS+ are given by the following grammar:*

$$\begin{array}{rcl} P & ::= & SP \;\mid\; \left(BP\right)^L \rfloor P \;\mid\; P \,|\, P \;\mid\; X \\ BP & ::= & SP \;\mid\; SP \,|\, SP \;\mid\; \overline{x} \\ SP & ::= & \epsilon \;\mid\; a \;\mid\; SP \cdot SP \;\mid\; \widetilde{x} \;\mid\; x \end{array}$$

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

As usual, rewrite rules are pairs of patterns.

Definition 4.4 (Rewrite Rules)**.** *A rewrite rule is a pair of patterns (P_1, P_2), denoted with $P_1 \mapsto P_2$, where $P_1, P_2 \in \mathcal{P}$, $P_1 \not\equiv \epsilon$ and such that $Var(P_2) \subseteq Var(P_1)$. We denote with $\Re$ the infinite set of all the possible rewrite rules. We say that a rewrite rule is* ground *if $Var(P_1) = Var(P_2) = \varnothing$, and a set of rewrite rules $\mathcal{R} \in Re$ is* ground *if all the rewrite rules it contains are ground.*

Now, differently from CLS, we have that a rule such as $a \,|\, b \mapsto c$ could be applied to elements of a looping sequence. For instance, $a \,|\, b \mapsto c$ can be applied to the term $\left(a \,|\, b\right)^L \rfloor d$ so to obtain the term $\left(c\right)^L \rfloor d$. However, a rule such as $\left(a\right)^L \rfloor b \mapsto c$ still cannot be applied to elements of a looping sequences, as $\left(\left(a\right)^L \rfloor b\right)^L \rfloor c$ is not a CLS+ term.

The rules that can be applied to elements of a looping sequence are those having the form (B_1, B_2) with $B_1, B_2 \in \mathcal{B}$. We call these rules *brane rules* and we denote as $\Re_{\mathcal{B}} \subset \Re$ the infinite set containing all of them. Now, in the semantics of CLS+ we have to take into account brane rules and allow them to be applied also to elements of looping sequences. Hence, we define the semantics as follows.

Definition 4.5 (Semantics)**.** *Given a set of rewrite rules $\mathcal{R} \subseteq \Re$, and a set of brane rules $\mathcal{R}_{\mathcal{B}} \subseteq \mathcal{R}$, such that $(\mathcal{R} \setminus \mathcal{R}_{\mathcal{B}}) \cap \Re_{\mathcal{B}} = \varnothing$, the* semantics *of CLS is the least transition relation*

$\rightarrow$ on terms closed under $\equiv$, and satisfying the following inference rules:

$$\frac{(P_1, P_2) \in \mathcal{R} \quad P_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma}{P_1\sigma \rightarrow P_2\sigma} \qquad \frac{T_1 \rightarrow T_2}{T \,|\, T_1 \rightarrow T \,|\, T_2} \qquad \frac{T_1 \rightarrow T_2}{\left(B\right)^L \rfloor T_1 \rightarrow \left(B\right)^L \rfloor T_2}$$

$$\frac{(BP_1, BP_2) \in \mathcal{R}_{\mathcal{B}} \quad BP_1\sigma \not\equiv \epsilon \quad \sigma \in \Sigma}{BP_1\sigma \rightarrow_{\mathcal{B}} BP_2\sigma} \qquad \frac{B_1 \rightarrow_{\mathcal{B}} B_2}{B \,|\, B_1 \rightarrow_{\mathcal{B}} B \,|\, B_2} \qquad \frac{B_1 \rightarrow_{\mathcal{B}} B_2}{\left(B_1\right)^L \rfloor T \rightarrow \left(B_2\right)^L \rfloor T}$$

where $\rightarrow_{\mathcal{B}}$ is a transition relation on branes, and where the symmetric rules for the parallel composition of terms and of branes are omitted.

In the definition of the semantics of CLS+ we use an additional transition relation $\rightarrow_{\mathcal{B}}$ on branes. This relation is used to describe the application of a brane rule to elements of a looping sequence. As usual, a CLS+ model is composed by a term, representing the initial state of the modeled system, and a set of rewrite rules.

In the following section we show that CLS+ models can be translated into CLS models. The translation into CLS preserves the semantics of the model.

4.3 Translation of CLS+ into CLS

The first step of the translation of a CLS+ models into CLS is a preprocessing procedure. For each brane rule (BP_1, BP_2) in the CLS+ model, we add to the set of rules of the model a new rule, namely $(\left(BP_1 \,|\, \overline{x}\right)^L \rfloor X, \left(BP_2 \,|\, \overline{x}\right)^L \rfloor X)$. This new rule is redundant in the model, as every time it can be applied to a CLS+ term, also the original one can be applied with the same result. However, the translation we are going to define will translate the original rule into a CLS rule that will be applicable only inside looping sequences, or at the top level of the term, and will translate the new rule only to elements that compose a looping sequence.

Now, the translation of CLS+ in to CLS consists mainly of an encoding function, denoted $\{\![\cdot]\!\}$, which maps CLS+ patterns into CLS patterns. This encoding function will be used to translate each rewrite rule of the CLS+ model into a rewrite rule for the corresponding CLS model, and to translate the term representing the initial state of the system in the CLS+ model into a CLS term for the corresponding CLS model.

The encoding function for CLS+ patterns is defined as follows. We assume a total and injective function from brane variables into a subset of term variables that are never used in CLS models. More easily, we assume brane variables to be a subset of the term variables of CLS. Moreover, we assume *in* and *out* to be symbols of the alphabet $\mathcal{E}$ never used in CLS models.

The encoding follows the "ball–bearing" technique described by Cardelli in [13]. Intuitively, every CLS+ looping sequence is translated into a couple of CLS looping sequences, one contained in the other, with the brane patterns of the CLS+ looping sequence between the two corresponding CLS looping sequences.

Definition 4.6 (Encoding Function)**.** *The encoding function $\{\![\cdot]\!\}$ maps CLS+ patterns*

into CLS patterns, and is given by the following recursive definition:

$$\{\!|SP|\!\} = SP$$
$$\{\!|X|\!\} = X$$
$$\{\!|(BP)^L \rfloor P|\!\} = (out)^L \rfloor (BP \,|\, (in)^L \rfloor \{\!|P|\!\})$$
$$\{\!|P_1 \,|\, P_2|\!\} = \{\!|P_1|\!\} \,|\, \{\!|P_2|\!\}$$

A CLS rewrite rule is obtained from each CLS+ rewrite rule of the translated model by applying the encoding function to the two patterns of the rule. More precisely, given a CLS+ rule $P_1 \mapsto P_2$, the corresponding CLS rule is $(in)^L \rfloor (\{\!|P_1|\!\} \,|\, X) \mapsto (in)^L \rfloor (\{\!|P_2|\!\} \,|\, X)$ where X is a term variable that does not occur in P_1 and P_2. For example, by applying the encoding to the two patterns of the CLS+ rewrite rule

$$R \;=\; b \cdot x \,|\, c \;\mapsto\; b \cdot x$$

we obtain

$$R_{\{\!|\cdot|\!\}} = (in)^L \rfloor (b \cdot x \,|\, c \,|\, X) \;\mapsto\; (in)^L \rfloor (b \cdot x \,|\, X)\,.$$

The encoding of a CLS+ term into a CLS term is as follows: given a CLS+ term T the corresponding CLS term is $(in)^L \rfloor \{\!|T|\!\}$. In this case we have that the encoding function never encounters variables. Consider, as an example, the following CLS+ term:

$$T = a \,|\, (c \,|\, d \,|\, b \cdot b \,|\, d)^L \rfloor d$$

the corresponding CLS term is as follows:

$$T_{\{\!|\cdot|\!\}} = (in)^L \rfloor (a \,|\, (out)^L \rfloor (c \,|\, d \,|\, b \cdot b \,|\, d \,|\, (in)^L \rfloor d))$$

Now, it is easy to see that R can be applied to T, because parallel components in the looping sequence can be commuted, and the result of the application is

$$T' = a \,|\, (b \cdot b \,|\, d \,|\, d)^L \rfloor d$$

but the corresponding CLS rewrite rule $R_{\{\!|\cdot|\!\}}$ cannot be applied to $T_{\{\!|\cdot|\!\}}$. However, we have that $R \in \mathcal{R}_{\mathcal{B}}$, hence by the preprocessing phase described above we have that also

$$R' = (b \cdot x \,|\, c \,|\, \overline{x})^L \rfloor X \;\mapsto\; (b \cdot x \,|\, \overline{x})^L \rfloor X$$

is a rule of the CLS+ model. By translating rule R' we obtain

$$R'_{\{\!|\cdot|\!\}} = (in)^L \rfloor ((out)^L \rfloor (b \cdot x \,|\, c \,|\, \overline{x} \,|\, (in)^L \rfloor X) \,|\, Y) \;\mapsto\; (in)^L \rfloor ((out)^L \rfloor (b \cdot x \,|\, \overline{x} \,|\, (in)^L \rfloor X) \,|\, Y)$$

that can be applied to $T_{\{\!|\cdot|\!\}}$. The result of the application is

$$(in)^L \rfloor (a \,|\, (out)^L \rfloor (b \cdot b \,|\, d \,|\, d \,|\, (in)^L \rfloor d))$$

that corresponds exactly to the encoding of T'.

Concluding, CLS+ allows describing biomolecular systems by abstracting membranes in a more natural manner with respect to CLS. However, we have shown, by giving some modeling guidelines, that CLS is expressive enough to model all the biomolecular entities and events we are interest in. Moreover, CLS has a simpler semantics than CLS+. For these reasons, in the rest of the thesis we will concentrate on CLS.

Chapter 5

CLS and Related Formalisms

In this chapter we compare CLS with Brane Calculi [13] and P–Systems [59]. We choose them because they are well-established formalisms with many similarities with CLS. As CLS, both Brane Calculi and P–Systems are inspired by biological systems, can be used to model these systems, and include a notion of membrane. Variants of P Systems that includes operations inspired by Brane Calculi are currently under study [15, 74].

Brane Calculi are a family of process calculi specialized in the description of membrane activity, and they allow associating processes with membranes. These processes are composed by actions the execution of which has an effect on the membrane structure. Some examples of actions are phagocytosis (a membrane engulfs another one), exocytosis (a membrane expels another one), and pinocytosis (a new membrane is created inside another one). These three actions are enough to define the simplest of Brane Calculi, namely the PEP calculus. Other actions, such as fusions of membranes and mitosis can be used to define different calculi of the family. Moreover, extensions of Brane Calculi allow describing interactions with molecules and complexes, such as letting them enter and exit membranes.

We consider the PEP calculus, as it is the simplest of Brane Calculi, and we provide a sound and complete encoding into CLS. We believe that the same technique we used to encode the PEP calculus can be used to encode also other Brane Calculi. Moreover, we do not consider the translation of CLS into Brane Calculi because the absence of constraints in the definition of CLS rewrite rules would make the work extremely hard.

Differently from Brane Calculi, P–Systems (in their most common formulation) do not allow describing complex membrane activities such as phagocytosis and exocytosis. However, they are specialized in the description of reactions between molecules which are placed in a compartment of a complex membrane structure.

A P–System is a membrane structure (a nesting of membranes) in which there could be multisets of objects representing molecules. A set of multiset rewrite rules is associated with each membrane, and describe the reactions that may occur between the molecules contained in the membrane. The result of the application of a rewrite rule can either remain in the same membrane, or exit the membrane, or enter an inner membrane. Priorities can be imposed on rewrite rules, meaning that some rules can be applied only if some others cannot, and it is possible for a membrane to dissolve and release its content into the environment.

A peculiarity of P–Systems is that rewrite rules are applied in a fully–parallel manner,

namely in one step of evolution of the system all rules are applied as many times as possible (to different molecules), and this is one of the main differences with respect to CLS in which at each step one only rewrite rule is applied. We show that P–Systems can be translated into CLS, and that the execution of a (fully parallel) step of a P–System is simulated by a sequence of steps in CLS. A variant of P Systems, called Sequential P Systems, in which rules are applied sequentially is described in [22]. We do not consider the translation of this variant into CLS as it would be quite trivial and devoid of interest. As for Brane Calculi, we do not provide the inverse translation because of absence of constraints on the rewrite rules of CLS.

The result of this comparison with Brane Calculi and P–System is a proof of the expressiveness of CLS, as models developed by using other formalisms can be translated into CLS models in a relatively easy way.

5.1 Encoding Brane Calculi

In this section, we recall the definition of the phago/exo/pino (PEP) calculus, which is the simplest of Brane Calculi [13], and we give a sound and complete encoding of it into CLS. The technique we will use to encode the PEP calculus can be used also to encode other calculi of the Brane Calculi family.

5.1.1 The PEP Calculus

The syntax and the semantics of the PEP calculus are summarized in Figure 5.1. Terms are systems. Systems consist of composition of systems, $\circ$, with unit $\diamond$. Replication ! is used to model the notion of "multitude" of systems. Systems can be membranes containing systems, $\sigma(\!|P|\!)$. Membranes can be a parallel compositions $\sigma|\sigma'$ with unit $\mathbf{0}$, or replication of membranes, or action prefixing.

Actions are: *phagocytosis*, denoted ϕ_n, incorporates one external membrane into another by "engulfing" it; *exocytosis*, denoted by ε_n, is the reverse process; *pinocytosis*, denoted by $\circledcirc$, engulfs zero external membranes. Phagocytosis and exocytosis have co-actions that are intended to interact with, indicated by the symbol $^{\perp}$. Pinocytosis does not have a co-action. Figure 5.2 gives a pictorial representation of the three actions.

We consider a structural congruence relation $\equiv$ that describes associativity, commutativity, replication and unit elements of operators on systems and membranes. We denote with PEP the infinite set of Systems, and with $Branes$ the infinite set of membranes in the PEP calculus. Moreover, we denote with $\mathcal{N}$ the (possibly infinite) set of names n used as subscripts of Actions.

5.1.2 Encoding of the PEP Calculus into CLS

We define an encoding of a system of the PEP calculus into a CLS term. The encoding of a system results in a pair of a CLS sequence and a set of alphabet symbols.

Operators and actions of the encoded system are translated into elements of the sequence. More precisely, $\diamond$ is translated into $\mathbf{0}$, and the three operators on systems $\_\circ\_$, $!\_$ and $\_(\!|\_|\!)$ are translated into $circ$,$bangS$ and $brane$, respectively, with $\mathbf{0}, circ, bangS, brane \in \mathcal{E}$. Moreover, as regards branes, $\mathbf{0}$ is translated into $\mathbf{0}$, $\_|\_$ into par and $!\_$ into $bangB$, with $par, bangB \in \mathcal{E}$. Phagocytosis and exocytosis actions are translated into sequences of

Syntax

$$
\begin{array}{rcll}
P, Q, R \ldots & ::= & \diamond \mid P \circ P \mid\ !P \mid \sigma(\!|P|\!) & \text{Systems} \\
\sigma, \tau, \rho, \ldots & ::= & \mathbf{0} \mid \sigma|\sigma \mid\ !\sigma \mid\ a.\sigma & \text{Branes} \\
a, b, c, \ldots & ::= & \phi_n \mid \phi_n^{\perp}(\sigma) \mid\ \varepsilon_n \mid \varepsilon_n^{\perp} \mid \circledcirc\,(\sigma) & \text{Actions}
\end{array}
$$

Structural Congruence

The least congruence relation $\equiv$ satisfying the following axioms

$$P \circ Q \equiv Q \circ P \qquad P \circ (Q \circ R) \equiv (P \circ Q) \circ R \qquad P \circ \diamond \equiv P$$

$$!\diamond \equiv \diamond \qquad !!P \equiv !P \qquad !P \equiv P \circ !P \qquad \mathbf{0}(\!| \diamond |\!) \equiv \diamond$$

$$\sigma|\tau \equiv \tau|\sigma \qquad \sigma|(\tau|\rho) \equiv (\sigma|\tau)|\rho \qquad \sigma|\mathbf{0} \equiv \sigma$$

$$!\mathbf{0} \equiv \mathbf{0} \qquad !!\sigma \equiv !\sigma \qquad !\sigma \equiv \sigma|!\sigma$$

Reaction Semantics

The least relation containing the following axioms, closed wrt $\_ \circ P$, $\sigma(\!|\_|\!)$ and $\equiv$

$$\text{(phago)} \quad \phi_n.\sigma|\sigma_0(\!|P|\!) \circ \phi_n^{\perp}(\rho).\tau|\tau_0(\!|Q|\!) \rightarrow \tau|\tau_0(\!|\rho(\!|\sigma|\sigma_0(\!|P|\!)|\!) \circ Q|\!)$$

$$\text{(exo)} \quad \varepsilon_n^{\perp}.\tau|\tau_0(\!|\varepsilon_n.\sigma|\sigma_0(\!|P|\!) \circ Q|\!) \rightarrow P \circ \sigma|\sigma_0|\tau|\tau_0(\!|Q|\!)$$

$$\text{(pino)} \quad \circledcirc\,(\rho).\sigma|\sigma_0(\!|P|\!) \rightarrow \sigma|\sigma_0(\!|\rho(\!|\diamond|\!) \circ P|\!)$$

Figure 5.1: The phago/exo/pino (PEP) calculus: syntax and semantics

two elements, namely $\phi_n, \phi_n^{\perp}, \varepsilon_n$ and $\varepsilon_n^{\perp}$ are translated into $\phi \cdot n, \phi^{\perp} \cdot n, \varepsilon \cdot n$ and $\varepsilon^{\perp} \cdot n$, respectively. Finally, pinocytosis $\circledcirc$ is translated into $\circledcirc \in \mathcal{E}$.

The encodings of the operands and of the action parameters follow in the sequence the encodings of the corresponding operators and actions, respectively, and are delimited by symbols acting as separators. The set of symbols returned by the encoding contains all these separators. Consider for example the simple PEP system $\diamond \circ \diamond$. The encoding translates it into a CLS sequence composed by a *circ* symbol followed by the encoding of the two operands of $\circ$, namely the two units $\diamond$. A fresh alphabet symbol is used to separate the three objects, hence we obtain $circ \cdot a \cdot \mathbf{0} \cdot a \cdot \mathbf{0}$ where $a \in \mathcal{E}$ is the separator.

Moreover, the alphabet symbol *act* is used in the result of the encoding as a program counter: during the evolution of the term it precedes every element which encodes a currently active action. In the definition of the encoding $T\{^x/_y\}$ denotes the substitution in T of each occurrence of x with y.

Definition 5.1 (Encoding)**.** *The encoding of a system P of the PEP calculus into CLS is the term $T \in \mathcal{T}$ such that, for some (finite) $E \subset \mathcal{E}$, it holds $\{\![P]\!\} = (T, E)$, where $\{\![\cdot]\!\} : PEP \rightarrow \mathcal{T} \times \mathcal{P}(\mathcal{E})$ is given by the following recursive definition:*

$$\{\![\diamond]\!\} = \big(act \cdot \mathbf{0}, \varnothing\big)$$

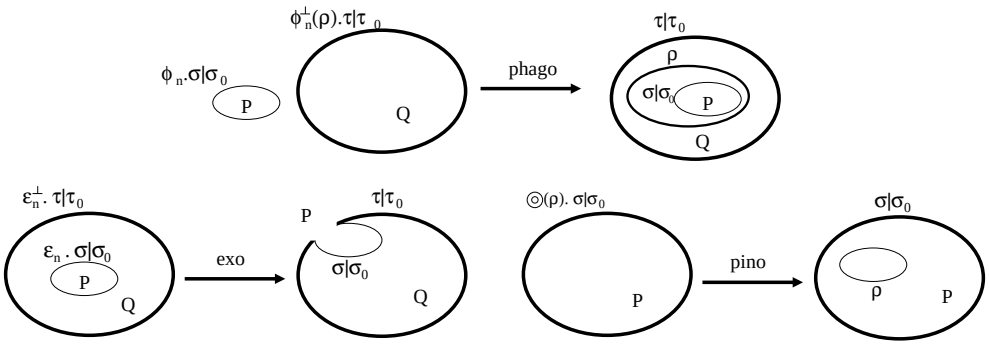


Figure 5.2: Pictorial representation of phagocytosis, exocytosis and pinocytosis

$$\{\![P_1 \circ P_2]\!\} = \big(act \cdot circ \cdot a \cdot P_1'\{^{\epsilon}/_{act}\} \cdot a \cdot P_2'\{^{\epsilon}/_{act}\}, \{a\} \cup E_1 \cup E_2\big)$$
$$\textit{where } \{\![P_i]\!\} = (P_i', E_i), E_1 \cap E_2 = \varnothing \textit{ and } a \in \mathcal{E} \setminus (E_1 \cup E_2)$$

$$\{\![!P]\!\} = \big(act \cdot bangS \cdot P'\{^{\epsilon}/_{act}\}, E\big) \qquad \textit{where } \{\![P]\!\} = (P', E)$$

$$\{\![\sigma(\!|P|\!)]\!\} = \big(act \cdot brane \cdot a \cdot \sigma'\{^{\epsilon}/_{act}\} \cdot a \cdot P'\{^{\epsilon}/_{act}\}, \{a\} \cup E_P \cup E_\sigma\big)$$
$$\textit{where } \{\![P]\!\} = (P', E_P), [\![\sigma]\!] = (\sigma', E_\sigma),$$
$$a \in \mathcal{E} \setminus (E_P \cap E_\sigma) \textit{ and } E_P \cap E_\sigma = \varnothing$$

where $[\![\cdot]\!] : Branes \to \mathcal{T} \times \mathcal{P}(\mathcal{E})$ *is given by the following recursive definition:*

$$[\![\mathbf{0}]\!] = \big(act \cdot \mathbf{0}, \varnothing\big)$$

$$[\![\sigma_1|\sigma_2]\!] = \big(act \cdot par \cdot a \cdot \sigma_1'\{^{\epsilon}/_{act}\} \cdot a \cdot \sigma_2'\{^{\epsilon}/_{act}\} \cdot a, E_1 \cup E_2 \cup \{a\}\big)$$
$$\textit{where } [\![\sigma_i]\!] = (\sigma_i', E_i), E_1 \cap E_2 = \varnothing \textit{ and } a \in \mathcal{E} \setminus (E_1 \cup E_2)$$

$$[\![!\sigma]\!] = \big(act \cdot bangB \cdot a \cdot \sigma'\{^{\epsilon}/_{act}\} \cdot a, E \cup \{a\}\big)$$
$$\textit{where } [\![\sigma]\!] = (\sigma', E) \textit{ and } a \in \mathcal{E} \setminus E$$

$$[\![\phi_n.\sigma]\!] = \big(act \cdot \phi \cdot n \cdot a \cdot \sigma'\{^{\epsilon}/_{act}\} \cdot a, E \cup \{a\}\big)$$
$$\textit{where } [\![\sigma]\!] = (\sigma', E) \textit{ and } a \in \mathcal{E} \setminus E$$

$$[\![\phi_n^{\perp}(\rho).\sigma]\!] = \big(act \cdot \phi^{\perp} \cdot n \cdot a \cdot \rho'\{^{\epsilon}/_{act}\} \cdot a \cdot \sigma'\{^{\epsilon}/_{act}\} \cdot a, E_\rho \cup E_\sigma \cup \{a\}\big)$$
$$\textit{where } [\![\rho]\!] = (\rho', E_\rho), [\![\sigma]\!] = (\sigma', E_\sigma) \textit{ and } a \in \mathcal{E} \setminus (E_\rho \cup E_\sigma)$$

$$[\![\varepsilon_n.\sigma]\!] = \big(act \cdot \varepsilon \cdot n \cdot a \cdot \sigma'\{^{\epsilon}/_{act}\} \cdot a, E \cup \{a\}\big)$$
$$\textit{where } [\![\sigma]\!] = (\sigma', E) \textit{ and } a \in \mathcal{E} \setminus E$$

$$[\![\varepsilon_n^{\perp}.\sigma]\!] = \big(act \cdot \varepsilon^{\perp} \cdot n \cdot a \cdot \sigma'\{^{\epsilon}/_{act}\} \cdot a, E \cup \{a\}\big)$$
$$\textit{where } [\![\sigma]\!] = (\sigma', E) \textit{ and } a \in \mathcal{E} \setminus E$$

$$[\![\circledcirc(\rho).\sigma]\!] = \big(act \cdot \circledcirc \cdot a \cdot \rho'\{^{\epsilon}/_{act}\} \cdot a \cdot \sigma'\{^{\epsilon}/_{act}\} \cdot a, E_\rho \cup E_\sigma \cup \{a\}\big)$$
$$\textit{where } [\![\rho]\!] = (\rho', E_\rho), [\![\sigma]\!] = (\sigma', E_\sigma) \textit{ and } a \in \mathcal{E} \setminus (E_\rho \cup E_\sigma)$$

$$\left(act \cdot par \cdot x \cdot \widetilde{y} \cdot x \cdot \widetilde{z} \cdot x \cdot \widetilde{w}\right)^L \rfloor X \;\mapsto\; \left(act \cdot \widetilde{y} \cdot act \cdot \widetilde{z} \cdot \widetilde{w}\right)^L \rfloor X \qquad \text{(par)}$$

$$act \cdot circ \cdot x \cdot \widetilde{y} \cdot x \cdot \widetilde{z} \;\mapsto\; act \cdot \widetilde{y} \,|\, act \cdot \widetilde{z} \qquad \text{(circ)}$$

$$act \cdot brane \cdot x \cdot \widetilde{y} \cdot x \cdot \widetilde{z} \;\mapsto\; \left(act \cdot \widetilde{y}\right)^L \rfloor act \cdot \widetilde{z} \qquad \text{(brane)}$$

$$x \cdot \widetilde{w} \,|\, act \cdot \mathbf{0} \;\mapsto\; x \cdot \widetilde{w} \qquad \text{(sc1)}$$

$$act \cdot bangS \cdot \mathbf{0} \;\mapsto\; act \cdot \mathbf{0} \qquad \text{(sc2)}$$

$$\left(act \cdot \mathbf{0}\right)^L \rfloor act \cdot \mathbf{0} \;\mapsto\; act \cdot \mathbf{0} \qquad \text{(sc3)}$$

$$\left(act \cdot \mathbf{0} \cdot x \cdot \widetilde{w}\right)^L \rfloor X \;\mapsto\; \left(x \cdot \widetilde{w}\right)^L \rfloor X \qquad \text{(sc4)}$$

$$\left(act \cdot bangB \cdot \mathbf{0} \cdot \widetilde{w}\right)^L \rfloor X \;\mapsto\; \left(act \cdot \mathbf{0} \cdot \widetilde{w}\right)^L \rfloor X \qquad \text{(sc5)}$$

$$\begin{aligned}\left(act \cdot \phi^{\perp} \cdot x_n \cdot x \cdot \widetilde{y} \cdot x \cdot \widetilde{z} \cdot x \cdot \widetilde{w}\right)^L \rfloor X \,|\, \left(act \cdot \phi \cdot x_n \cdot x' \cdot \widetilde{y'} \cdot x' \cdot \widetilde{z'}\right)^L \rfloor Y \\ \mapsto\; \left(act \cdot \widetilde{z} \cdot \widetilde{w}\right)^L \rfloor \left(X \,|\, \left(act \cdot \widetilde{y}\right)^L \rfloor \left(act \cdot \widetilde{y'} \cdot \widetilde{z'}\right)^L \rfloor Y\right)\end{aligned} \qquad \text{(phago)}$$

$$\begin{aligned}\left(act \cdot \varepsilon^{\perp} \cdot x_n \cdot x \cdot \widetilde{y} \cdot x \cdot \widetilde{z}\right)^L \rfloor \left(X \,|\, \left(act \cdot \varepsilon \cdot x_n \cdot x' \cdot \widetilde{y'} \cdot x' \cdot \widetilde{z'}\right)^L \rfloor Y\right) \\ \mapsto\; Y \,|\, \left(act \cdot \widetilde{y} \cdot \widetilde{z} \cdot act \cdot \widetilde{y'} \cdot \widetilde{z'}\right)^L \rfloor X\end{aligned} \qquad \text{(exo)}$$

$$\left(act \cdot \circledcirc \cdot x \cdot \widetilde{y} \cdot x \cdot \widetilde{z} \cdot x \cdot \widetilde{w}\right)^L \rfloor X \;\mapsto\; \left(act \cdot \widetilde{z} \cdot \widetilde{w}\right)^L \rfloor \left(X \,|\, \left(act \cdot \widetilde{y}\right)^L\right) \qquad \text{(pino)}$$

$$act \cdot bangS \cdot \widetilde{x} \;\mapsto\; act \cdot bangS \cdot \widetilde{x} \,|\, act \cdot \widetilde{x} \qquad \text{(bangs)}$$

$$\left(act \cdot bangB \cdot x \cdot \widetilde{y} \cdot x \cdot \widetilde{w}\right)^L \rfloor X \;\mapsto\; \left(act \cdot bangB \cdot x \cdot \widetilde{y} \cdot x \cdot act \cdot \widetilde{y} \cdot \widetilde{w}\right)^L \rfloor X \qquad \text{(bangb)}$$

Figure 5.3: Rewrite rules associated with the encoding of the PEP calculus

In Figure 5.3 we give the rewrite rules which are applicable to encoded PEP systems.

Rules are conceptually of two kinds. Rules from rule (par) to rule (sc5) rearrange elementary CLS sequences encoding PEP systems and membranes, into CLS terms (containing all CLS operators) and simplifying them accordingly to structural congruence on PEP terms. We denote with $\mathcal{R}_{\langle\rangle}$ this set of rules. Rules from rule (phago) to rule (bangB) correspond to PEP semantics. In particular, rules (phago), (exo) and (pino) correspond to phagocytosis, exocytosis and pinocytosis, respectively, and rules (bangS) and (bangB) correspond to structural congruence for the replication operator. Note that element variables are used repeatedly in rules to match exactly the symbols introduced as separators and identify exactly the subsequences representing the encoding of operands and action parameters.

We remark that by applying rules in $\mathcal{R}_{\langle\rangle}$ to the encoding of a PEP system P we obtain a term T in which each membrane system $\sigma(\!|P'|\!)$ in P is represented by a looping sequence

in T, and each occurrence of $\circ$ in P is represented by an occurrence of $\mid$ in T.

Example 5.2. Let us consider the PEP system $!\big(P\big)$ where $P = \phi_n(\!| \diamond |\!) \circ \phi_n^{\perp}(\mathbf{0})(\!| \diamond |\!)$. According to the semantics of the calculus the system may evolve as follows:

$$!(P) \;\equiv\; !(P) \circ \phi_n(\!| \diamond |\!) \circ \phi_n^{\perp}(\mathbf{0})(\!| \diamond |\!) \;\dot{\rightarrow}\; !(P) \circ \mathbf{0}(\!|\mathbf{0}(\!|\mathbf{0}(\!| \diamond |\!)|\!) \circ \diamond|\!) \;\equiv\; !(P)$$

By applying the encoding to the system we obtain the following term T:

$$act \cdot bangS \cdot circ \cdot e \cdot brane \cdot b \cdot \phi \cdot n \cdot a \cdot \mathbf{0} \cdot a \cdot b \cdot \mathbf{0} \cdot e \cdot brane \cdot d \cdot \phi^{\perp} \cdot n \cdot c \cdot \mathbf{0} \cdot c \cdot \mathbf{0} \cdot c \cdot d \cdot \mathbf{0}$$

which may evolve as follows:

$$\begin{aligned}
T \;&\xrightarrow{\square}\; T \mid act \cdot circ \cdot e \cdot brane \cdot b \cdot \phi \cdot n \cdot a \cdot \mathbf{0} \cdot a \cdot b \cdot \mathbf{0} \cdot e \\
&\quad \cdot brane \cdot d \cdot \phi^{\perp} \cdot n \cdot c \cdot \mathbf{0} \cdot c \cdot \mathbf{0} \cdot c \cdot d \cdot \mathbf{0} && (bangS) \\
&\xrightarrow{\square}\; T \mid act \cdot brane \cdot b \cdot \phi \cdot n \cdot a \cdot \mathbf{0} \cdot a \cdot b \cdot \mathbf{0} \\
&\quad \mid act \cdot brane \cdot d \cdot \phi^{\perp} \cdot n \cdot c \cdot \mathbf{0} \cdot c \cdot \mathbf{0} \cdot c \cdot d && (circ) \\
&\xRightarrow{\square}\; T \mid \big(act \cdot \phi \cdot n \cdot a \cdot \mathbf{0} \cdot a\big)^{L} \rfloor act \cdot \mathbf{0} \\
&\quad \big(act \cdot \phi^{\perp} \cdot n \cdot c \cdot \mathbf{0} \cdot c \cdot \mathbf{0} \cdot c\big)^{L} \rfloor act \cdot \mathbf{0} && 2 \times (brane) \\
&\xrightarrow{\square}\; T \mid \big(act \cdot \mathbf{0}\big)^{L} \rfloor (act \cdot \mathbf{0} | \big(act \cdot \mathbf{0}\big)^{L} \rfloor \big(act \cdot \mathbf{0}\big)^{L} \rfloor act \cdot \mathbf{0}) && (phago) \\
&\xRightarrow{\square}\; T
\end{aligned}$$

Now we introduce a normal form for CLS terms which will be used to prove the correctness of the encoding. This normal form can be obtained by applying rules in $\mathcal{R}_{\langle\rangle}$ as long as possible.

Proposition 5.3 (Normal Form)**.** *Assume $\mathcal{R}_{\langle\rangle}$ as the set of rules that can be applied to terms. Given a CLS term T, there exists a unique CLS term (modulo structural congruence), denoted $\langle T\rangle$, such that $T \rightarrow^{*} \langle T\rangle$ and $\langle T\rangle \overset{\square}{\not\rightarrow}$.*

Proof. The term $\langle T\rangle$ is reachable after a finite number of rule applications as all rules in $\mathcal{R}_{\langle\rangle}$ reduce the number of elementary constituents in the term. Moreover, it is easy to see that, by definition of the rules in $\mathcal{R}_{\langle\rangle}$, $\langle T\rangle$ is unique. □

We prove now the correctness of the encoding in terms of soundness and completeness. For the sake of simplicity, let us denote with $\{\!|P|\!\}$ and $[\![\sigma]\!]$ the terms obtained by the application of the encoding to system P and to membrane σ, respectively. Moreover, we denote with $\rightarrow^{*}$ the reflexive and transitive closure of $\rightarrow$ for both CLS and PEP semantics.

Theorem 5.4 (Soundness)**.** *Given a system P of the PEP calculus:*

$$P \rightarrow P' \implies \exists T.\exists P''.\ s.t.\ \{\!|P|\!\} \rightarrow^{*} T, \langle T\rangle \equiv \langle \{\!|P''|\!\}\rangle \text{ and } P'' \equiv P' \ .$$

Proof. Let us first show that the encoding of structurally congruent systems and branes of the PEP Calculus have the same behavior, modulo $\overset{\square}{\rightarrow}$. In particular, for several axioms of the structural congruence it is easy to see that the encoding of the system on the left–hand side has the same normal form of the encoding of the system on the right–hand side.

Moreover, in the case of the axiom $!P \equiv P \circ !P$ we have $\{\!|!P|\!\} \xrightarrow{\Box} \{\!|!P|\!\}|\{\!|P|\!\}$ by applying the (bangs) rule and $\langle\{\!|P \circ !P|\!\}\rangle \equiv \langle\{\!|!P|\!\}|\{\!|P|\!\}\rangle$. In the case of the idempotency law $!!P \equiv !P$ we have $\{\!|!!P|\!\} \xrightarrow{\Box} \{\!|!!P|\!\}|\{\!|!P|\!\}$ by applying again the (bangs) rule: the behavior is as expected (it generates infinitely many copies of P), but it may produce additional copies of $\{\!|!P|\!\}$ in the term.

Commutativity and associativity of $|$ in branes can be reduced to rotations of looping sequences by noting that the normal form of $\{\!|\sigma_1|\sigma_2(\!|P|\!)|\!\}$ is $\left(\langle[\![\sigma_1]\!]\rangle \cdot \langle[\![\sigma_2]\!]\rangle\right)^L \rfloor \langle\{\!|P|\!\}\rangle$. Actually, in the PEP Calculus, commutativity and associativity of $|$ are used only to allow one of the branes in the parallel composition to reach the left–most position, in order to be able to comply with one of the rules of the reaction semantics. After encoding, the same result can be obtained by rotating the looping sequence which represents the whole brane. Finally, the cases of axiom $!\sigma \equiv \sigma|!\sigma$ and of the idempotency law $!!\sigma \equiv !\sigma$ are similar to the cases of the corresponding axioms for systems discussed above.

Now we can prove the theorem by induction on the structure of P without considering the closure under structural congruence. We have the following cases:

- $(P = \diamond)$. In this case neither P nor $\{\!|P|\!\}$ perform any transition.

- $(P = P_1 \circ P_2)$. In this case either one of the two components performs the transition independently and the same transition is performed by P because of the closure under $\_\circ\_$, or P_1 and P_2 interact by performing a (phago) reaction. The proof of the first sub–case is a trivial application of the induction hypothesis, while in the second sub–case we have $P = \phi_n.\sigma|\sigma_0(\!|R|\!) \circ \phi_n^{\perp}(\rho).\tau|\tau_0(\!|Q|\!)$ which performs a reaction into $\tau|\tau_0(\!|\rho(\!|\sigma|\sigma_0(\!|R|\!)|\!) \circ Q|\!)$. By applying the encoding to P we obtain:

$$\begin{aligned}\{\!|P|\!\} \;=\; & act \cdot circ \cdot d \cdot brane \cdot b \cdot \phi \cdot n \cdot d \cdot par \cdot f \cdot [\![\sigma]\!] \cdot f \cdot [\![\sigma_0]\!] \cdot f \cdot d \cdot b \cdot \{\!|R|\!\} \cdot d \\ & \cdot brane \cdot c \cdot \phi^{\perp} \cdot n \cdot e \cdot [\![\rho]\!] \cdot e \cdot par \cdot g \cdot [\![\tau]\!] \cdot g \cdot [\![\tau_0]\!] \cdot g \cdot e \cdot c \cdot \{\!|Q|\!\}\end{aligned}$$

  which, by applying rule (circ) once and rule (brane) twice, reduces to:

$$\begin{aligned}&\left(act \cdot \phi \cdot n \cdot d \cdot par \cdot f \cdot [\![\sigma]\!] \cdot f \cdot [\![\sigma_0]\!] \cdot f \cdot d\right)^L \rfloor \{\!|R|\!\} \;|\; \\ &\qquad\qquad \left(act \cdot \phi^{\perp} \cdot n \cdot e \cdot [\![\rho]\!] \cdot e \cdot par \cdot g \cdot [\![\tau]\!] \cdot g \cdot [\![\tau_0]\!] \cdot g \cdot e\right)^L \rfloor \{\!|Q|\!\}\end{aligned}$$

  which, by applying rule (phago) once and rule (par) twice, reduces to:

$$T \;=\; \left(act \cdot [\![\tau]\!] \cdot act \cdot [\![\tau_0]\!]\right)^L \rfloor \left(act \cdot \{\!|Q|\!\} \;|\; \left(act \cdot [\![\rho]\!]\right)^L \rfloor \left(act \cdot [\![\sigma]\!] \cdot act \cdot [\![\sigma_0]\!]\right)^L \rfloor act \cdot \{\!|R|\!\}\right) .$$

  Now, by applying the encoding to $\tau|\tau_0(\!|\rho(\!|\sigma|\sigma_0(\!|R|\!)|\!) \circ Q|\!)$ we obtain:

$$act \cdot brane \cdot a \cdot [\![\tau|\tau_0]\!] \cdot a \cdot circ \cdot d \cdot brane \cdot b \cdot [\![\rho]\!] \cdot b \cdot brane \cdot c \cdot [\![\sigma|\sigma_0]\!] \cdot c \cdot \{\!|R|\!\} \cdot d \cdot \{\!|Q|\!\}$$

  whose normal form is:

$$\left(act \cdot \langle[\![\tau|\tau_0]\!]\rangle\right)^L \rfloor \left(\left(act \cdot \langle[\![\rho]\!]\rangle\right)^L \rfloor \left(act \cdot \langle[\![\sigma|\sigma_0]\!]\rangle\right)^L \rfloor \langle\{\!|R|\!\}\rangle\right)|\langle\{\!|Q|\!\}\rangle$$

  which is structurally congruent to $\langle T\rangle$.

- $(P = !P_1)$. The semantics of replication is given by the structural congruence relation, then this case has already been discussed.

- $(P = \sigma(\!|P_1|\!))$. We have to consider three sub–cases: in the first P_1 performs the transition independently and the same transition is performed by P because of the closure under $\sigma(\!|\_|\!)$, and we have that the proof is a trivial application of the induction hypothesis. In the second sub–case we have $P = \varepsilon_n^{\perp}.\tau|\tau_0(\!|\varepsilon_n.\sigma|\sigma_0(\!|R|\!) \circ Q|\!)$ which performs a (exo) reaction leading to $R \circ \sigma|\sigma_0|\tau|\tau_0(\!|Q|\!)$. By applying the encoding we obtain:

$$\begin{aligned}\{\!|P|\!\} \;=\; & act \cdot brane \cdot a \cdot \varepsilon^{\perp} \cdot n \cdot b \cdot par \cdot d \cdot [\![\tau]\!] \cdot d \cdot [\![\tau_0]\!] \cdot d \cdot b \cdot a \\ & \cdot circ \cdot c \cdot brane \cdot e \cdot \varepsilon \cdot n \cdot f \cdot par \cdot g \cdot [\![\sigma]\!] \cdot g \cdot [\![\sigma_0]\!] \cdot g \cdot f \cdot e \cdot \{\!|R|\!\} \cdot c \cdot \{\!|Q|\!\}\end{aligned}$$

which, by applying rules (brane), (circ) and again rule (brane), reduces to:

$$\begin{aligned}&\big(act \cdot \varepsilon^{\perp} \cdot n \cdot b \cdot par \cdot d \cdot [\![\tau]\!] \cdot d \cdot [\![\tau_0]\!] \cdot d \cdot b\big)^{L} \rfloor \\ &\qquad \big(\big(act \cdot \varepsilon \cdot n \cdot f \cdot par \cdot g \cdot [\![\sigma]\!] \cdot g \cdot [\![\sigma_0]\!] \cdot g \cdot f\big)^{L} \rfloor act \cdot \{\!|R|\!\} \mid act \cdot \{\!|Q|\!\}\big)\end{aligned}$$

which, by applying rule (exo) once and rule (par) twice, reduces to:

$$T \;=\; act \cdot \{\!|R|\!\} \mid \big(act \cdot [\![\tau]\!] \cdot act \cdot [\![\tau_0]\!] \cdot act \cdot [\![\sigma]\!] \cdot act \cdot [\![\sigma_0]\!]\big)^{L} \rfloor act \cdot \{\!|Q|\!\} \;.$$

Now, by applying the encoding to $R \circ \sigma|\sigma_0|\tau|\tau_0(\!|Q|\!)$ we obtain:

$$act \cdot circ \cdot a \cdot \{\!|R|\!\} \cdot a \cdot brane \cdot b \cdot par \cdot c \cdot par \cdot d \cdot [\![\sigma]\!] \cdot d \cdot [\![\sigma_0]\!] \cdot d \cdot c \cdot par \cdot e \cdot [\![\tau]\!] \cdot e \cdot [\![\tau_0]\!] \cdot e \cdot c \cdot b \cdot \{\!|Q|\!\}$$

whose normal form is:

$$act \cdot \langle \{\!|R|\!\} \rangle \mid \big(act \cdot \langle [\![\tau|\tau_0]\!] \rangle \cdot act \cdot \langle [\![\sigma|\sigma_0]\!] \rangle\big)^{L} \rfloor act \cdot \langle \{\!|Q|\!\} \rangle$$

which is structurally congruent to $\langle T \rangle$.

Finally, in the third sub–case we have $P = \circledcirc(\rho).\sigma|\sigma_0(\!|Q|\!)$ which performs a (pino) reaction into $\sigma|\sigma_0(\!|\rho(\!|\diamond|\!) \circ Q|\!)$. By applying the encoding we obtain:

$$\{\!|P|\!\} \;=\; act \cdot brane \cdot a \cdot \circledcirc \cdot b \cdot [\![\rho]\!] \cdot b \cdot par \cdot c \cdot [\![\sigma]\!] \cdot c \cdot [\![\sigma_0]\!] \cdot c \cdot b \cdot a \cdot \{\!|Q|\!\}$$

which, by applying rules (brane), (pino) and (par), reduces to:

$$T \;=\; \big(act \cdot [\![\sigma]\!] \cdot act \cdot [\![\sigma_0]\!]\big)^{L} \rfloor \big(act \cdot \{\!|Q|\!\} \mid \big(act \cdot [\![\rho]\!]\big)^{L}\big) \;.$$

Now, by applying the encoding to $\sigma|\sigma_0(\!|\rho(\!|\diamond|\!) \circ Q|\!)$ we obtain:

$$act \cdot brane \cdot a \cdot par \cdot b \cdot [\![\sigma]\!] \cdot b \cdot [\![\sigma_0]\!] \cdot b \cdot a \cdot circ \cdot c \cdot brane \cdot d \cdot [\![\rho]\!] \cdot d \cdot \mathbf{0} \cdot c \cdot \{\!|Q|\!\}$$

whose normal form is:

$$\big(act \cdot \langle [\![\sigma|\sigma_0]\!] \rangle\big)^{L} \rfloor \big(\big(act \cdot \langle [\![\rho]\!] \rangle\big)^{L} \mid act \cdot \{\!|Q|\!\}\big)$$

which is structurally congruent to $\langle T \rangle$.

□

Theorem 5.5 (Completeness)**.** *Given a system P of the PEP calculus:*

$$\{\!|P|\!\} \rightarrow^{*} T \implies \exists P' \text{ s.t. } \langle T \rangle \equiv \langle \{\!|P'|\!\} \rangle \text{ and either } P \equiv P' \text{ or } P \rightarrow^{*} P' \;.$$

Proof. We prove the theorem by induction on the number of steps in $\{\!|P|\!\} \stackrel{\square}{\Longrightarrow} T$.

- *Base case.* The case of zero steps in $\{\!|P|\!\} \stackrel{\Box}{\Longrightarrow} T$ is trivial and we have $P' = P$. In the case of one step we have $\{\!|P|\!\} \stackrel{\Box}{\longrightarrow} T$. By looking at the definition of $\{\!|\cdot|\!\}$ we have that the only rules that can be applied to $\{\!|P|\!\}$ are rules (circ), (bangs) and (brane). Since rules (circ) and (brane) are in $\mathcal{R}_{\langle\rangle}$, by applying them to $\{\!|P|\!\}$, we obtain a term T such that $\langle\{\!|P|\!\}\rangle \equiv \langle T\rangle$, and therefore we have $P' = P$. Finally, we can apply rule (bangs) to $\{\!|P|\!\}$ only if $P = !Q$ for some system Q, and $!Q \equiv !Q \circ Q$. Moreover, it is easy to see that $\{\!|!Q|\!\} \stackrel{\Box}{\longrightarrow} \{\!|!Q|\!\}|\{\!|Q|\!\}$ and that $\langle\{\!|!Q \circ Q|\!\}\rangle \equiv \langle\{\!|!Q|\!\} \circ \{\!|Q|\!\}\rangle$. Hence $P' = !Q \circ Q$ verifies the thesis.

- *Inductive step.* We assume that the thesis holds for sequences of transitions of length n. We have to prove the thesis for $n+1$ steps in $\{\!|P|\!\} \stackrel{\Box}{\Longrightarrow} T$. In this case there exists T' such that $\{\!|P|\!\} \stackrel{\Box}{\Longrightarrow} T'$ in n steps and $T' \stackrel{\Box}{\longrightarrow} T$. By applying the induction hypothesis we know that there exists P'' such that $\langle T'\rangle \equiv \langle\{\!|P''|\!\}\rangle$ and either $P \equiv P''$ or $P \rightarrow^* P''$. We prove the inductive step by cases on the rules which could be applied during the transition $T' \stackrel{\Box}{\longrightarrow} T$.

  - If the rule that is applied is one of those in $\mathcal{R}_{\langle\rangle}$ we have that $\langle T'\rangle \equiv \langle T\rangle$ and therefore $P' = P''$ verifies the thesis.
  - If the rule is (bangs), then $P'' = C[!Q]$ for some system Q and some PEP context C. Hence, we have $\langle T'\rangle \equiv \langle\{\!|C[!Q]|\!\}\rangle$ and, since the rule transforms $\{\!|!Q|\!\}$ into $\{\!|!Q|\!\}|\{\!|Q|\!\}$ and $\langle\{\!|!Q|\!\}|\{\!|Q|\!\}\rangle \equiv \langle\{\!|!Q \circ Q|\!\}\rangle$, we have $\langle T\rangle \equiv \langle\{\!|C[!Q \circ Q]|\!\}\rangle$. Hence, since $\{\!|C[!Q]|\!\} \equiv \{\!|C[!Q \circ Q]|\!\}$ we have $P' = C[!Q \circ Q]$.
  - The case of application of rule (bangb) is similar to the previous one but with $P'' = C[!\sigma]$ and therefore $P' = C[!\sigma|\sigma]$.
  - The last cases are those of rules (phago), (exo) and (pino). If one of these rules can be applied to T', then, by the definition of the encoding and since $\langle T\rangle \equiv \langle\{\!|P''|\!\}\rangle$, we have that P'' must be able to perform the corresponding reaction into some P'. In the proof of Theorem 5.4 we have shown that after the application of one of the rules (phago), (exo) or (pino) we obtain a term whose normal form is congruent to the normal form of the encoding of the system reached after the corresponding PEP reaction. Hence we have $\langle T\rangle \equiv \langle\{\!|P'|\!\}\rangle$.

□

5.2 Encoding P Systems

As we did in the previous section for the PEP calculus, in this section we first recall the definition of P Systems, then we describe its translation into CLS.

5.2.1 P Systems

A P system consists of a hierarchy of membranes that do not intersect, with a distinguishable membrane, called the *skin membrane*, surrounding them all. We assume membranes to be labeled by natural numbers. Membranes contain multisets of *objects, evolution rules*

and possibly other membranes. Objects represent molecules swimming in a chemical solution, and evolution rules represent chemical reactions that may occur in the membrane–delimited region containing them. Evolution rules are pairs of multisets of objects, denoted $u \rightarrow v$, describing the reactants and the products of the chemical reactions. Rules in a membrane can be applied only to objects in the same membrane, and they cannot be applied to objects contained in inner membranes. The rules must contain target indications, specifying the membrane where the new objects obtained after applying the rule are sent. The new objects either remain in the same membrane when they have a *here* target, of they pass through membranes, in two directions: the can be sent *out* of the membrane which delimit a region from outside, or can be sent *in* one of the membranes which delimit a region from inside, precisely identified by its label. Given a possibly empty multiset of objects w and a natural number i, the multiset of an evolution rule describing the products of the represented chemical reaction contains messages having one of the following forms:

- $(w, here)$ – the new objects w remain in the same membrane of the applied rule;
- (w, out) – the new objects w are sent outside;
- (w, in_i) – the new objects w are sent into the membrane labeled by i.

A membrane is *dissolved* by the symbol δ resulted after a rule application. When such an action takes place, the membrane disappears, the objects and membranes it contains remain free in the membrane placed immediately outside, and the evolution rules of the dissolved membranes are lost. The skin membrane is never dissolved. The evolution rule is done in parallel, and it could be regulated by *priority* relationships between rules. Parallelism is maximal: at each evolution step a multiset of instances of rewrite rules is chosen non–deterministically such that no other rule can be applied to the system obtained by removing all the objects necessary to apply all the chosen rules. The priority relationships are such that a rule with a smaller priority than another one cannot be chosen for application if the one with the greater priority is applicable. The low–priority rule cannot be chosen even if the high–priority one is not chosen for application: what really matters is the fact that the latter is applicable. The application of the rules consists of removing all the reactants of the chosen rules from the system, adding the products of the rules by taking into account the target indications, and dissolving all the membranes in which a δ object has been produced.

A P System has a tree–structure in which the skin membrane is the root and the membranes containing no other membranes are the leaves. The only change to the structure that may happen is the removal of some node of the tree (apart from the root) caused by some δ object produced by evolution rules. Hence, we assume membranes labels to be unique: they are assigned at the beginning of the evolution by counting the membranes encountered during a breadth-first visit of the tree–structure, with 1 as the label of the skin membrane. A membrane structure can be represented graphically as a Venn diagram.

Now, we formally define P Systems.

Definition 5.6 (P Systems)**.** *A P System is a tuple*

$$\Pi = (V, \mu, w_1, \ldots, w_n, (R_1, \rho_1), \ldots (R_n, \rho_n))$$

where:

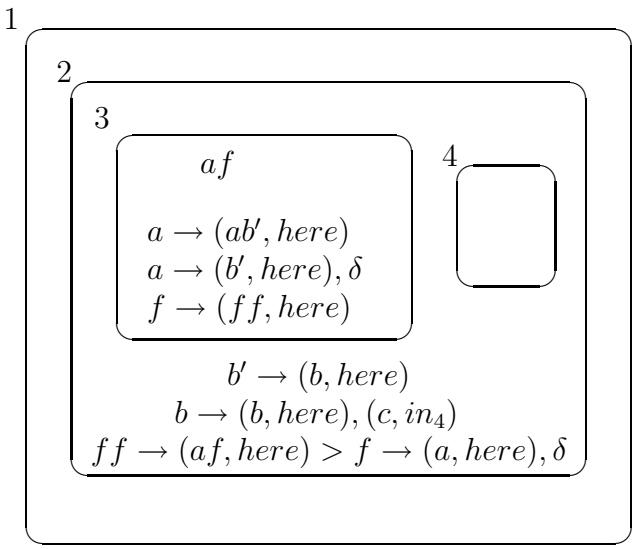


Figure 5.4: Example of a P System generating n^2, with $n \geq 1$.

- V *is an* alphabet *whose elements are called* objects

- $\mu \subset \mathbb{N} \times \mathbb{N}$ *is a* membrane structure*, such that* $(i, j) \in \mu$ *denotes that the membrane labeled by* j *is contained in the membrane labeled by* i*.*

- w_i *with* $1 \leq i \leq n$ *are strings from* V^* *representing multisets over* V *associated with the regions* $1, 2, \ldots, n$ *of* μ*.*

- R_i *with* $1 \leq i \leq n$ *are finite sets of* evolution rules *associated with the regions* $1, 2, \ldots, n$ *of* μ*. An evolution rule is a pair* (u, v) *where* u *is a string over* V *and* v *is a string over* $(V \times \{here, out\}) \cup (V \times \{in_j | 1 \leq j \leq n\}) \cup \{\delta\}$ *and* δ *is a special symbol not in* V*.*

- ρ_i *is a partial order relation over* R_i*, specifying a* priority *relation among the rules:* $(r_1, r_2) \in \rho_1$ *if and only if* $r_1 > r_2$ *(i.e.* r_1 *has an higher priority than* r_2*).*

A typical example of P System is the following system composed by four membranes, which is able to generate in membrane number 4 a multiset of n^2 objects c with n chosen non–deterministically. Details can be found in [58].

Example 5.7. Consider the following P System:

$$\Pi = (V, \mu, w_1, w_2, w_3, w_4, (R_1, \rho_1), (R_2, \rho_2), (R_3, \rho_3), (R_4, \rho_4))$$

where:

$$
\begin{aligned}
&V = \{a, b, b', c, f\}\\
&\mu = \{(1,2),(2,3),(2,4)\}\\
&w_1 = \varnothing, R_1 = \varnothing, \rho_1 = \varnothing\\
&w_2 = \varnothing\\
&R_2 = \{r_1 : b' \rightarrow (b, here), r_2 : b \rightarrow (b, here)(c, in_4),\\
&\qquad r_3 : (ff, here) \rightarrow (af, here), r_4 : f \rightarrow (a, here)\delta\}\\
&\rho_2 = \{r_3 > r_4\}\\
&w_3 = \varnothing, R_3 = \{a \rightarrow (ab, here), a \rightarrow (b', here)\delta, f \rightarrow (ff, here)\}, \rho_3 = \varnothing\\
&w_4 = \varnothing, R_4 = \varnothing, \rho_4 = \varnothing
\end{aligned}
$$

The system is shown in Figure 5.4

5.2.2 Encoding of P Systems into CLS

The translation of P Systems into CLS is rather complicated. In particular, the major difficulty is simulating the maximal parallelism of rule application and the priority notion of P Systems with the sequential rule application mechanism of CLS. For the easy of the presentation, we first give the pseudocode of a simulation algorithm for P Systems, and then we show how the algorithm can be "implemented" in CLS.

The idea of the simulation algorithm is to divide the simulation of a maximally parallel step of the considered P System into three phases. Initially, the algorithm computes the set of all the rules of the system which are applicable, in this phase it takes into account also priorities in order to decide whether a rule is applicable. As a second phase, the algorithm applies rules which are applicable in a sequential way by choosing non–deterministically the next rule to be applied, by removing the objects consumed by the applied rule and by putting the results of the application in a temporary data structure. Finally, the last phase: when no more rules are applicable, the system is updated by copying the contents of the temporary data structure into the effective data structures and by dissolving membranes which must be dissolved. The updating must be performed first by the skin membrane and then by inner membranes in order of nesting (first the parents and then the children in the tree representing the structure of the P System).

For each membrane of the system, the necessary data structures are the following:

- w : the multiset of objects contained in the membrane;

- $next$: the multiset containing the object produced by the application of local rules, and of rules in the outer and inner membranes having some influence on the local membrane by means of some in_i and out target;

- r : a vector of n integers, where n is the number of rules of the membrane. It holds that $r[i] = 1$ if the ith rule is applicable, $r[i] = 0$ if it is not applicable, and $r[i] = -1$ if it is not known wether it is applicable or not.

- $In \subset \mathbb{N}$: the set of the labels of the contained membranes;

The three phases of the simulation algorithm are described in Figures 5.5,5.6, and 5.7 as the check_rules(), apply_rules() and update() procedures. These procedure are executed once for each membrane of the system, each time by using different instances of the data structures described above. We assume $R = \{r_1, \ldots, r_i, \ldots, r_n\}$ to be the set of rules contained in the current membrane. We denote rule r_i with $u_i \rightarrow v_i$, and the restriction of multiset v_i to those object having as target *here*,*out* and in_j as $v_i|_{here}$,$v_i|_{out}$, and $v_i|_{in_j}$, respectively. Finally, we assume that the rule ordering is such that $r_i > r_j \implies i > j$, hence high–priority rules comes before low–priority ones.

In the pseudocode, data structures of a membrane which is not the current one are accessed as follows (consider for example the *next* multiset):

- *parent.next* is the multiset *next* of the membrane containing the current one
- *i.next* is the multiset *next* of the membrane labeled having i as label. Such a membrane is always inside the current one.

Moreover, the label of the current membrane is referred as *this*.

The first two phases of the simulation algorithm can be performed concurrently by the membranes of the system, as they are based on local operations. Actually, in the second phase (procedure apply_rules()) the multiset *next* of the parent and of the contained membranes are incremented. These are not local operations, however they do not create problems. The last phase, instead, must be performed in a synchronized and coordinated manner. This means that a membrane which is ready to update and start the simulation of the next (parallel) evolution step must wait the other membranes to be ready too. In this way, when the simulation of a new evolution step starts, we are sure that all the membranes have updated their data structure by taking into account all the objects received by the outer and inner membranes in the previous step.

Synchronization of the update phase is performed as follows: when a leaf membrane is ready to update its data structures, it sends a synchronization signal to the outside membrane, and waits for a reply. When a membrane receives the signal from all the membranes it contains, it propagates the signal to the upper levels of the nesting tree and waits for the reply. When the signal reaches the skin membrane, such a membrane updates its data structures, sends a reply signal back to the membranes it contains and start simulation of the next step. The contained membranes update their own data structure, propagate down the reply and start the simulation of the next step. When the leaf membranes receive the signal, they update their data structures and start the simulation of the next step. We remark that sending a synchronization signal is a blocking action, hence a membrane does not start simulation of the next step until all the membrane it contains are updated. This avoid, for instance, starting the computation of the set of applicable rules before knowing which are the contained membrane (a contained membrane could dissolve), that is important as a rule producing objects with in_i target is applicable only if a membrane labeled by i exists in the same membrane.

Now, the scheduling of the three phases of the simulation and all the necessary synchronizations are described by the membrane() procedure shown in Figure 5.8. As the previous ones, this procedure is executed by each membrane of the system, apart from the skin which executes the skin() procedure shown in Figure 5.9. Hence, the whole simulation algorithm consists of the concurrent execution of one instance of membrane() for each membrane of the system apart from the skin, and one instance of skin(). Every instance

```
procedure check_rules()
  for all i ∈ 1...n do
    if (u_i ⊆ w)
    and (∀j ∈ 1...i−1 it holds r_j > r_i ⟹ r_j = 0)
    and (∀(v, in_k) ∈ v_i it holds k ∈ In) then
      r[i] := 1
    else
      r[i] := 0
```

Figure 5.5: Phase 1: computing the set of applicable rules.

```
procedure apply_rules()
  if ∃i ∈ 1...n such that r[i] = 1 then
    choose i such that r[i] = 1
    w := w \ u_i
    next := next ∪ v_i|_here
    parent.next := parent.next ∪ v_i|_out
    for all j ∈ In do
      j.next := j.next ∪ v_i|_in_j
    if u_i ⊈ w then
      r[i] := 0
  else
    next := next ∪ w
    w := ∅
```

Figure 5.6: Phase 2: applying rules.

```
procedure update()
  w := next \ all occurrences of δ
  if δ ∈ next then
    parent.In := (parent.In \ this) ∪ In
    parent.w := parent.w ∪ w
    for all j ∈ In do
      j.parent := parent
    dissolve this
  else
    next := ∅
```

Figure 5.7: Phase 3: updating the system.

```
procedure membrane()
  while true do
    if state = Check then
      check_rules()
      state := Run
    else if state = Run then
      apply_rules()
      for all j ∈ In do
        synch(j)
      state := Pause
    else if state = Pause then
      synch(parent)
      state := Stop
    else if state = Stop then
      synch(parent)
      state := Update
    else if state = Update then
      update()
      for all j ∈ In do
        synch(j)
      state := Check
```

Figure 5.8: Scheduling the three phases of the simulation (for a generic membrane).

```
procedure skin()
  while true do
    if state = Check then
      check_rules()
      state := Run
    else if state = Run then
      apply_rules()
      for all j ∈ In do
        synch(j)
      state := Update
    else if state = Update then
      update()
      for all j ∈ In do
        synch(j)
      state := Check
```

Figure 5.9: Scheduling the three phases of the simulation (for the skin).

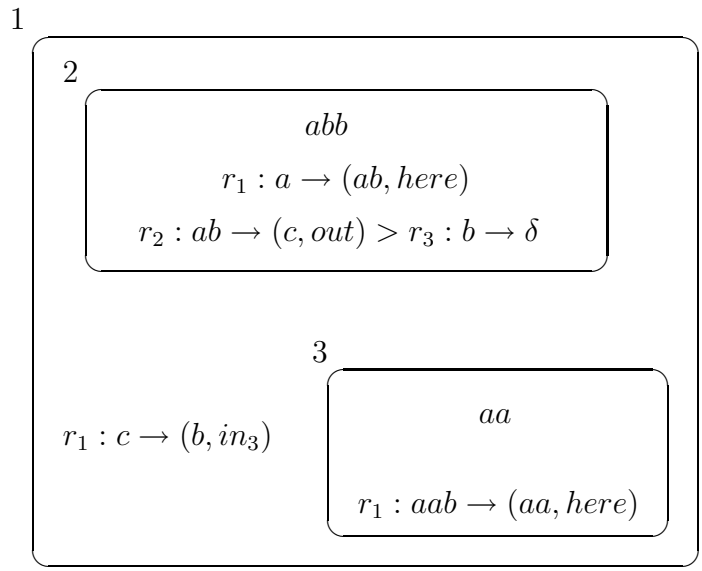


Figure 5.10: An example of P System for the description of the encoding into CLS

of membrane() and skin() will be started by setting the data structures of the membranes coherently with respect to the corresponding initial state in the simulated P System.

Now, a P System is translated into a CLS term representing its structure, and a set of CLS rewrite rules representing the implementation of the simulation algorithm. As regards the structure of the system, the nesting of membranes is modeled as a nesting of looping sequences. The looping sequence corresponding to a membrane is composed by a single element which is exactly the number used as label of the membrane. For the skin, instead of using the number 1 on the looping sequence, we use ϵ.

For example, consider the P System shown in Figure 5.10 (it will be our running example). From the structure of that system we obtain the following nesting of looping sequences:

$$\big(\epsilon\big)^L \rfloor (\ldots \mid \big(2\big)^L \rfloor (\ldots) \mid \ldots \mid \big(3\big)^L \rfloor (\ldots) \mid \ldots)$$

The multiset of objects contained in each membrane is modeled as follows: let $V = \{a_1, \ldots, a_m\}$ be the alphabet of the P System, and let n_i be the number of occurrences of a_i in the multiset, then

$$a_1 \cdot \overbrace{1 \cdot \ldots \cdot 1}^{n_1} \mid \ldots \mid a_m \cdot \overbrace{1 \cdot \ldots \cdot 1}^{n_m}$$

is the term representing the multiset. We choose this representation as it allows us checking wether an object is absent, by checking wether the corresponding symbol if followed by zero 1s. An empty multiset is represented as $a_1 \mid \ldots \mid a_m$.

In every membrane we have also the *next* multiset. In order to keep its elements separated from those of the membrane, we encapsulate such a multiset into a looping sequence as follows:

$$\big(next\big)^L \rfloor (a_1 \cdot 1 \cdot \ldots \cdot 1 \mid \ldots \mid a_m \cdot 1 \cdot \ldots \cdot 1)$$

The vector r used in the algorithms is modeled as a parallel composition of r_i elements, each one possibly followed by either a 0 or a 1 symbol. A symbol r_i folllowed by no other symbols models the situation in which $r[i] = -1$, $r_i \cdot 0$ models $r[i] = 0$ and $r_i \cdot 1$ models $r[i] = 1$. Moreover, we model the In set as a looping sequence containing the parallel composition of the elements of the set. To allow testing the absence of an element, we use another looping sequence modeling the $NotIn$ set, namely the complement of the In set. During the evolution of the system, these two sets will be updated coherently.

To model the transmission of the synchronization signals we use the following technique. A membrane, to synchronize with its children, simply copies the content of the In set into another set called $Wait$. The membrane remains blocked until all the children recognize their own identifiers in such a set, and remove them. This action performed by each child is the synchronization with the parent. When the parent membrane becomes aware that its $Wait$ set is empty, it can continue its execution. Similarly, each child has to wait until its own identifier appears in the $Wait$ set of the parent, then it removes it and continue its execution.

Now we can give the complete CLS term obtained from the structure of the P System in Figure 5.4. The only thing we have not described is the presence in each membrane of a $Check$ element. It is the state of the membrane, and will be modified during execution. The complete CLS term is as follows.

$$
\begin{aligned}
&\big(\epsilon\big)^L \rfloor (Check \ \mid \ a \,|\, b \,|\, c \ \mid \ r_1 \ \mid \\
&\quad \big(In\big)^L \rfloor (2 \,|\, 3) \ \mid \ \big(NotIn\big)^L \rfloor 1 \ \mid \ \big(Wait\big)^L \rfloor \epsilon \ \mid \ \big(next\big)^L \rfloor (a \,|\, b \,|\, c \,|\, \delta) \ \mid \\
&\quad \big(2\big)^L \rfloor (Check \ \mid \ a \cdot 1 \,|\, b \cdot 1 \cdot 1 \,|\, c \ \mid \ r_1 \,|\, r_2 \,|\, r_3 \ \mid \\
&\qquad \big(In\big)^L \rfloor \epsilon \ \mid \ \big(NotIn\big)^L \rfloor (1 \,|\, 2 \,|\, 3) \ \mid \ \big(Wait\big)^L \rfloor \epsilon \ \mid \ \big(next\big)^L \rfloor (a \,|\, b \,|\, c \,|\, \delta) \ ) \ \mid \\
&\quad \big(3\big)^L \rfloor (Check \ \mid \ a \,|\, b \,|\, c \ \mid \ r_1 \ \mid \\
&\qquad \big(In\big)^L \rfloor \epsilon \ \mid \ \big(NotIn\big)^L \rfloor (1 \,|\, 2 \,|\, 3) \ \mid \ \big(Wait\big)^L \rfloor \epsilon \ \mid \ \big(next\big)^L \rfloor (a \,|\, b \,|\, c \,|\, \delta) \ ) \ )
\end{aligned}
$$

The three algorithms described by procedures check_rules(), apply_rules() and update() can be translated into sets of CLS rewrite rules. As regards the check_rules() procedure, we require that the membranes of the system are in state $Check$. The idea is to define a rewrite rule for each evolution rule of the system which checks whether the evolution rule is applicable, and concatenates 1 to the corresponding r_i element. Moreover, for each possible situation in which the rule is not applicable, we define another rewrite rule which concatenates 0 to the corresponding r_i element. The rewrite rules for the running example given in Figure 5.10 are shown in Figure 5.11. Rewrite rules from (C1) to (C3) test the applicability of the rule in the skin membrane, rewrite rules from (C4) to (C11) regard those in membrane 2, and rewrite rules from (C12) to (C15) regard those in membrane 3.

As regards the apply_rules() procedure we require that the membranes of the system are in state Run. The idea is to define a rewrite rule for each evolution rule of the system, that produces the effects of applying the evolution rule and store the results into the appropriate $next$ multiset. Each of these rewrite rules will require that the corresponding r_i element of the membrane is followed by a 1 (i.e. that the rule is applicable). We define also rewrite rules that check wether the evolution rules are not still applicable, and replace the 1 with a 0 after the corresponding r_i element. Finally, when no rules are applicable in a membrane, the whole multiset of the membrane is added to the $next$ multiset. The

$$\big(\epsilon\big)^L \rfloor (X \,|\, Check \,|\, c \cdot 1 \cdot \widetilde{x} \,|\, \big(In\big)^L \rfloor (3 \,|\, Y) \,|\, r_1) \;\mapsto\; \big(\epsilon\big)^L \rfloor (X \,|\, Check \,|\, c \cdot 1 \cdot \widetilde{x} \,|\, \big(In\big)^L \rfloor (3 \,|\, Y) \,|\, r_1 \cdot 1) \tag{C1}$$

$$\big(\epsilon\big)^L \rfloor (X \,|\, Check \,|\, c \,|\, r_1) \;\mapsto\; \big(\epsilon\big)^L \rfloor (X \,|\, Check \,|\, c \,|\, r_1 \cdot 0) \tag{C2}$$

$$\big(\epsilon\big)^L \rfloor (X \,|\, Check \,|\, \big(NotIn\big)^L \rfloor (3 \,|\, Y) \,|\, r_1) \;\mapsto\; \big(\epsilon\big)^L \rfloor (X \,|\, Check \,|\, \big(NotIn\big)^L \rfloor (3 \,|\, Y) \,|\, r_1 \cdot 0) \tag{C3}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, a \cdot 1 \cdot \widetilde{x} \,|\, r_1) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, a \cdot 1 \cdot \widetilde{x} \,|\, r_1 \cdot 1) \tag{C4}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, a \,|\, r_1) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, a \,|\, r_1 \cdot 0) \tag{C5}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, a \cdot 1 \cdot \widetilde{x} \,|\, b \cdot 1 \cdot \widetilde{y} \,|\, r_1 \cdot x \,|\, r_2) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, a \cdot 1 \cdot \widetilde{x} \,|\, b \cdot 1 \cdot \widetilde{y} \,|\, r_1 \cdot x \,|\, r_2 \cdot 1) \tag{C6}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, a \,|\, r_1 \cdot x \,|\, r_2) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, a \,|\, r_1 \cdot x \,|\, r_2 \cdot 0) \tag{C7}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, b \,|\, r_1 \cdot x \,|\, r_2) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, b \,|\, r_1 \cdot x \,|\, r_2 \cdot 0) \tag{C8}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, b \cdot 1 \cdot \widetilde{x} \,|\, r_2 \cdot 0 \,|\, r_3) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, b \cdot 1 \cdot \widetilde{x} \,|\, r_2 \cdot 0 \,|\, r_3 \cdot 1) \tag{C9}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, b \,|\, r_2 \cdot x \,|\, r_3) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, b \,|\, r_2 \cdot x \,|\, r_3 \cdot 0) \tag{C10}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, r_2 \cdot 1 \,|\, r_3) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, r_2 \cdot 1 \,|\, r_3 \cdot 0) \tag{C11}$$

$$\big(3\big)^L \rfloor (X \,|\, Check \,|\, a \cdot 1 \cdot 1 \cdot \widetilde{x} \,|\, b \cdot 1 \cdot \widetilde{y} \,|\, r_1) \;\mapsto\; \big(3\big)^L \rfloor (X \,|\, Check \,|\, a \cdot 1 \cdot 1 \cdot \widetilde{x} \,|\, b \cdot 1 \cdot \widetilde{y} \,|\, r_1 \cdot 1) \tag{C12}$$

$$\big(3\big)^L \rfloor (X \,|\, Check \,|\, a \cdot 1 \,|\, r_1) \;\mapsto\; \big(3\big)^L \rfloor (X \,|\, Check \,|\, a \cdot 1 \,|\, r_1 \cdot 0) \tag{C13}$$

$$\big(3\big)^L \rfloor (X \,|\, Check \,|\, a \,|\, r_1) \;\mapsto\; \big(3\big)^L \rfloor (X \,|\, Check \,|\, a \,|\, r_1 \cdot 0) \tag{C14}$$

$$\big(3\big)^L \rfloor (X \,|\, Check \,|\, b \,|\, r_1) \;\mapsto\; \big(3\big)^L \rfloor (X \,|\, Check \,|\, b \,|\, r_1 \cdot 0) \tag{C15}$$

Figure 5.11: CLS rewrite rules for the phase 1 of the simulation algorithm.

$$\big(\epsilon\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, c \cdot 1 \cdot \widetilde{x} \,|\, \big(3\big)^L \rfloor (Y \,|\, \big(next\big)^L \rfloor (Z \,|\, b \cdot \widetilde{y}))) \;\; \mapsto$$
$$\big(\epsilon\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, c \cdot \widetilde{x} \,|\, \big(3\big)^L \rfloor (Y \,|\, \big(next\big)^L \rfloor (Z \,|\, b \cdot 1 \cdot \widetilde{y}))) \tag{R1}$$

$$\big(\epsilon\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, c) \;\; \mapsto \;\; \big(\epsilon\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, c) \tag{R2}$$

$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, a \cdot 1 \cdot \widetilde{x} \,|\, \big(next\big)^L \rfloor (Y \,|\, a \cdot \widetilde{y} \,|\, b \cdot \widetilde{z})) \;\; \mapsto$$
$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, a \cdot \widetilde{x} \,|\, \big(next\big)^L \rfloor (Y \,|\, a \cdot 1 \cdot \widetilde{y} \,|\, b \cdot 1 \cdot \widetilde{z})) \tag{R3}$$

$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, a) \;\; \mapsto \;\; \big(2\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, a) \tag{R4}$$

$$\big(next\big)^L \rfloor (X \,|\, c \cdot \widetilde{x}) \,|\, \big(2\big)^L \rfloor (Y \,|\, Run \,|\, r_2 \cdot 1 \,|\, a \cdot 1 \cdot \widetilde{y} \,|\, b \cdot 1 \cdot \widetilde{z}) \;\; \mapsto$$
$$\big(next\big)^L \rfloor (X \,|\, c \cdot 1 \cdot \widetilde{x}) \,|\, \big(2\big)^L \rfloor (Y \,|\, Run \,|\, r_2 \cdot 1 \,|\, a \cdot \widetilde{y} \,|\, b \cdot \widetilde{z}) \tag{R5}$$

$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_2 \cdot 1 \,|\, a) \;\; \mapsto \;\; \big(2\big)^L \rfloor (X \,|\, Run \,|\, r_2 \cdot 0 \,|\, a) \tag{R6}$$

$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_2 \cdot 1 \,|\, b) \;\; \mapsto \;\; \big(2\big)^L \rfloor (X \,|\, Run \,|\, r_2 \cdot 0 \,|\, b) \tag{R7}$$

$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_3 \cdot 1 \,|\, b \cdot 1 \cdot \widetilde{x} \,|\, \big(next\big)^L \rfloor (Y \,|\, \delta \cdot \widetilde{y})) \;\; \mapsto$$
$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_3 \cdot 1 \,|\, b \cdot \widetilde{x} \,|\, \big(next\big)^L \rfloor (Y \,|\, \delta \cdot 1 \cdot \widetilde{y})) \tag{R8}$$

$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_3 \cdot 1 \,|\, b) \;\; \mapsto \;\; \big(2\big)^L \rfloor (X \,|\, Run \,|\, r_3 \cdot 0 \,|\, b) \tag{R9}$$

$$\big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, a \cdot 1 \cdot 1 \cdot \widetilde{x} \,|\, b \cdot 1 \cdot \widetilde{y} \,|\, \big(next\big)^L \rfloor (Y \,|\, a \cdot \widetilde{z})) \;\; \mapsto$$
$$\big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, a \cdot \widetilde{x} \,|\, b \cdot \widetilde{y} \,|\, \big(next\big)^L \rfloor (Y \,|\, a \cdot 1 \cdot 1 \cdot \widetilde{z})) \tag{R10}$$

$$\big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, a) \;\; \mapsto \;\; \big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, a) \tag{R11}$$

$$\big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, a \cdot 1) \;\; \mapsto \;\; \big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, a \cdot 1) \tag{R12}$$

$$\big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 1 \,|\, b) \;\; \mapsto \;\; \big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, b) \tag{R13}$$

$$\big(\epsilon\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, x \cdot 1 \cdot \widetilde{x} \,|\, \big(next\big)^L \rfloor (Y \,|\, x \cdot \widetilde{y}) \;\; \mapsto$$
$$\big(\epsilon\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, x \,|\, \big(next\big)^L \rfloor (Y \,|\, x \cdot 1 \cdot \widetilde{x} \cdot \widetilde{y}) \tag{R14}$$

$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, r_2 \cdot 0 \,|\, r_3 \cdot 0 \,|\, x \cdot 1 \cdot \widetilde{x} \,|\, \big(next\big)^L \rfloor (Y \,|\, x \cdot \widetilde{y}) \;\; \mapsto$$
$$\big(2\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, r_2 \cdot 0 \,|\, r_3 \cdot 0 \,|\, x \,|\, \big(next\big)^L \rfloor (Y \,|\, x \cdot 1 \cdot \widetilde{x} \cdot \widetilde{y}) \tag{R15}$$

$$\big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, x \cdot 1 \cdot \widetilde{x} \,|\, \big(next\big)^L \rfloor (Y \,|\, x \cdot \widetilde{y}) \;\; \mapsto$$
$$\big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot 0 \,|\, x \,|\, \big(next\big)^L \rfloor (Y \,|\, x \cdot 1 \cdot \widetilde{x} \cdot \widetilde{y}) \tag{R16}$$

Figure 5.12: CLS rewrite rules for the phase 2 of the simulation algorithm.

$$Update \,|\, x \,|\, \big(next\big)^L \rfloor\, (X \,|\, x \cdot 1 \cdot \widetilde{x}) \;\; \mapsto \;\; Update \,|\, x \cdot 1 \cdot \widetilde{x} \,|\, \big(next\big)^L \rfloor\, (X \,|\, x) \qquad \text{(U1)}$$

Figure 5.13: CLS rewrite rules for the phase 3 of the simulation algorithm.

rewrite rules for the running example are shown in Figure 5.12. Rewrite rules (R1) and (R2) regard the rule in the skin membrane, rewrite rules from (R3) to (R9) regard those in membrane 2, and rewrite rules from (R10) to (R13) regard those in membrane 3. Finally, the rewrite rules from (R14) to (R16) are executed when there is not any evolution rule applicable in each membrane.

As regards the update() procedure, we require that the membranes of the system are in state *Update*. The idea is to copy the content of the *next* multiset to the multiset contained by the membrane. A single rewrite rule is enough for the whole system, and it is shown in Figure 5.13. It copies the occurrences of a single object matched by x, and it will be applied once for each object occurring at least once in *next*. The same rule will be applied in all the membranes.

To conclude the description of the implementation of the simulation algorithm into CLS we have to give the rewrite rules corresponding to the membrane() and skin() procedures. These rewrite rules, for our running example, are given in Figure 5.14. Most of the rules are used for the implementation of both the membrane() and the skin() procedures. The idea is that each rule describe the transition from one state of a membrane to another one. With respect to the algorithms given in Figures 5.8 and 5.9 we add a few more states, namely states *Run_blocked* and *Update_blocked*, that are reached when waiting for a synchronization with the children, and state *Clean*, which is reached after state *Update* and in which the r_i elements are reset. Rewrite rules from (S1) to (S3) describe the transition from state *Check* to state *Run*. It is performed when the last evolution rule has been checked for applicability, and hence when the last r_i element is followed by something. Rewrite rules from (S4) to (S6) describe the transition from state *Run* to state *Pause* (for a generic membrane) or to state *Update* (for the skin) via an intermediate state *Run_blocked* reached to wait the synchronization with the children. The *Run* state is left when the multiset in the membrane is empty. Rewrite rules (S7) and (S8) describe the two synchronizations with the parent performed by a membrane in states *Pause* and *Stop*. Rewrite rule (S9) describe the synchronization with the children performed in the *Update* state, and (S10) the dissolution of a membrane. Note that all the data structures of the dissolved membrane are copied into the parent membrane, and the dissolved membrane disappears. Finally, rewrite rules from (S11) to (S14) describe the activity performed in state *Clean*, namely all the r_i elements are reset.

$$\big(\epsilon\big)^L \rfloor (X \,|\, Check \,|\, r_1 \cdot x) \;\mapsto\; \big(\epsilon\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot x) \tag{S1}$$

$$\big(2\big)^L \rfloor (X \,|\, Check \,|\, r_3 \cdot x) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Run \,|\, r_3 \cdot x) \tag{S2}$$

$$\big(3\big)^L \rfloor (X \,|\, Check \,|\, r_1 \cdot x) \;\mapsto\; \big(3\big)^L \rfloor (X \,|\, Run \,|\, r_1 \cdot x) \tag{S3}$$

$$\begin{gathered} Run \,|\, a \,|\, b \,|\, c \,|\, \big(In\big)^L \rfloor X \,|\, \big(Wait\big)^L \rfloor \epsilon \;\mapsto \\ Run\_blocked \,|\, a \,|\, b \,|\, c \,|\, \big(In\big)^L \rfloor X \,|\, \big(Wait\big)^L \rfloor X \end{gathered} \tag{S4}$$

$$\big(x\big)^L \rfloor (X \,|\, Run\_blocked \,|\, \big(Wait\big)^L \rfloor \epsilon) \;\mapsto\; \big(x\big)^L \rfloor (X \,|\, Pause \,|\, \big(Wait\big)^L \rfloor \epsilon) \tag{S5}$$

$$\big(\epsilon\big)^L \rfloor (X \,|\, Run\_blocked) \;\mapsto\; \big(\epsilon\big)^L \rfloor (X \,|\, Update) \tag{S6}$$

$$\big(Wait\big)^L \rfloor (x \,|\, X) \,|\, \big(x\big)^L \rfloor (Y \,|\, Pause) \;\mapsto\; \big(Wait\big)^L \rfloor X \,|\, \big(x\big)^L \rfloor (Y \,|\, Stop) \tag{S7}$$

$$\big(Wait\big)^L \rfloor (x \,|\, X) \,|\, \big(x\big)^L \rfloor (Y \,|\, Stop) \;\mapsto\; \big(Wait\big)^L \rfloor X \,|\, \big(x\big)^L \rfloor (Y \,|\, Update) \tag{S8}$$

$$\begin{gathered} Update \,|\, \big(In\big)^L \rfloor X \,|\, \big(Wait\big)^L \rfloor \epsilon \,|\, \big(next\big)^L \rfloor (a \,|\, b \,|\, c \,|\, \delta) \;\mapsto \\ Update\_blocked \,|\, \big(In\big)^L \rfloor X \,|\, \big(Wait\big)^L \rfloor X \,|\, \big(next\big)^L \rfloor (a \,|\, b \,|\, c \,|\, \delta) \end{gathered} \tag{S9}$$

$$\begin{gathered} \big(In\big)^L \rfloor (x \,|\, X) \,|\, \big(NotIn\big)^L \rfloor (X' \,|\, Y) \,|\, \big(Wait\big)^L \rfloor (x \,|\, Z) \,| \\ \big(x\big)^L \rfloor (\big(In\big)^L \rfloor X' \,|\, \big(NotIn\big)^L \rfloor Y' \,|\, \big(Wait\big)^L \rfloor Z' \,|\, Update \,|\, a \,|\, b \,|\, c \,| \\ \big(next\big)^L \rfloor (a \,|\, b \,|\, c \,|\, \delta \cdot 1 \cdot \widetilde{x}) \,|\, W) \;\mapsto \\ \big(In\big)^L \rfloor (X \,|\, X') \,|\, \big(NotIn\big)^L \rfloor Y \,|\, \big(Wait\big)^L \rfloor (Z \,|\, Z') \,|\, W \end{gathered} \tag{S10}$$

$$Update\_blocked \,|\, \big(Wait\big)^L \rfloor \epsilon \;\mapsto\; Clean \,|\, \big(Wait\big)^L \rfloor \epsilon \tag{S11}$$

$$\big(\epsilon\big)^L \rfloor (X \,|\, Clean \,|\, r_1 \cdot x) \;\mapsto\; \big(\epsilon\big)^L \rfloor (X \,|\, Check \,|\, r_1) \tag{S12}$$

$$\big(2\big)^L \rfloor (X \,|\, Clean \,|\, r_1 \cdot x \,|\, r_2 \cdot x \,|\, r_3 \cdot x) \;\mapsto\; \big(2\big)^L \rfloor (X \,|\, Check \,|\, r_2 \,|\, r_2 \,|\, r_3) \tag{S13}$$

$$\big(3\big)^L \rfloor (X \,|\, Clean \,|\, r_1 \cdot x) \;\mapsto\; \big(3\big)^L \rfloor (X \,|\, Check \,|\, r_1) \tag{S14}$$

Figure 5.14: CLS rewrite rules for scheduling of the three phases of the simulation algorithm.

Part II

Bisimulation Relations for Biological Systems

Chapter 6

Bisimulations in CLS

In the previous chapter we have introduced the Calculus of Looping Sequences (CLS) as a formalism based on rewrite rules for modeling biological systems. We have given the calculus a semantics describing all the possible evolutions of a term caused by applications of rewrite rules to its subterms. This kind of semantics does not allow component–wise reasoning on the behavior of a term, because the behavior (the semantics) of a composition of terms cannot be inferred by the behavior (the semantics) of the components.

To allow component–wise reasoning, the semantics of a formalism must not describe only what happens inside a component of the system, but also what the component could do by interacting with the environment. This is typically obtained in process calculi by defining semantics based on labeled transition systems (LTSs), where a transition denotes an action performed by the process either internally, or by interacting with the environment. In the latter case, a symbol is used as label of the transition to denote the action performed by the component. At composition time, the behavior of the system can be inferred by the behavior of its components because complementary actions can be observed as labels of the transitions of the components and can be interpreted as an interaction between the components exhibiting them. The ability of inferring the behavior of a systems from the behavior of its components, usually called *compositionality*, could be very useful to verify properties of the system, as it could allow reducing complex properties to be satisfied by the whole system into some simpler ones to be satisfied by each component.

Another advantage of labeled semantics (i.e. of semantics based on LTSs) is that they allow defining behavioral equivalences, which are relations that can be used to compare the behavior of two processes. The two most common examples of behavioral equivalence are *trace equivalence* and *bisimulation*. The former relates processes the LTSs of which produce the same set of traces, the latter relates processes that are step by step able to perform the same set of actions. These equivalence relations are very useful mainly for two reasons: (i) they allow verifying properties of a process by assessing its equivalence with some other process satisfying the properties, and (ii) they allow simplifying and reducing the size of a process by replacing some of its components with simpler ones proved to be equivalent.

As regards CLS, we cannot define a labeled semantics exactly as in process calculi, because CLS is based on rewrite rules rater than on actions, hence we have no actions to be used as transition labels. However, since a transition label should describe a potential

interaction with the environment, and since interactions in CLS are described by rewrite rules having more than one component in their left hand side, we have that we can use as labels exactly the components that are missing in the term to obtain the left hand side of a rule. In other words, in CLS an interactions with the environment can be described as the context in which the current term should be placed in order to enable the application of a rewrite rule, and this context could be used as transition label. For example, given the rule $a \,|\, b \;\mapsto\; c$, we have that a term a can perform the transition

$$a \xrightarrow{\Box \,|\, b} c$$

where $\Box \,|\, b$ is the context in which the rule could be applied ($\Box$ is a placeholder for the current term), and c is the result of the potential application.

This approach of using contexts as labels is not new. Recently it has been used by Sewell [72], and Leifer and Milner [49] to derive transition systems for which bisimulations are congruences from rewrite rules describing the operational semantics of process calculi. In this chapter we show that also bisimulations for CLS are congruences, and this enriches CLS with a powerful tool for verifying properties. Moreover, to continue the comparison of our work with related ones, we define a bisimulation relation for Cardelli's PEP calculus, and we show, through the encoding of PEP into CLS we gave in Section 5.1.2, that this relation is related with the bisimulation relation defined for CLS.

6.1 Labeled Semantics

First of all we introduce some notations that will be used in the definition of the labeled semantics of CLS.

Definition 6.1 ($T' \sqcap T''$ and $T' \setminus T''$)**.** *Assume two terms T' and T'' such that*

$$T' \equiv T_1 \,|\, \ldots \,|\, T_n \,|\, T'_1 \,|\, \ldots \,|\, T'_m \qquad \textit{and} \qquad T'' \equiv T_1 \,|\, \ldots \,|\, T_n \,|\, T''_1 \,|\, \ldots \,|\, T''_o$$

with $T_i, T'_j, T''_k \not\equiv \epsilon$ and $T_i, T'_j, T''_k \equiv T''' \,|\, T''''$ iff either $T''' \equiv \epsilon$, or $T'''' \equiv \epsilon$, for all $0 < i \leq n, 0 < j \leq m, 0 < k \leq o$. We denote with $T' \sqcap T''$ the parallel components shared by the two terms, and with $T' \setminus T''$ the parallel components in T' that are absent in T'', namely

$$T' \sqcap T'' \equiv T_1 \,|\, \ldots \,|\, T_n \qquad \textit{and} \qquad T' \setminus T'' \equiv T'_1 \,|\, \ldots \,|\, T'_m.$$

Now, in order to define a labeled semantics for CLS, we have to introduce contexts, that will be used as transition labels.

Definition 6.2 (Contexts)**.** *Contexts $\mathcal{C}$ are given by the following grammar:*

$$\mathcal{C} ::= \Box \;\;\big|\;\; \mathcal{C} \,|\, T \;\;\big|\;\; T \,|\, \mathcal{C} \;\;\big|\;\; \big(S\big)^L \rfloor \mathcal{C}$$

where $T \in \mathcal{T}$ and $S \in \mathcal{S}$. Context $\Box$ is called the empty context*.*

Some simple examples of CLS contexts are $a \cdot b \,|\, \Box$, $\big(a \cdot b\big)^L \rfloor \Box$, and $\big(a\big)^L \rfloor (b \,|\, \Box)$. By definition, a context contains always a single $\Box$. We denote with $C[T]$ (*context application*) the term obtained by replacing $\Box$ with T in C, and with $C[C']$ (*context composition*) the context obtained by replacing $\Box$ with C' in C. The structural congruence relation and

the operators $\_ \sqcap \_$ and $\_ \setminus \_$ on terms defined above can be trivially extended to contexts by setting $C_1 \equiv C_2$ iff $C[\epsilon] \equiv C_2[\epsilon]$, $C_1 \sqcap C_2 = C_1[\epsilon] \sqcap C_2[\epsilon]$ and $C_1 \setminus C_2 = C_1[\epsilon] \setminus C_2[\epsilon]$.

We introduce also a subclass of contexts that will be used in the definition of the labeled semantics. In these contexts, the empty context $\Box$ cannot be inserted into a looping sequence, hence it can appear only as one of the top–level parallel components.

Definition 6.3 (Parallel contexts)**.** *Parallel contexts $\mathcal{C}_P$ are a subset of contexts given by the following grammar, where $T \in \mathcal{T}$:*

$$\mathcal{C}_P ::= \Box \quad | \quad \mathcal{C}_P \,|\, T \quad | \quad T \,|\, \mathcal{C}_P$$

The following lemma gives a simple property of parallel contexts that will be used in the following.

Lemma 6.4. *Given $T, T' \in \mathcal{T}$ and $C \in \mathcal{C}_P$, it holds $C[T]|T' \equiv C[T|T']$.*

Proof. Since $C \in \mathcal{C}_P$ there exists T_C such that $C[T] \equiv T_C|T$, and moreover we have that $(T_C|T)|T' \equiv T_C|(T|T') \equiv C[T|T']$. □

Contexts and parallel contexts are used in the labeled semantics of CLS.

Definition 6.5 (Labeled Semantics)**.** *Given a set of rewrite rules $\mathcal{R} \subseteq \Re$, the* labeled semantics *of CLS is the labeled transition system given by the following inference rules:*

$$\text{(rule\_appl)}\ \frac{P_1 \mapsto P_2 \in \mathcal{R} \quad C[T''] \equiv P_1\sigma \quad T'' \not\equiv \epsilon \quad \sigma \in \Sigma \quad C \in \mathcal{C}}{T'' \xrightarrow{C} P_2\sigma}$$

$$\text{(cont)}\ \frac{T \xrightarrow{\Box} T'}{\big(S\big)^L \rfloor\, T \xrightarrow{\Box} \big(S\big)^L \rfloor\, T'} \qquad \text{(par)}\ \frac{T \xrightarrow{C} T' \quad C \in \mathcal{C}_P \quad C[\epsilon] \sqcap T'' \equiv \epsilon}{T \,|\, T'' \xrightarrow{C} T' \,|\, T''}$$

where the dual version of the (par) *rule is omitted.*

The labeled semantics is similar to the one in [72] for ground term rewriting. A transition $T \xrightarrow{C} T'$ indicates that the application of the context C to the term T creates a term that matches the left pattern of one of the rewrite rules, and T' is the result of the application of that rule to $C[T]$. In other words, C represents the environment in which T can perform a transition: if $C \equiv \Box$, then T can evolve by performing some internal change, otherwise T can evolve by interacting with some components in the context C.

Rule (rule_appl) describes the (potential) application of a rewrite rule to a term. If there exists a context C such that a rewrite rule can be applied to $C[T'']$, then T'' can perform a transition labeled by C. Note that presence of $T'' \not\equiv \epsilon$ in the premise of the inference rule implies that the context C cannot provide completely the left part of the rewrite rule.

Rule (cont) propagates $\Box$–labeled transitions from the inside to the outside of a looping sequence. Transitions labeled by a non–empty context C cannot be propagated because a looping sequence avoids interactions between its content and its context. As an example, consider the rewrite rule $a \,|\, b \mapsto c$ and the term a. We have $a \xrightarrow{\Box|b} c$. Now, if we insert a into a looping sequence as in $\big(d\big)^L \rfloor\, a$, we obtain that a is no longer allowed to interact with the context $\Box|b$. In other words $\big(d\big)^L \rfloor\, a \not\xrightarrow{\Box|b}$, because the rewrite rule cannot be

applied to $b \mid \left(d\right)^L \rfloor a$. Instead, if we consider the term $T'' = a \mid b$, we have $a \mid b \xrightarrow{\Box} c$, and also $\left(d\right)^L \rfloor (a \mid b) \xrightarrow{\Box} \left(d\right)^L \rfloor c$, as the interaction between a and b can occur completely inside the looping sequence.

Rule (par) propagates to parallel components transitions with parallel contexts as labels. Differently from rule (cont), and from the semantics for ground term rewriting given in [72], some non empty labels can be propagated, as the parallel composition operator (that is commutative) does not forbid its operands to interact with the context. For example, if we have again the rewrite rule $a \mid b \mapsto c$ and the term $a \mid d$, we obtain $a \mid d \xrightarrow{\Box \mid b} c \mid d$, because the context $\Box \mid b$ applied to $a \mid d$ gives the term $a \mid d \mid b$ that is structurally equivalent to $a \mid b \mid d$ that is a term in which the rewrite rule can be applied obtaining exactly $c \mid d$. Moreover, to explain why this holds only for parallel contexts ($C \in \mathcal{C}_P$ is in the premise of (par)) we give the following simple example. Consider the rule $\left(a\right)^L \rfloor b \mapsto c$ and the term b, we have $b \xrightarrow{(a)^L \rfloor \Box} c$. However, if we compose b in parallel with d we obtain $b \mid d \not\xrightarrow{(a)^L \rfloor \Box}$ because the rewrite rule cannot be applied to $\left(a\right)^L \rfloor (b \mid d)$. Finally, the condition $C[\epsilon] \sqcap T'' \equiv \epsilon$ is imposed to ensure that the context used as transition label is always the least necessary to apply the rewrite rule. For example, if we have the rewrite rule $a \mid b \mid c \mapsto d$, we have $a \xrightarrow{\Box \mid b \mid c} d$, but $a \mid b \not\xrightarrow{\Box \mid b \mid c} c \mid b$ because the parallel component b is already present in the term and hence it is not required to be in the environment to apply the rewrite rule. However, $a \mid b \xrightarrow{\Box \mid c} d$ by applying (rule_appl), and $\Box \mid c$ is actually the least context in which the rule can be applied.

The following proposition states that the labeled semantics is equivalent to the standard semantics when the context is empty.

Proposition 6.6. $T \to T' \iff T \xrightarrow{\Box} T'$.

Proof. Trivial: the inference rules of the reaction semantics are the same as the rules of the labeled semantics in particular cases in which C is $\Box$ and $C[T'']$ is exactly the same term $P_1\sigma$. Closure under structural congruence $\equiv$ is assumed explicitly in the reduction semantics while it derives from the use of the structural congruence in the premise of rule (rule_appl) in the labeled semantics. □

The following lemma gives a property of the labeled semantics with respect to context composition that will be used in the following.

Lemma 6.7. $T \xrightarrow{C[C']} T' \iff C'[T] \xrightarrow{C} T'$.

Proof. By induction on the depth of the derivation tree of $T \xrightarrow{C[C']} T'$.

- *Base.* Derivation trees of depth 1 are obtained by rule (rule_appl).

  $T \xrightarrow{C[C']} T' \iff$ there exists $T_1 \mapsto T_1' \in \mathcal{R}$ such that $T_1\sigma = C[C'[T]]$ and $T_1'\sigma = T'$ for some instantiation function $\sigma \iff C'[T] \xrightarrow{C} T'$.

- *Induction step.* We assume that the thesis holds for depth n.

  - (par). We first prove the direction $\Longrightarrow$. Let us assume $T = T_1 | T_2$; then $T' = T_1' | T_2$, $T_1 \xrightarrow{C[C']} T_1'$ and $C[C'] \in \mathcal{C}_P$. We have $C'[T_1] \xrightarrow{C} T_1'$ by induction

hypothesis, which implies $C'[T_1]|T_2 \xrightarrow{C} T_1'|T_2$ (by applying rule (par)), and hence $C'[T] \xrightarrow{C} T'$, since $T' = T_1'|T_2$, $C' \in \mathcal{C}_P$ and by Lemma 6.4. The direction $\Longleftarrow$ can be proved symmetrically.

- (cont). This case is trivial because $C[C'] = \Box$.

□

We denote with $\stackrel{\Box}{\Longrightarrow}$ a sequence of zero or more transitions $\xrightarrow{\Box}$, that is $T \stackrel{\Box}{\Longrightarrow} T'$ if and only if either $T \equiv T'$ or there exist $T_1, \ldots, T_n \in \mathcal{T}$ such that $T \xrightarrow{\Box} T_1 \xrightarrow{\Box} \ldots \xrightarrow{\Box} T_n \xrightarrow{\Box} T'$, and with $\stackrel{C}{\Longrightarrow}$, where $C \neq \Box$, the sequence of transitions such that $T \stackrel{C}{\Longrightarrow} T'$ if and only if there exist $T_1, T_2 \in \mathcal{T}$ such that $T \stackrel{\Box}{\Longrightarrow} T_1 \xrightarrow{C} T_2 \stackrel{\Box}{\Longrightarrow} T'$. We have two lemmas.

Lemma 6.8. *If one of the following two conditions holds:*
(i) $C, C' \in \mathcal{C}_P$,
(ii) $C = \Box$, $C' \in \mathcal{C}$,
then $T \stackrel{C}{\Longrightarrow} T' \iff C'[T] \stackrel{C}{\Longrightarrow} C'[T']$.

Proof. By definition of $\stackrel{C}{\Longrightarrow}$ and of the labeled semantics. □

Lemma 6.9. $T \stackrel{C[C']}{\Longrightarrow} T' \iff C'[T] \stackrel{C}{\Longrightarrow} T'$.

Proof. First of all, it is worth noticing that, by Lemma 6.8, $T \stackrel{\Box}{\Longrightarrow} T' \iff C[T] \stackrel{\Box}{\Longrightarrow} C[T']$ for any context C. Now, $T \stackrel{C[C']}{\Longrightarrow} T' \iff$ there exist T_1, T_2 such that $T \stackrel{\Box}{\Longrightarrow} T_1 \xrightarrow{C[C']} T_2 \stackrel{\Box}{\Longrightarrow} T'$. By Lemma 6.7, we have that $C'[T_1] \xrightarrow{C} T_2$, and hence $C'[T] \stackrel{\Box}{\Longrightarrow} C'[T_1] \xrightarrow{C} T_2 \stackrel{\Box}{\Longrightarrow} T'$, that is $C'[T] \stackrel{C}{\Longrightarrow} T'$. □

6.2 Strong and Weak Bisimulations

In this section we introduce strong and weak bisimilarities on CLS terms, and we show them to be congruences. These relations can be used to compare the behavior of two terms that may evolve by means of applications of rewrite rules from a set which is the same for both terms. The congruence results are very important, as they allow componentwise verification of properties of systems. Moreover, we introduce a notion of *CLS system* as a pair composed by a term and a set of rewrite rules, and we define bisimiulations on systems. These new relations can be used to compare the behavior of two terms that may evolve by means of applications of two distinct sets of rewrite rules. Unfortunately, bisimilarities on CLS systems are not congruences (and we shall give a example to prove this negative result), however, they can be used to verify properties on whole models, and we shall use them also to define an equivalence relation on sets of rewrite rules. Concluding, we will show an example of application of some of the defined bisimilarities to the model of the lactose operon example introduced in Section 3.4. We will use the weak bisimilarity on terms to obtain a (slightly) more succinct model of the phenomenon, and the weak bisimilarity on systems to prove a causality property.

We introduce the notion of *strong bisimilarity* between CLS terms. The definition is standard.

Definition 6.10 (Strong Bisimulation)**.** *A binary relation R on terms is a* strong bisimulation *if, given T_1, T_2 such that $T_1 R T_2$, the two following conditions hold:*

$T_1 \xrightarrow{C} T_1' \implies \exists T_2'$ *such that* $T_2 \xrightarrow{C} T_2'$ *and* $T_1' R T_2'$

$T_2 \xrightarrow{C} T_2' \implies \exists T_1'$ *such that* $T_1 \xrightarrow{C} T_1'$ *and* $T_2' R T_1'$.

The strong bisimilarity $\sim$ *is the largest of such relations.*

The strong bisimilarity $\sim$ is a congruence with respect to CLS operators.

Theorem 6.11 (Strong Congruence)**.** *The relation $\sim$ is a congruence.*

Proof. We show that

$$\mathcal{S} \stackrel{def}{=} \{\ (C[T_1], C[T_2])\ |T_1 \sim T_2 \text{ and } C \text{ is a context}\}$$

is a bisimulation. First of all, it is worth noting that $\mathcal{S}$ includes $\sim$ because $C[T_1] = T_1$ when $C = \Box$. Moreover, the following holds:

$$T_1 \mathcal{S} T_2 \implies C[T_1] \mathcal{S} C[T_2] \tag{6.1}$$

because $T_1 \mathcal{S} T_2$ implies $\exists C'.T_1 = C'[T_1'], T_2 = C'[T_2']$ for some $T_1', T_2' \in \mathcal{T}$ such that $T_1' \sim T_2'$. Hence $C[C'[T_1']] \mathcal{S} C[C'[T_2']]$, that is $C[T_1] \mathcal{S} C[T_2]$.

Now, since $\sim$ is a symmetric relation, we have only to show that given $T_1 \sim T_2$ the following holds

$$C[T_1] \xrightarrow{C'} T_1' \implies \exists T_2'.C[T_2] \xrightarrow{C'} T_2' \text{ and } T_1' \mathcal{S} T_2'\ .$$

We prove this by induction on the depth of the derivation tree of $C[T_1] \xrightarrow{C'} T_1'$:

Base case

- (rule_appl). In this case there exists $T \mapsto T_1' \in \mathcal{R}$ such that $C'[C[T_1]] \equiv T\sigma$ for some instantiation function σ. This implies $T_1 \xrightarrow{C'[C]} T_1'$ and, since $T_1 \sim T_2$, there exists T_2' such that $T_2 \xrightarrow{C'[C]} T_2'$ with $T_1' \sim T_2'$. Finally, $T_2 \xrightarrow{C'[C]} T_2'$ implies $C[T_2] \xrightarrow{C'} T_2'$ by Lemma 6.7 and $T_1' \sim T_2'$ implies $T_1' \mathcal{S} T_2'$.

Inductive step

- (par). In this case $C = C_1[C_2]$ for some C_2 and where $C_1 = \Box | T$ for some T. Hence, $C[T_1] = C_1[C_2[T_1]]$ and by the premise of the inference rule we obtain $C_2[T_1] \xrightarrow{C'} T_1''$. It follows that $T_1' = C_1[T_1'']$. By applying the induction hypothesis we have that there exists T_2'' such that $C_2[T_2] \xrightarrow{C'} T_2''$ and $T_1'' \mathcal{S} T_2''$, hence, by applying the (par) rule, $C_1[C_2[T_2]] \xrightarrow{C'} C_1[T_2'']$, that is $C[T_2] \xrightarrow{C'} T_2'$. Finally, by the closure of $\mathcal{S}$ to contexts given in (6.1), we have $C_1[T_1''] \mathcal{S} C_1[T_2'']$, that is $T_1' \mathcal{S} T_2'$.
- (cont). In this case $C' = \Box$ and $C = C_1[C_2]$ for some C_2 and where $C_1 = T \rfloor \Box$ for some T. Hence, $C[T_1] = C_1[C_2[T_1]]$ and by the premise of the inference rule we obtain $C_2[T_1] \xrightarrow{\Box} T_1''$. It follows that $T_1' = C_1[T_1'']$. By applying the induction hypothesis we have that there exists T_2'' such that $C_2[T_2] \xrightarrow{\Box} T_2''$ and $T_1'' \mathcal{S} T_2''$, hence, by applying the (cont) rule, $C_1[C_2[T_2]] \xrightarrow{\Box} C_1[T_2'']$, that is $C[T_2] \xrightarrow{\Box} T_2'$. Finally, by the closure of $\mathcal{S}$ to contexts given in (6.1), we have $C_1[T_1''] \mathcal{S} C_1[T_2'']$, that is $T_1' \mathcal{S} T_2'$.

□

Most of the time we want to consider bisimilarity without taking into account system internal moves. This relation is usually called *weak bisimilarity*. We recall that we denote with $\stackrel{\Box}{\Longrightarrow}$ a sequence of zero or more transitions $\stackrel{\Box}{\longrightarrow}$, that is $T \stackrel{\Box}{\Longrightarrow} T'$ if and only if either $T \equiv T'$ or there exist $T_1, \ldots, T_n \in \mathcal{T}$ such that $T \stackrel{\Box}{\longrightarrow} T_1 \stackrel{\Box}{\longrightarrow} \ldots \stackrel{\Box}{\longrightarrow} T_n \stackrel{\Box}{\longrightarrow} T'$, and with $\stackrel{C}{\Longrightarrow}$, where $C \neq \Box$, the sequence of transitions such that $T \stackrel{C}{\Longrightarrow} T'$ if and only if there exist $T_1, T_2 \in \mathcal{T}$ such that $T \stackrel{\Box}{\Longrightarrow} T_1 \stackrel{C}{\longrightarrow} T_2 \stackrel{\Box}{\Longrightarrow} T'$.

Definition 6.12 (Weak Bisimulation)**.** *A binary relation R on terms is a* weak bisimulation *if, given T_1, T_2 such that $T_1 R T_2$, the two following conditions hold:*

$T_1 \stackrel{C}{\longrightarrow} T_1' \implies \exists T_2'$ *such that* $T_2 \stackrel{C}{\Longrightarrow} T_2'$ *and* $T_1' R T_2'$

$T_2 \stackrel{C}{\longrightarrow} T_2' \implies \exists T_1'$ *such that* $T_1 \stackrel{C}{\Longrightarrow} T_1'$ *and* $T_2' R T_1'$.

The weak bisimilarity $\approx$ *is the largest of such relations.*

As the strong bisimilarity, also the weak bisimilarity on terms is a congruence.

Theorem 6.13 (Weak Congruence)**.** *The relation $\approx$ is a congruence.*

Proof. We show that

$$\mathcal{S} \stackrel{def}{=} \{\, (C[T_1], C[T_2]) \mid T_1 \approx T_2 \text{ and } C \text{ is a context} \}$$

is a weak bisimulation. First of all it is worth noting that $\mathcal{S}$ includes $\approx$ because $C[T_1] = T_1$ when $C = \Box$. Moreover, the following holds:

$$T_1 \mathcal{S} T_2 \implies C[T_1] \mathcal{S} C[T_2] \tag{6.2}$$

because $T_1 \mathcal{S} T_2$ implies $\exists C'. T_1 = C'[T_1'], T_2 = C'[T_2']$ for some $T_1', T_2' \in \mathcal{T}$ such that $T_1' \approx T_2'$. Hence $C[C'[T_1']] \mathcal{S} C[C'[T_2']]$ that is $C[T_1] \mathcal{S} C[T_2]$.

Now, since $\approx$ is a symmetric relation, we have only to show that given $T_1 \approx T_2$ the following holds:

$$C[T_1] \stackrel{C'}{\longrightarrow} T_1' \implies \exists T_2'. C[T_2] \stackrel{C'}{\Longrightarrow} T_2' \text{ and } T_1' \mathcal{S} T_2' \,.$$

We prove this by induction on the depth of the derivation tree of $C[T_1] \stackrel{C'}{\longrightarrow} T_1'$:

Base case

- (rule_appl). In this case there exists $T \mapsto T_1' \in \mathcal{R}$ such that $C'[C[T_1]] \equiv T\sigma$ for some instantiation function σ. This implies $T_1 \stackrel{C'[C]}{\longrightarrow} T_1'$ and, since $T_1 \approx T_2$, there exists T_2' such that $T_2 \stackrel{C'[C]}{\Longrightarrow} T_2'$ with $T_1' \approx T_2'$. Finally, $T_2 \stackrel{C'[C]}{\Longrightarrow} T_2'$ implies $C[T_2] \stackrel{C'}{\Longrightarrow} T_2'$ by Lemma 6.9 and $T_1' \approx T_2'$ implies $T_1' \mathcal{S} T_2'$.

Inductive step

- (par). In this case $C = C_1[C_2]$ for some C_2 and where $C_1 = \Box | T$ for some T. Hence, $C[T_1] = C_1[C_2[T_1]]$ and by the premise of the inference rule we obtain $C_2[T_1] \stackrel{C'}{\longrightarrow} T_1''$. It follows that $T_1' = C_1[T_1'']$. By applying the induction hypothesis we have that there exists T_2'' such that $C_2[T_2] \stackrel{C'}{\Longrightarrow} T_2''$ and $T_1'' \mathcal{S} T_2''$, hence, by Lemma 6.8, $C_1[C_2[T_2]] \stackrel{C'}{\Longrightarrow} C_1[T_2'']$, that is $C[T_2] \stackrel{C'}{\Longrightarrow} T_2'$. Finally, by the closure of $\mathcal{S}$ to contexts given in (6.2), we have $C_1[T_1''] \mathcal{S} C_1[T_2'']$, that is $T_1' \mathcal{S} T_2'$.

- (cont). In this case $C' = \Box$ and $C = C_1[C_2]$ for some C_2 and where $C_1 = T \rfloor \Box$ for some T. Hence, $C[T_1] = C_1[C_2[T_1]]$ and by the premise of the inference rule we obtain $C_2[T_1] \xrightarrow{\Box} T_1''$. It follows that $T_1' = C_1[T_1'']$. By applying the induction hypothesis we have that there exists T_2'' such that $C_2[T_2] \stackrel{\Box}{\Longrightarrow} T_2''$ and $T_1'' \mathcal{S} T_2''$, hence, by Lemma 6.8, $C_1[C_2[T_2]] \stackrel{\Box}{\Longrightarrow} C_1[T_2'']$, that is $C[T_2] \stackrel{\Box}{\Longrightarrow} T_2'$. Finally, by the closure of $\mathcal{S}$ to contexts given in (6.2), we have $C_1[T_1'']\mathcal{S}C_1[T_2'']$, that is $T_1'\mathcal{S}T_2'$.

□

Example 6.14. Consider the following set of rewrite rules:

$$\mathcal{R} = \{ \quad a \,|\, b \mapsto c \quad , \quad d \,|\, b \mapsto e \quad , \quad e \mapsto e \quad , \quad c \mapsto e \quad , \quad f \mapsto a \quad \}$$

We have that $a \sim d$, because

$$a \xrightarrow{\Box|b} c \xrightarrow{\Box} e \xrightarrow{\Box} e \xrightarrow{\Box} \ldots \qquad \text{and} \qquad d \xrightarrow{\Box|b} e \xrightarrow{\Box} e \xrightarrow{\Box} \ldots$$

and $f \approx d$, because

$$f \xrightarrow{\Box} a \xrightarrow{\Box|b} c \xrightarrow{\Box} e \xrightarrow{\Box} e \xrightarrow{\Box} \ldots$$

On the other hand, $f \not\sim e$ and $f \not\approx e$, because

$$e \xrightarrow{\Box} e \xrightarrow{\Box} e \xrightarrow{\Box} \ldots$$

One may also be interested in comparing the behavior of terms whose evolution is given by the application of two possibly different sets of rewrite rules. To this aim we define *CLS systems* as pairs consisting of a CLS term and a set of rewrite rules.

Definition 6.15 (System)**.** *A* CLS System *is a pair* $\langle T, \mathcal{R} \rangle$ *with* $T \in \mathcal{T}$, $\mathcal{R} \subseteq \Re$.

Given a system $\langle T, \mathcal{R} \rangle$, we write $\mathcal{R} : T \xrightarrow{C} T'$ to mean that the transition $T \xrightarrow{C} T'$ is performed by applying a rule in $\mathcal{R}$, and we write $\mathcal{R} : T \stackrel{C}{\Longrightarrow} T'$ to mean that the sequence of transitions $T \stackrel{C}{\Longrightarrow} T'$ is performed by applying rules in $\mathcal{R}$. Now, we introduce strong and weak bisimilarities between CLS systems. With abuse of notation we denote such relations with $\sim$ and $\approx$, respectively.

Definition 6.16 (Strong Bisimulation on Systems)**.** *A binary relation* R *on CLS systems is a* strong bisimulation *if, given* $\langle T_1, \mathcal{R}_1 \rangle$ *and* $\langle T_2, \mathcal{R}_2 \rangle$ *such that* $\langle T_1, \mathcal{R}_1 \rangle R \langle T_2, \mathcal{R}_2 \rangle$*, the two following conditions hold:*

$\mathcal{R}_1 : T_1 \xrightarrow{C} T_1' \implies \exists T_2'$ *such that* $\mathcal{R}_2 : T_2 \xrightarrow{C} T_2'$ *and* $\langle T_1', \mathcal{R}_1 \rangle R \langle T_2', \mathcal{R}_2 \rangle$

$\mathcal{R}_2 : T_2 \xrightarrow{C} T_2' \implies \exists T_1'$ *such that* $\mathcal{R}_1 : T_1 \xrightarrow{C} T_1'$ *and* $\langle T_2', \mathcal{R}_2 \rangle R \langle T_1', \mathcal{R}_1 \rangle$.

The strong bisimilarity $\sim$ *is the largest of such relations.*

Definition 6.17 (Weak Bisimulation on Systems)**.** *A binary relation* R *on CLS systems is a* weak bisimulation *if, given* $\langle T_1, \mathcal{R}_1 \rangle$ *and* $\langle T_2, \mathcal{R}_2 \rangle$ *such that* $\langle T_1, \mathcal{R}_1 \rangle R \langle T_2, \mathcal{R}_2 \rangle$*, the two following conditions hold:*

$\mathcal{R}_1 : T_1 \xrightarrow{C} T_1' \implies \exists T_2'$ *such that* $\mathcal{R}_2 : T_2 \stackrel{C}{\Longrightarrow} T_2'$ *and* $\langle T_1', \mathcal{R}_1 \rangle R \langle T_2', \mathcal{R}_2 \rangle$

$\mathcal{R}_2 : T_2 \xrightarrow{C} T_2' \implies \exists T_1'$ *such that* $\mathcal{R}_1 : T_1 \stackrel{C}{\Longrightarrow} T_1'$ *and* $\langle T_2', \mathcal{R}_2 \rangle R \langle T_1', \mathcal{R}_1 \rangle$.

The weak bisimilarity $\approx$ *is the largest of such relations.*

If we fix a set of rewrite rules, strong and weak bisimilarities on CLS systems correspond to strong and weak bisimilarities on terms, respectively. Namely, for a given $\mathcal{R} \in \Re$, $\langle T_1, \mathcal{R}\rangle \sim \langle T_2, \mathcal{R}\rangle$ if and only if $T_1 \sim T_2$ and $\langle T_1, \mathcal{R}\rangle \approx \langle T_2, \mathcal{R}\rangle$ if and only if $T_1 \approx T_2$. However, as we show in the following example, bisimilarity relations introduced for CLS systems are not congruences.

Example 6.18. Consider the following sets of rewrite rules

$$\mathcal{R}_1 = \{a \,|\, b \mapsto c\} \qquad \mathcal{R}_2 = \{a \,|\, d \mapsto c\ ,\ \ b \,|\, e \mapsto c\}$$

We have that $\langle a, \mathcal{R}_1\rangle \approx \langle e, \mathcal{R}_2\rangle$ because

$$\mathcal{R}_1 : a \xrightarrow{\Box|b} c \qquad \mathcal{R}_2 : e \xrightarrow{\Box|b} c$$

and $\langle b, \mathcal{R}_1\rangle \approx \langle d, \mathcal{R}_2\rangle$, because

$$\mathcal{R}_1 : b \xrightarrow{\Box|a} c \qquad \mathcal{R}_2 : d \xrightarrow{\Box|a} c$$

but $\langle a \,|\, b, \mathcal{R}_1\rangle \not\approx \langle e \,|\, d, \mathcal{R}_2\rangle$, because

$$\mathcal{R}_1 : a \,|\, b \xrightarrow{\Box} c \qquad \mathcal{R}_2 : e \,|\, d \not\xrightarrow{\Box}$$

Even if bisimilarity on CLS systems are not congruences, they allow us to define equivalence relations on sets of rewrite rules.

Definition 6.19 (Rules Equivalence)**.** *Two sets of rewrite rules $\mathcal{R}_1$ and $\mathcal{R}_2$ are strongly (resp. weakly) equivalent, denoted $\mathcal{R}_1 \simeq \mathcal{R}_2$ (resp. $\mathcal{R}_1 \cong \mathcal{R}_2$), if and only if for any term $T \in \mathcal{T}$ it holds $\langle T, \mathcal{R}_1\rangle \sim \langle T, \mathcal{R}_2\rangle$ (resp. $\langle T, \mathcal{R}_1\rangle \approx \langle T, \mathcal{R}_2\rangle$).*

Example 6.20. Given $\mathcal{R}_1 = \{a \mapsto c\}$, $\mathcal{R}_2 = \{a \mapsto f\}$ and $\mathcal{R}_3 = \{a \mapsto b\ ,\ \ b \mapsto c\}$, we have that $\mathcal{R}_1 \simeq \mathcal{R}_2$, but $\mathcal{R}_1 \not\simeq \mathcal{R}_3$ and $\mathcal{R}_1 \cong \mathcal{R}_2$.

Now, if we resort to equivalent rules, we can prove congruence results on CLS systems.

Proposition 6.21 (Congruences on Systems)**.** *Given $\mathcal{R}_1 \simeq \mathcal{R}_2$ (resp. $\mathcal{R}_1 \cong \mathcal{R}_2$) and $\langle T, \mathcal{R}_1\rangle \sim \langle T', \mathcal{R}_2\rangle$ (resp. $\langle T, \mathcal{R}_1\rangle \approx \langle T', \mathcal{R}_2\rangle$), for any $C \in \mathcal{C}$ we have $\langle C[T], \mathcal{R}_1\rangle \sim \langle C[T'], \mathcal{R}_2\rangle$ (resp. $\langle C[T], \mathcal{R}_1\rangle \approx \langle C[T'], \mathcal{R}_2\rangle$).*

Proof. Since $\mathcal{R}_1 \simeq \mathcal{R}_2$ we have that $\langle T, \mathcal{R}_1\rangle \sim \langle T, \mathcal{R}_2\rangle$; moreover, by hypothesis, $\langle T, \mathcal{R}_1\rangle \sim \langle T', \mathcal{R}_2\rangle$, and therefore $\langle T, \mathcal{R}_2\rangle \sim \langle T', \mathcal{R}_2\rangle$. Now, since the set of rewrite rules is the same ($\mathcal{R}_2$), by the congruence results for CLS terms, we have $\langle C[T], \mathcal{R}_2\rangle \sim \langle C[T'], \mathcal{R}_2\rangle$. Again, since $\mathcal{R}_1 \simeq \mathcal{R}_2$, we have $\langle C[T], \mathcal{R}_1\rangle \sim \langle C[T], \mathcal{R}_2\rangle$, and hence, $\langle C[T], \mathcal{R}_1\rangle \sim \langle C[T'], \mathcal{R}_2\rangle$. The proof is identical for $\cong$ and $\approx$ instead of $\simeq$ and $\sim$, respectively. □

6.2.1 Bisimulations and E.Coli

Now we use some of the bisimulation relations we have defined to propose a simplification for the model of the gene regulation process we gave in Section 3.4, and to verify a property of the regulation process on that model. Let us denote by T the term $lacP \cdot lacO \cdot lacZ \cdot lacY \cdot lacA \,|\, repr$. Note that T behaves as $lacI{-}A$ apart from the transcription of the

lac Repressor, which is already present. Therefore, the transition system derived from T corresponds the one derived form $lacI{-}A$ apart from some $\Box$–labeled transitions obtained by the application of rule (R1). As a consequence, $T \approx lacI{-}A$. Now, since $\approx$ is a congruence, we may replace $lacI{-}A$ with T in $Ecoli$, thus obtaining an equivalent term.

Now we use the weak bisimulation defined on CLS systems to verify a simple property of the described system, namely that by starting from a situation in which the lac Repressor is bound to gene o, and none of the three enzymes produced by the lactose operon is present (which is a typical stable state of the system), production of such enzymes can start only if lactose appears.

In order to verify this property with the bisimulation relation we defined, we need to modify the rules of the model in such a way that the event of starting the production of the three enzymes becomes observable. We can obtain this result, for instance, by replacing rule (R10) with the rule

$$\begin{aligned}\left(\widetilde{w}\right)^{L} \rfloor \left(\widetilde{x} \cdot RO \cdot \widetilde{y} \,|\, LACT \,|\, X\right) \,|\, START \;&\mapsto \\ \left(\widetilde{w}\right)^{L} \rfloor \left(\widetilde{x} \cdot lacO \cdot \widetilde{y} \,|\, RLACT \,|\, X\right) \qquad &\text{(R10bis)}\end{aligned}$$

We choose to modify (R10) because we know that, after applying rule (R10), the three enzymes can be produced freely, and we add to the rule the interaction with the artificial element $START$ in the environment in order to obtain $\Box|START$ as a transition label every time the rule is applied to the term.

The property we want to verify is satisfied by the system $\langle T_1, \mathcal{R}\rangle$, where $\mathcal{R}$ consists of the following four rules:

$$\begin{array}{llll} T_1 \,|\, LACT \;\mapsto\; T_2 & \text{(R1')} & T_2 \,|\, START \;\mapsto\; T_3 & \text{(R3')} \\ T_2 \,|\, LACT \;\mapsto\; T_2 & \text{(R2')} & T_3 \,|\, LACT \;\mapsto\; T_3 & \text{(R4')} \end{array}$$

for some ground terms T_1, T_2 and T_3.

It can be proved that the system $\langle T_1, \mathcal{R}\rangle$ is weakly bisimilar to the system $\langle EcoliRO, (\mathcal{R}_{lac} \setminus \{R_{10}\}) \cup \{(R_{10bis})\}\rangle$, where:

$$EcoliRO = \left(m\right)^{L} \rfloor lacI' \cdot PP \cdot RO \cdot lacZ \cdot lacY \cdot lacA$$

In particular, the bisimulation relation associates (the system containing) term T_1 with (the system containing) term $EcoliRO$, term T_2 with all the terms representing a bacterium containing at least one molecule of lactose with the Lac repressor bound to gene o, and, finally, term T_3 with all the terms in which the repressor has left gene o and is bound to the lactose.

Chapter 7

Bisimulations in Brane Calculi

In Section 5.1 we recalled the definition of the PEP Calculus (the simplest of Brane Calculi) and we showed how to translate PEP systems into CLS terms. Moreover, in Chapter 6 we defined a labeled semantics for CLS and we used it to define bisimulation relations. As far as we know, labeled semantics for Brane Calculi, and consequently bisimulation relations, have never been defined.

In this Chapter we develop a theory of bisimulations for the PEP calculus: we start by identifying what should be observable in the behavior of a PEP system, then we define a labeled semantics in which these observable things are used as transition labels, and finally we define bisimulation relations. A related work on a similar formalsim is [53], where a complete behavioural theory is developed for Mobile Ambients [14].

Once bisimulation relations for the PEP calculus are defined, we can compare them with those of CLS, by using the encoding given in Section 5.1. We shall give the results of this comparison at the end of this chapter.

7.1 A Labeled Semantics for the PEP Calculus

In process calculi theory, a labeled semantics usually allow describing the potential behavior of a process in terms of possible interactions with other processes that could occur in its environment. This is obtained by allowing the process performing as many transitions as are its active actions, each transition having the corresponding action as label and leading to a new process which corresponds to the result of the execution of the action. Moreover, labeled semantics include silent transitions, often labeled with τ, describing internal activity, namely interactions occurred between internal components of the process. Silent transitions are used also to describe actions that are performed by a single component of the process, without any interaction. Furthermore, if actions of the process calculus require parameters (for instance an action of sending or receiving a message may require the transmitted message as parameter) then also the parameter is shown in the transition label.

In the PEP calculus, actions are phagocytosis, exocytosis and pinocytosis. The first and the second actions describe interaction between two different membranes, while the third is performed by a single membrane (it will cause a silent transition). We use $\_$ as the label of silent transitions, and now we discuss which labels should be used in the labeled semantics.

Phagocytosis and exocytosis can be seen as communications: a membrane which is engulfed by another one can be seen as a membrane sending itself to the other, and a membrane which is expelled by another one can be seen as a membrane sending itself to the external environment. Hence, from the process calculus viewpoint, the message which is transmitted is the continuation of the process which perform the action of sending. As a consequence, for transitions corresponding to ϕ_n and ε_n actions we use as labels pairs $(\phi_n, \sigma(\!|P|\!))$ and $(\varepsilon_n, \sigma(\!|P|\!))$, respectively, in which $\sigma(\!|P|\!)$ is the continuation of the membrane performing ϕ_n and ε_n, respectively.

Now, we have to consider the $\phi_n^\perp$ and the $\varepsilon_n^\perp$ actions, namely the actions of receiving a message. In the first case, the case of phagocytosis, we have that the two membranes performing ϕ_n and $\phi_n^\perp$ are composed by $\circ$, hence each one is in the context of the other, and we can use $(\phi_n^\perp, \sigma(\!|P|\!))$ as label for the $\phi_n^\perp$ when the received message is $\sigma(\!|P|\!)$. In the case of exocytosis, instead, the process performing the ε_n action is not in the context of the one performing $\varepsilon_n^\perp$, but it is inside the membrane performing $\varepsilon_n^\perp$. Hence, the use of $(\varepsilon_n^\perp, \sigma(\!|P|\!))$ as a label for this action would be meaningless, as $\varepsilon_n^\perp$ do not cause a potential interaction with the environment, but a potential internal interaction. After this discussion, we conclude that the set of labels of the labeled semantics of the PEP calculus is $\mathcal{L} = \{(\phi_n, \sigma(\!|R|\!)), (\phi_n^\perp, \sigma(\!|R|\!)), (\varepsilon_n, \sigma(\!|R|\!)), \_ \mid n \in \mathcal{N}, R \in PEP, \sigma \in Branes\}$. Let ℓ range over this set.

Now, a system P that is able to perform a $(\phi_n, \sigma(\!|R|\!))$ transition, $P \xrightarrow{\phi_n, \sigma(\!|R|\!)} P'$, is a system which has a component $\sigma(\!|R|\!)$ that can enter a membrane, while a system Q that is able to perform a $(\phi_n^\perp, \sigma(\!|R|\!))$ transition, $Q \xrightarrow{\phi_n^\perp, \sigma(\!|R|\!)} Q'$, is a system that can engulf another system $\sigma(\!|R|\!)$. When P and Q are composed by $\circ$, they can evolve together, with a silent action, to a new system in which $\sigma(\!|R|\!)$ is inside Q'.

Definition 7.1 (Labeled Semantics)**.** *The labeled semantics of the PEP calculus is given by the labeled transition system generated by inference rules in Figure 7.1. Terms in the rules are considered modulo structural congruence.*

Rules (Ph1), (Ph2) and (Ph3) describe the behavior of systems which can perform phagocytosis. Rule (Ph1) describe the evolution of a system which can be engulfed. Rule (Ph2) describe the evolution of a system which can engulf any other system $\sigma(\!|R|\!)$. Rule (Ph3) describes an actual phagocytosis which involve two systems.

Rules (E1) and (E2) describe the exocytosis process. Rule (E1) describes the behavior of a system which can exit a membrane. Recall that, in the labeled semantics, an action represents the potentiality of a system when inserted in a suitable context, thus the transition $P \xrightarrow{\varepsilon_n, \sigma(\!|R|\!)} P'$, intuitively means that P has a component $\sigma(\!|R|\!)$ that, when inserted in a membrane τ, can abandon its membrane σ and also can get away from τ becoming R. For this reason, there is no corresponding transition with $\varepsilon_n^\perp$ as label. The possibility of a system to allow an internal system to get away does not depend on the context but on the internal state of the system. In fact, rule (E2) states that a system can allow an internal system to exit by a silent action, while membranes σ, τ and τ_0 coalesce (see also Fig. 5.2).

Rule (Pi1) describes the pinocytosis process. Rules (Par1) and (Par2) state that an action of a system P can be observed also when P is composed by $\circ$ with other systems. Finally, rule (Br1) states that actions internal to a membrane cannot be observed from outside, and only the silent action is allowed in such a context.

$$\phi_n.\sigma|\sigma_0(\!|P|\!) \xrightarrow{\phi_n,\sigma|\sigma_0(\!|P|\!)} \diamond \quad \text{(Ph1)}$$

$$\phi_n^{\perp}(\rho).\tau|\tau_0(\!|Q|\!) \xrightarrow{\phi_n^{\perp},\sigma(\!|R|\!)} \tau|\tau_0(\!|\rho(\!|\sigma(\!|R|\!)|\!) \circ Q|\!) \quad \text{(Ph2)}$$

$$\varepsilon_n.\sigma|\sigma_0(\!|P|\!) \xrightarrow{\varepsilon_n,\sigma|\sigma_0(\!|P|\!)} \diamond \quad \text{(E1)}$$

$$\circledcirc(\rho).\sigma|\sigma_0(\!|P|\!) \dashrightarrow \sigma|\sigma_0(\!|P \circ \rho(\!|\diamond|\!)|\!) \quad \text{(Pi1)}$$

$$\frac{P \xrightarrow{\phi_n,\sigma(\!|R|\!)} P' \qquad Q \xrightarrow{\phi_n^{\perp},\sigma(\!|R|\!)} Q'}{P \circ Q \dashrightarrow P' \circ Q'} \quad \text{(Ph3)}$$

$$\frac{P \xrightarrow{\ell} P'}{P \circ Q \xrightarrow{\ell} P' \circ Q} \qquad \frac{Q \xrightarrow{\ell} Q'}{P \circ Q \xrightarrow{\ell} P \circ Q'} \quad \text{(Par1,Par2)}$$

$$\frac{P \xrightarrow{\varepsilon_n,\sigma(\!|R|\!)} P'}{\varepsilon_n^{\perp}.\tau|\tau_0(\!|P \circ Q|\!) \dashrightarrow R \circ \sigma|\tau|\tau_0(\!|Q \circ P'|\!)} \quad \text{(E2)}$$

$$\frac{P \dashrightarrow P'}{\sigma(\!|P|\!) \dashrightarrow \sigma(\!|P'|\!)} \quad \text{(Br1)}$$

Figure 7.1: The phago/exo/pino (PEP) calculus: inference rules for the Labeled Semantics

7.2 Bisimulation Relations

We define strong and weak bisimulation relation on the labeled semantics of the PEP calculus. Usually, (strongly) bisimilar processes must be step by step able to perform transitions with the same labels. In the labeled semantics we have defined for the PEP calculus, labels are not simple objects: they may contain branes which can be arbitrarily complex. These branes that may occur in transition labels will become active parts of the considered PEP system once the transitions having them as labels have been performed.

In the definition of the bisimulation relation we have to take into account this role of the brane used as transition label. We could require that the transitions of two bisimilar systems are exactly the same, but this would be a too strong requirement. Instead, we require that the branes appearing in the transitions of two bisimilar systems are bisimilar too. In such a way we can ensure that when these branes will be activated, they will have the same behavior, even if they are not syntactically identical. This is the approach commonly followed to define bisimulation relations for higher–order process calculi [71].

The strong bisimulation relation for PEP systems is defined as follows.

Definition 7.2 (PEP Strong Bisimulation)**.** *A binary relation κ on PEP terms is a* strong bisimulation *if, given P and Q such that $P\kappa Q$, the following conditions hold:*

$P \dashrightarrow P' \implies \exists Q'$ *such that* $Q \dashrightarrow Q'$ *and* $P'\kappa Q'$

$Q \dashrightarrow Q' \implies \exists P'$ *such that* $P \dashrightarrow P'$ *and* $Q'\kappa P'$

$P \xrightarrow{a,R} P' \implies \exists Q'$ *such that* $Q \xrightarrow{a,R'} Q', R\kappa R'$ *and* $P'\kappa Q'$

$Q \xrightarrow{a,R} Q' \implies \exists P'$ *such that* $P \xrightarrow{a,R'} P', R\kappa R'$ *and* $Q'\kappa P'$

The strong bisimilarity $\backsimeq$ *is the largest of such relations.*

As usual, the the weak bisimulation relation for PEP systems differs from the strong relation as it allows systems to differ in the silent transitions they perform.

We denote the reflexive transitive closure of $\stackrel{\tau}{\rightarrow}$ as $\Longrightarrow$, namely $P \Longrightarrow P'$ if either $P \equiv P'$ or there exist $P_1, \ldots, P_n$ such that $P \stackrel{\tau}{\rightarrow} P_1 \stackrel{\tau}{\rightarrow} \ldots \stackrel{\tau}{\rightarrow} P_n \stackrel{\tau}{\rightarrow} P'$. We denote with $\stackrel{\ell}{\Longrightarrow}$ with $\ell \neq \_$ a composition of transitions $\Longrightarrow\stackrel{\ell}{\rightarrow}\Longrightarrow$, namely $P \stackrel{\ell}{\Longrightarrow} P'$ if there exist P_1, P_2 such that $P \Longrightarrow P_1 \stackrel{\ell}{\rightarrow} P_2 \Longrightarrow P'$.

Definition 7.3 (PEP Weak Bisimulation)**.** *A binary relation κ on PEP terms is a* weak bisimulation *if, given P and Q such that $P \kappa Q$, the following conditions hold:*

$P \stackrel{\tau}{\rightarrow} P' \implies \exists Q'$ *such that* $Q \Longrightarrow Q'$ *and* $P' \kappa Q'$

$Q \stackrel{\tau}{\rightarrow} Q' \implies \exists P'$ *such that* $P \Longrightarrow P'$ *and* $Q' \kappa P'$

$P \xrightarrow{a,R} P' \implies \exists Q'$ *such that* $Q \stackrel{a,R'}{\Longrightarrow} Q', R \kappa R'$ *and* $P' \kappa Q'$

$Q \xrightarrow{a,R} Q' \implies \exists P'$ *such that* $P \stackrel{a,R'}{\Longrightarrow} P', R \kappa R'$ *and* $Q' \kappa P'$

The weak bisimilarity $\approx$ *is the largest of such relations.*

As usual, strong bisimilarity between systems implies weak bisimilarity, namely the from relation is contained into the latter.

Lemma 7.4. *Given $P, Q \in PEP$, it holds $P \simeq Q \implies P \approx Q$*

Proof. Follows directly from the definitions of the two relations. □

It holds that both bisimilarities are congruences. This, in process calculi theory, means that given two bisimilar systems P and Q, for any context C in which P and Q could be placed, it holds that $C[P]$ and $C[Q]$ are still bisimilar systems. Contexts for the PEP calculus can be defined as follows.

Definition 7.5 (Contexts)**.** Contexts *of PEP systems are given by the following grammar:*

$$C ::= \Box \quad | \quad C \circ R \quad | \quad R \circ C \quad | \quad !C \quad | \quad \sigma(\!|C|\!)$$

where R is a PEP system and $\Box$ denotes the empty context.

As usual, $C[P]$ denotes the application of the context C to the PEP system P, that is a new PEP system obtained by replacing $\Box$ with P in the context C, and $C[C']$ denotes context composition, that is a new context obtained by replacing $\Box$ with C' in the context C.

The congruence results are stated in the following two theorems.

Theorem 7.6 (Strong Congruence)**.** *The strong bisimilarity on PEP systems $\simeq$ is a congruence.*

Proof. We have to prove that given $P, Q \in PEP$ such that $P \simeq Q$, it holds $C[P] \simeq C[Q]$, namely that (i) for any transition $C[P] \stackrel{\tau}{\rightarrow} P'$ there exists a corresponding transition $C[Q] \stackrel{\tau}{\rightarrow} Q'$ such that $P' \simeq Q'$, and (ii) for any transition $C[P] \xrightarrow{a,R} P'$ there exists a corresponding transition $C[Q] \xrightarrow{a,R'} Q'$ such that $P' \simeq Q'$ and $R \simeq R'$. We prove this proposition by induction on the structure of C, and in each case by induction on the derivation of the performed transitions.

The base case is $C = \Box$. This case is trivial as $C[P] = P$ and $C[Q] = Q$.

In the inductive cases of $C = R \circ C'$ and $C = C' \circ R$ we have that $C[P] = R \circ C'[P]$ and $C[Q] = R \circ C'[Q]$. A transition $C[P] \xrightarrow{\ell} P'$ can be performed either by R in isolation, or by $C'[P]$ in isolation, or through an interaction between the two components. The first case is trivial as R can perform the same transitions also when it occurs in $C[Q]$. In the second case we can trivially apply the induction hypothesis. In the third case we have that ℓ is equal to $\_$, and R and $C'[P]$ are able to perform two transitions representing phagocytosis. By induction hypothesis we have that $C'[Q]$ can perform an equivalent transition and interact with R in $C[Q]$.

In the inductive case of $C = !C'$, by the definition of the structural congruence relation we have $!C'[P] \equiv !C'[P] \circ C'[P] \circ C'[P]$, and $!C'[Q] \equiv !C'[Q] \circ C'[Q] \circ C'[Q]$. If $!C'[P] \stackrel{\cdot}{\rightarrow} P'$, then there exist P'' such that $C'[P] \circ C'[P] \stackrel{\cdot}{\rightarrow} P''$ and $P' \equiv P'' \circ !C'[P]$. By induction hypothesis, there exists Q'' such that $C'[Q] \circ C'[Q] \stackrel{\cdot}{\rightarrow} Q''$ with $P'' \doteq Q''$. Similarly, $!C'[P] \equiv !C'[P] \circ C'[P]$ and $!C'[Q] \equiv !C'[Q] \circ C'[Q]$. If $!C'[P] \xrightarrow{a,R} P'$, then there exist P'' such that $C'[P] \xrightarrow{a,R'} P''$ and $P' \equiv P'' \circ !C'[P]$. By induction hypothesis, there exists Q'' such that $C'[Q] \xrightarrow{a,R'} Q''$ with $P'' \doteq Q''$ and $R \doteq R'$. Hence, for any transition performed by $!C[P]$ there exists an equivalent transition performed by $!C[P]$ leading to bisimilar systems.

Finally, in the inductive case of $C = \sigma(\!|C'|\!)$ the induction hypothesis can be applied trivially. □

Theorem 7.7 (Weak Congruence)**.** *The strong bisimilarity on PEP systems $\doteqdot$ is a congruence.*

Proof. The proof is similar to the proof of Theorem 7.6 □

7.3 Comparing PEP and CLS Bisimilarities

We have defined bisimulation relations for the PEP calculus and for CLS, and we have that the former can be translated into the latter by using the encoding we have given in Section 5.1. Now, it is interesting to verify whether there is some relationship between the bisimulation relations of the two formalisms. More precisely, we can verify whether the bisimilarity of two PEP systems is preserved by the encoding into CLS.

We start by comparing strong bisimilarities. It is easy to see that the strong bisimilarity is not preserved by the encoding (i.e. $P \doteq Q \implies \{\!|P|\!\} \sim \{\!|Q|\!\}$ does not hold) because $\Box$–labeled transitions are performed by the encoded systems to create looping sequences and to simulate some axioms of the structural congruence of the PEP calculus. As an example, consider the PEP calculus systems $P = \diamond$ and $Q = \mathbf{0}(\!|\diamond|\!)$. These two systems are structurally congruent, and both perform no transition. However, the encoding of the former (that is $act \cdot \mathbf{0}$) perform no transition too, while the encoding of the latter (that is $act \cdot brane \cdot a \cdot \mathbf{0} \cdot a \cdot \mathbf{0}$) performs two $\Box$–labeled transitions, the first caused by the application of the (*brane*) rule, and the second by the application of the (*sc*3) rule (and the term reached at the end is $act \cdot \mathbf{0}$, see Figure 5.3 in Section 5.1 for the definition of the rewrite rules associated with the encoding).

However, we can show that the encoding adds only silent behavior to a PEP system by proving the following theorem, which relates the strong bisimilarity on PEP systems with the weak bisimilarity on CLS terms.

Theorem 7.8. *Given two systems P, Q of the PEP calculus, the following holds:*

$$P \doteq Q \implies \{\!|P|\!\} \approx \{\!|Q|\!\}$$

To prove the theorem we introduce the following three lemmata.

Lemma 7.9. $\langle T_1 \rangle \equiv \langle T_2 \rangle \implies T_1 \approx T_2$.

Proof. Trivial: the transition system of $\langle T_i \rangle$ is the same of T_i apart from the transitions $\xrightarrow{\Box}$ due to the application of rules in $\mathcal{R}_{\langle\rangle}$. □

Lemma 7.10. $\{\!|P|\!\} \approx \langle \{\!|P|\!\} \rangle$.

Proof. Because $\langle \{\!|P|\!\} \rangle \equiv \langle \langle \{\!|P|\!\} \rangle \rangle$, and by Lemma 7.9. □

Lemma 7.11. $P \doteq Q \implies \langle \{\!|P|\!\} \rangle \approx \langle \{\!|Q|\!\} \rangle$.

Proof. By definition of $\doteq$ we know that if $P \xrightarrow{\ell} P'$ for some label ℓ, then $Q \xrightarrow{\ell'} Q'$ with $P' \doteq Q'$ and ℓ equivalent to ℓ'. We show that there exist P'' and Q'' such that: (i) the former can be constructed from ℓ and P' and the latter from ℓ' and Q'; (ii) $P'' \doteq Q''$; and (iii) $\langle \{\!|P|\!\} \rangle \overset{C}{\Longrightarrow} \langle \{\!|P''|\!\} \rangle$ and $\langle \{\!|Q|\!\} \rangle \overset{C}{\Longrightarrow} \langle \{\!|Q''|\!\} \rangle$. Since all the transitions performed by $\langle \{\!|P|\!\} \rangle$ and $\langle \{\!|Q|\!\} \rangle$ can be constructed in this manner, we have that $\langle \{\!|P|\!\} \rangle \approx \langle \{\!|Q|\!\} \rangle$.

If the transition performed by both P and Q is a silent transition, namely $\ell = \ell' = \_$, we have that P'' and Q'' correspond to P' and Q', respectively. The same holds when both P and Q perform a $\phi_n^\perp$ action, and in all these cases we have $P'' \doteq Q''$ by definition of the strong bisimulation on PEP systems.

More complex is the case in which both P and Q perform either a ϕ_n or a ε_n action, because they originate, in the corresponding CLS term, a transition for every possible context in which the action can be performed, and each of these transitions leads to a state in which the context has been incorporated in the term. We consider only the case of the ϕ_n action as the case of ε_n is analogous. We have that $\ell = (\phi_n, \sigma(\!|R|\!))$ and $\ell' = (\phi_n, \sigma'(\!|R'|\!))$, with $\sigma(\!|R|\!) \doteq \sigma'(\!|R'|\!)$, and we can construct infinitely many pairs of processes P'' and Q'' for each possible process having the form $\phi_n^\perp(\rho).\tau | \tau_0(\!|R_0|\!)$. More precisely, we have $P'' = \tau | \tau_0(\!|\rho(\!|\sigma(\!|R|\!)|\!) \circ R_0|\!)$ and $Q'' = \tau | \tau_0(\!|\rho(\!|\sigma'(\!|R'|\!)|\!) \circ R_0|\!)$. Since $\sigma(\!|R|\!)$ and $\sigma'(\!|R'|\!)$ are bisimilar and are placed in the same contexts, and since bisimulation is a congruence we have that $P'' \doteq Q''$.

So far we have proved points (i) and (ii). The proof of point (iii) follows from the definition of the rewrite rules associated with the encoding. □

The proof of Theorem 7.8 follows directly from Lemma 7.10 and Lemma 7.11. Moreover, to show that the inverse of Theorem 7.8 does not hold, we give a counterexample. Consider the PEP systems $P = \diamond$ and $Q = \varepsilon_n^\perp(\!|\varepsilon_n(\!|\diamond|\!)|\!)$. Their encodings are weakly bisimilar as the encoding of the former performs no transition, while the encoding of the latter performs only $\Box$–labeled transition, however the two PEP systems are not strongly bisimilar, as in the PEP labeled semantics, the former performs no transition and the latter one $\_$–labeled transition.

A stronger correspondence exists between the two weak bisimilarity relations of the PEP calculus and CLS. We show that the encodings of two weakly bisimilar PEP systems are two weakly bisimilar term, and vice–versa.

Theorem 7.12 (Full Abstraction)**.** *Given two systems P, Q of the PEP calculus, the following holds:*

$$P \div Q \iff \{\!|P|\!\} \approx \{\!|Q|\!\}$$

Proof. Lemma 7.10 allow us to reduce the proof of the theorem to the proof of $P \div Q \iff \langle\{\!|P|\!\}\rangle \approx \langle\{\!|Q|\!\}\rangle$. To prove direction $\Longrightarrow$ of this proposition we first notice that $P \stackrel{-}{\rightarrow} P'$ implies $\langle\{\!|P|\!\}\rangle \stackrel{\Box}{\Longrightarrow} \langle\{\!|P'|\!\}\rangle$. This can be proved by induction on the derivation of $P \stackrel{-}{\rightarrow} P'$. The base cases are when the last applied rules of the PEP labeled semantics are either (Pi1), or (Ph3), or (E2). In all these three cases we have that $\langle\{\!|P|\!\}\rangle$ can perform a $\Box$–labeled transition caused by the application of one of the CLS rewrite rules associated with the encoding, namely (pino), (phago) and (exo), respectively. After the application of these rules, a state equivalent to $\langle\{\!|P'|\!\}\rangle$ is reached by applying rewrite rules in $\mathcal{R}_{\langle\rangle}$, the application of which causes other $\Box$–labeled transitions. The rest of the proof is similar to the proof of Lemma 7.11.

As regards direction $\Longleftarrow$, we first notice that by the assumption that we start from terms which are in normal forms, namely $\langle\{\!|P|\!\}\rangle$ and $\langle\{\!|Q|\!\}\rangle$, we have that the only transitions that can be performed are those related to the rewrite rules (phago),(exo),(pino), (bangS) and (bangB). We omit considering rules (bangS) and (bangB) as they cause $\Box$–labeled transitions leading to terms that are the normal forms of the translations of other PEP processes that are structurally congruent to P and Q. Now, the proof consists in showing that if $\langle\{\!|P|\!\}\rangle \stackrel{C}{\longrightarrow} T_1$ and $\langle\{\!|Q|\!\}\rangle \stackrel{C}{\longrightarrow} T_2$, then there exist ℓ, P' and Q' such that $P \stackrel{\ell}{\rightarrow} P'$ and $Q \stackrel{\ell}{\rightarrow} Q'$, and $\langle T_1\rangle$ and $\langle T_2\rangle$ are structurally congruent to the translations of PEP processes resulting from compositions of ℓ and P' and ℓ and Q'. This can be done by cases on the rewrite rules applied to derive the transitions of $\langle\{\!|P|\!\}\rangle$ and$\langle\{\!|Q|\!\}\rangle$, by analyzing the structure of T_1 and T_2, and by reconstructing ℓ, P' and Q'. This is the inverse procedure of the one used in the proof of Lemma 7.11 to construct P'' and Q''. $\square$

Part III

Quantitative Modeling of Biological Systems

Chapter 8

Stochastic CLS

In the previous chapters we have introduced formalisms for the description of biological systems and we have provided formal tools, more precisely bisimulation relations, for the verification of properties of systems described these formalisms. We have considered only qualitative aspects of biological systems, such as their structure and the presence (or the absence) of certain molecules. As a consequence, by using the verification tools we have defined it is only possible to verify properties such as the reachability of particular states or causality relationships between events. What we are not able to verify with this tools are properties such as the time spent to reach a particular state, of the probability of reaching it. To face this problem, in this chapter we develop a stochastic extension of CLS, called *Stochastic CLS*, in which quantitative aspects such as time and probabilities are taken into account.

The standard way of extending a formalism to model quantitative aspects of biological systems is by incorporating the stochastic framework developed by Gillespie with its simulation algorithm for chemical reactions [31] in the semantics of the formalism. This has been done for instance for the π–calculus [62, 64].

We recalled Gillespie's algorithm in Section 2.3. The idea of the algorithm is that a rate constant is associated with each chemical reaction that may occur in the system. Such a constant is obtained by multiplying the kinetic constant of the reaction[1] by the number of possible combinations of reactants that may occur in the system. The resulting rate constant is then used as the parameter of an exponential distribution modeling the time spent between two occurrences of the considered chemical reaction.

The use of exponential distributions to represent the (stochastic) time spent between two occurrences of chemical reactions allows describing the system as a Continuous Time Markov Chain (CTMC – we recalled their definition in Section 2.2) and consequently allows verifying properties of the described system by means of analytic means and by means of stochastic model checkers.

In Stochastic CLS, incorporating Gillespie's stochastic framework is not a simple exercise. As we shall see, the main difficulty is counting the number of possible reactant combinations of the chemical reaction described by a rewrite rule. However, at the end we will be able to derive a CTMC from the semantics of the system modeled in Stochastic CLS, and hence we will be able to perform simulations and analysis. In order to use stochastic model checkers, such as PRISM [46], to verify properties of the described sys-

[1] Actually, not exactly the kinetic constant, but a constant derived from it. See Section 2.3 for details.

tem, it is often necessary that the state space of the system is finitary. However, this happens rarely, but we will discuss how an infinite–state model can be approximated by a finite–state one by considering a model of a real biological system and by imposing upper bounds to the quantities of reactants. This approximation of the model should be constructed after a study of the behavior of the system by means of simulations. In this manner, the system modeler may acquire a knowledge on the system which permits him/her to impose upper bounds to the quantities of reactants that are reasonable.

8.1 Definition of Stochastic CLS

We introduce the stochastic extension of CLS called Stochastic CLS. Terms and patterns of the new calculus are the same as in CLS (see Section 3). As regards rewrite rules, the choice of which rule to apply among many applicable depends on the frequencies of the events one wants to model. Thus, rewrite rules are enriched with rates representing those frequencies. The stochastic semantics of CLS is based on such rewrite rules.

To define the semantics of Stochastic CLS we will use the notion of contexts as defined in Section 6.1. We recall their definition here.

Definition 8.1 (Contexts)**.** *Contexts $\mathcal{C}$ are given by the following grammar:*

$$\mathcal{C} ::= \Box \quad | \quad \mathcal{C} \,|\, T \quad | \quad T \,|\, \mathcal{C} \quad | \quad (S)^L \rfloor \mathcal{C}$$

where $T \in \mathcal{T}$ and $S \in \mathcal{S}$. Context $\Box$ is called the empty context*.*

8.1.1 Rewrite rules in Stochastic CLS

To describe the evolution of a term, the stochastic semantics must take into account, besides the rate of a rule, also the number of occurrences of subterms to which the rule can be applied and the terms produced. Intuitively, subterms to which the rule can be applied correspond to reactants in a biological system. More precisely, in what follows a *subterm* of a term T will be a term $T' \not\equiv \epsilon$ for which a context C exists such that $T \equiv C[T']$, while a *reactant* will be an occurrence in T of a subterm. For example, if $T = a \,|\, a \,|\, b \,|\, b$, then the set of subterms of T is

$$\{a \ , \ b \ , \ a \,|\, a \ , \ a \,|\, b \ , \ b \,|\, b \ , \ a \,|\, a \,|\, b \ , \ a \,|\, b \,|\, b \ , \ T\}$$

and the multiset of reactants in T is

$$\{a \ , \ a \ , \ b \ , \ b \ , \ a \,|\, a \ , \ a \,|\, b \ , \ a \,|\, b \ , \ a \,|\, b \ , \ a \,|\, b \ , \ b \,|\, b \ , \ a \,|\, a \,|\, b \ , \ a \,|\, a \,|\, b \ , \ a \,|\, b \,|\, b \ , \ a \,|\, b \,|\, b \ , \ T\}$$

Now, defining the stochastic semantics would be easy if rules would contain no variable. For instance, if we have the rewrite rule $a \,|\, b \overset{k}{\mapsto} c$, where k is the kinetic constant of the modeled chemical reaction, then its application rate is k multiplied by the number of possible combinations of occurrences of a and b in the term, namely the number of occurrences of $a \,|\, b$ in the multiset of reactants of the term. For example, if the term is T defined as above, we have that it contains two occurrences of a and two of b, hence the number of possible combinations of reactants is $2 \times 2 = 4$, and this holds also in the multiset of reactants of T, which contains four instances of $a \,|\, b$.

If we have variables, we have to take into account how they can be instantiated in order to compute the application rate of the rewrite rule. For instance, consider the rule $a \cdot \widetilde{x} \,|\, a \cdot \widetilde{y} \overset{k}{\mapsto} d$ and the term $T = a \cdot b \,|\, a \cdot c$. The multiset of reactants in T in this case is $\{a \cdot b\ ,\ a \cdot c\ ,\ T\}$. For which value should we multiply the kinetic constant of the rule? Maybe one, as there is one only reactant which could be matched by the left hand side of the rule. Or maybe two, as there are two possible instantiations of the variables which can match a reactant, namely $\widetilde{x} = b, \widetilde{y} = c$ and $\widetilde{x} = c, \widetilde{y} = b$. In both cases, however, it would be quite difficult to define the procedure to compute the application rate of the rule in an operational manner.

We remark that this situation is rather unusual, and this problem has never been faced during the development of the stochastic extension of other formalisms such as the π–calculus, as those formalisms are not able to model chemical reactions with symbolic molecules (as CLS patterns are). Also Gillespie's work does not deal with variables in the simulated chemical reactions. As a consequence, we are free to give the interpretation to rewrite rules with variables that makes the definition of the stochastic semantics easier, provided it is a reasonable interpretation.

We choose to consider a rewrite rule with variables as a *rewrite rule schema*, namely as a succinct notation for the possibly infinite set of ground rewrite rules which can be obtained by instantiating its variables. For instance, the rule $a \cdot x \overset{k}{\mapsto} b$ can be seen as equivalent to the infinite set of ground rules

$$\{a \cdot a \overset{k}{\mapsto} b\ ,\ a \cdot b \overset{k}{\mapsto} b\ ,\ a \cdot c \overset{k}{\mapsto} b\ ,\ a \cdot d \overset{k}{\mapsto} b\ ,\ a \cdot e \overset{k}{\mapsto} b\ ,\ \ldots\}$$

Now, given an infinite set of ground rewrite rules obtained from a rewrite rule schema, we can select a finite subset of ground rules which are the only applicable to the term we are considering. For instance, if the rewrite rule is $a \cdot \widetilde{x} \,|\, a \cdot \widetilde{y} \overset{k}{\mapsto} d$ as before and the term is $T = a \cdot b \,|\, a \cdot c$, we can derive one only ground rewrite rule which is applicable to T, namely $a \cdot b \,|\, a \cdot c \overset{k}{\mapsto} d$. Instead, if the rule is $a \cdot \widetilde{x} \,|\, a \cdot \widetilde{y} \overset{k}{\mapsto} \widetilde{y}$, we can derive two ground rules, namely $a \cdot b \,|\, a \cdot c \overset{k}{\mapsto} b$ and $a \cdot b \,|\, a \cdot c \overset{k}{\mapsto} c$. In this manner we reduce the problem of defining the semantics of the application of rewrite rule schemata to the simpler problem of defining the semantics with ground rules only.

There is another point we have to discuss about the presence of variables in rewrite rules, and it is related with the value of the kinetic constant of the reaction. Our interpretation of variables allow a rewrite rule to represent a schema of ground rules, and this means that it represent a schema of chemical reactions. Now, these reactions may have different kinetic constants. Moreover, it often happens that the application rate of a rewrite rule depends on how many molecules of some kind are contained in the part of the system represented by a variable. For instance, if we have a rule such as $a \,|\, \left(b \cdot \widetilde{x}\right)^L \rfloor X \overset{k}{\mapsto} \left(c \cdot \widetilde{x}\right)^L \rfloor X$, representing the binding of molecule a with an instance of b on the membrane represented by the looping sequence, we should have that the application rate of the derived ground reactions is proportional to the number of b which are present on the membrane, that is the number of b in the instantiation of the $\widetilde{x}$ variable plus one. To solve this problem, in rewrite rule schemata we use a function from variable instantiations to real numbers instead of kinetic constants. In each ground rule derived by a schema, such a function will be applied to the instantiation function used to derive the ground rule and the result will be used as the kinetic constant of the ground rule.

Definition 8.2 (Stochastic Rewrite Rule Schema)**.** *A* rewrite rule schema *is a triple* (P_1, P_2, f)*, denoted with* $P_1 \overset{f}{\mapsto} P_2$*, where* $P_1, P_2 \in \mathcal{P}$*,* $P_1 \not\equiv \epsilon$ *and such that* $Var(P_1) \subseteq Var(P_2)$*, and* $f : \Sigma \to \mathbb{R}^{\geq 0}$ *is the* rewriting rate function*. We denote with* $\Re$ *the infinite set of all the possible rewrite rules.*

By instantiating the variables of a stochastic rewrite rule schema, a *stochastic ground rewrite rule* is obtained.

Definition 8.3 (Stochastic Ground Rewrite Rule)**.** *Given a stochastic rewrite rule schema* $R = (P_1, P_2, f)$ *and an instantiation function* $\sigma \in \Sigma$*, the* ground rewrite rule *derived from* R *and* σ *is a triple* (T_1, T_2, c)*, denoted* $T_1 \overset{c}{\mapsto} T_2$*, where* $T_1 = P_1\sigma$*,* $T_2 = P_2\sigma$*, and* $c = f(\sigma)$ *is the* rewriting rate constant*. We denote with* $\Re g$ *the infinite set of all the possible ground rewrite rules.*

Example 8.4. Let us assume a function $occ : \mathcal{E} \times \mathcal{T} \to \mathbb{N}$ such that $occ(a, T)$ returns the number of elements a syntactically occurring in the term T. Consider the rewrite rule schema $R = (a \,|\, \left(c \cdot \widetilde{x}\right)^L, b \,|\, \left(\widetilde{x}\right)^L, f(\sigma) = occ(c, \sigma(\widetilde{x})) + 1)$ and the instantiation $\sigma(\widetilde{x}) = b \cdot c$. We obtain a stochastic ground rewrite rule $(a \,|\, \left(c \cdot b \cdot c\right)^L, b \,|\, \left(b \cdot c\right)^L, p)$, where $p = f(\sigma) = occ(c, b \cdot c) + 1 = 2$.

We now define the set of all the ground rules, derived from a set of rewrite rule schemata, that can be applied to a given term.

Definition 8.5 (Applicable Ground Rewrite Rules)**.** *Given a rewrite rule schema* $R = P_1 \overset{f}{\mapsto} P_2$ *and a term* $T \in \mathcal{T}$*, the set of* ground rewrite rules *derived from* R *and applicable to* T *is defined as*

$$AR(R, T) = \{T_1 \overset{c}{\mapsto} T_2 \,|\, \exists \sigma \in \Sigma, C \in \mathcal{C}.\ T = P_1\sigma, T_2 = P_2\sigma, T \equiv C[T_1], c = f(\sigma)\}$$

Given a finite set of rewrite rule schemata $\mathcal{R}$ *and a term* $T \in \mathcal{T}$*, the set of* ground rewrite rules *derived from* $\mathcal{R}$ *and applicable to* T *is the set:*

$$AR(\mathcal{R}, T) = \bigcup_{R \in \mathcal{R}} AR(R, T)$$

By the finiteness of T and $\mathcal{R}$ we immediately obtain the following proposition.

Proposition 8.6. *For any* $R, \mathcal{R}$ *and* T*,* $AR(R, T)$ *and* $AR(\mathcal{R}, T)$ *are finite.*

Example 8.7. Consider again the function $occ : \mathcal{E} \times \mathcal{T} \to \mathbb{N}$ defined in Example 8.4. The rewrite rule schema $R = (a \,|\, \left(c \cdot \widetilde{x}\right)^L, b \,|\, \left(\widetilde{x}\right)^L, f(\sigma) = occ(c, \sigma(\widetilde{x})) + 1)$ and the initial term $T = a \,|\, \left(c \cdot c\right)^L \,|\, \left(c\right)^L$. We have that $AR(\mathcal{R}, T) = \{a \,|\, \left(c \cdot c\right)^L \overset{2}{\mapsto} b \,|\, \left(c\right)^L, a \,|\, \left(c\right)^L \overset{1}{\mapsto} b \,|\, \left(\epsilon\right)^L\}$

Now, in order to compute the application rate of the rewrite rule we define multiset of reactants of a term T. In the following we will need also to know from which context each reactant has been extracted, hence we defined the *multiset of extracted reactants* of T as the multiset of all the pairs (T', C) where $T' \not\equiv \epsilon$ is a reactant in T and C is the context such that $C[T'] \equiv T$. In the definition we will use the auxiliary function $\circ : \mathcal{C} \times (\mathbb{N} \times \mathcal{T} \times \mathcal{C}) \mapsto (\mathbb{N} \times \mathcal{T} \times \mathcal{C})$ defined as $C \circ (i, T, C') = (i, T, C[C'])$ extended to multisets of triples over $\mathbb{N} \times \mathcal{T} \times \mathcal{C}$ in the obvious way.

Definition 8.8 (Multiset of Extracted Reactants)**.** *Given a term $T \in \mathcal{T}$, the multiset of reactants extracted from T is defined as*

$$ext(T) = \big\{(T', C) | (n, T', C) \in ext_\ell(0, T)\big\}$$

where ext_ℓ is given by the following recursive definition:

$$\begin{aligned}
ext_\ell(i, S) &= \{(i, S, \Box)\} \\
ext_\ell(i, \big(S\big)^L) &= \{(i, \big(S\big)^L, \Box)\} \\
ext_\ell(i, \big(S\big)^L \rfloor T') &= \{(i, \big(S\big)^L \rfloor T', \Box)\} \ \cup \ \big(S\big)^L \rfloor \Box \circ ext_\ell(i+1, T') \\
ext_\ell(i, T_1 \,|\, T_2) &= T_2 \,|\, \Box \circ ext_\ell(i, T_1) \ \cup \ T_1 \,|\, \Box \circ ext_\ell(i, T_2) \\
&\quad \cup \ \{(i, T_1^e \,|\, T_2^e, C_1^e[C_2^e]) \,|\, (i, T_j^e, C_j^e) \in ext_\ell(i, T_j), j \in \{1, 2\}\}
\end{aligned}$$

Given a term $T \in \mathcal{T}$, $ext(T)$ extracts from T the multiset of reactants to which a rewrite rule could be applied. Each element of the multiset contains also the context from which each reactant is extracted. Recall that a reactant is an occurrence of a subterm in T, we have, for example, $ext(a \,|\, a) = \{(a, a \,|\, \Box), (a, a \,|\, \Box), (a \,|\, a, \Box)\}$, where the two $(a, a \,|\, \Box)$-elements correspond to the two reactants a in $a \,|\, a$. The function ext makes use of the function ext_ℓ in order to separate reactants obtained at different levels of containment (which cannot be mixed). For example, let T be $a \,|\, \big(b\big)^L \rfloor c$, then $ext_\ell(0, T) = \{(0, a, \Box \,|\, \big(b\big)^L \rfloor c), (0, \big(b\big)^L \rfloor c, a \,|\, \Box), (1, c, a \,|\, \big(b\big)^L \rfloor \Box), \ (0, T, \Box)\}$, and $ext(T) = \{(a, \Box \,|\, \big(b\big)^L \rfloor c), (\big(b\big)^L \rfloor c, a \,|\, \Box), (c, a \,|\, \big(b\big)^L \rfloor \Box), (T, \Box)\}$. Note that ext_ℓ avoids $(a \,|\, c, C)$ to be extracted from T, for any context C.

We have given a constructive definition of $ext(T)$. In the next subsection we develop some theoretical work to prove that the definition is correct. The proof is rather complicated and not essential to understand the semantics of the formalism. A reader interested only in the definition of the formalism can skip the next subsection and jump to Subsection 8.1.3.

8.1.2 On the correctness of the definition of ext

We give two lemmas stating some properties of the ext_ℓ and ext functions.

Lemma 8.9. *The following two properties hold:*
1. $(i, T', C') \in ext_\ell(i, T) \implies \exists T''.C' \equiv T'' \,|\, \Box \vee C' \equiv \Box$
2. $(i, T', C') \in ext_\ell(i, T) \implies T \equiv T' \,|\, C'[\epsilon]$.

Proof. We prove point number 1 by induction on the structure of T. The only non trivial case is $T = T_1 | T_2$. By the definition of ext_ℓ we have
either $(i, T', C') \in T_2 \,|\, \Box \circ ext_\ell(i, T_1)$,
or $(i, T', C') \in T_1 \,|\, \Box \circ ext_\ell(i, T_2)$,
or $(i, T', C') \in \{(i, T_1^e \,|\, T_2^e, C_1^e[C_2^e]) \,|\, (i, T_j^e, C_j^e) \in ext_\ell(i, T_j), j \in \{1, 2\}\}$.
In the first case by induction hypothesis $(i, T_1', C_1') \in ext_\ell(i, T_1) \implies \exists T_1''.C_1' \equiv T_1'' \,|\, \Box \vee C_1' = \Box$. Hence, $C' = T_2 \,|\, C_1'$ that is either $C' = T_2 \,|\, T_1'' \,|\, \Box$, or $C' = T_2 \,|\, \Box$. The second case is analogous to the first. In the third case, by induction hypothesis $(i, T_j^e, C_j^e) \in$

$ext_\ell(i,T_j) \implies \exists T_j''.C_j^e \equiv T_j'' \,|\, \Box \vee C_j^e = \Box$. Hence, $C' = C_1^e[C_2^e]$ that is either $C' = \Box$, or $C' = T_1'' \,|\, \Box$, or $C' = T_2'' \,|\, \Box$, or $C' = T_1'' \,|\, T_2'' \,|\, \Box$.

Now we prove point number 2 by induction on the structure of T. Again, the only non trivial case is $T = T_1 \,|\, T_2$ and we have the same three cases of above. If $(i,T',C') \in T_2 \,|\, \Box \circ ext_\ell(i,T_1)$, by the induction hypothesis we have $(i,T'',C'') \in ext_\ell(i,T_1) \implies T_1 \equiv T'' \,|\, C''[\epsilon]$. Now, $T' = T''$ and $C' = T_2 \,|\, C''$, hence $T' \,|\, C'[\epsilon] \equiv T'' \,|\, T_2 \,|\, C''[\epsilon] \equiv T_1 \,|\, T_2$. The second case is analogous to the first. In the third case, namely if $(i,T',C') \in \{(i,T_1^e \,|\, T_2^e, C_1^e[C_2^e]) \,|\, (i,T_j^e,C_j^e) \in ext_\ell(i,T_j), j \in \{1,2\}\}$, by the induction hypothesis we have $(i,T_j^e,C_j^e) \in ext_\ell(i,T_j) \implies T_j \equiv T_j^e \,|\, C_j^e[\epsilon]$. Hence, $T \equiv T_1^e \,|\, C_1^e[\epsilon] \,|\, T_2^e \,|\, C_2^e[\epsilon]$.

Note that, by point number 1 it holds either $C_j^e \equiv \Box$ or $C_j^e \equiv T_j' \,|\, \Box$ for some T_j'. As a consequence, $C_1^e[C_2^e] \equiv C_1^e[\epsilon] \,|\, C_2^e[\epsilon]$. Hence, $T \equiv T_1^e \,|\, T_2^e \,|\, C_1^e[C_2^e[\epsilon]] \equiv T' \,|\, C'[\epsilon]$. □

Lemma 8.10. *The following two properties hold:*
1. $(T',C) \in ext(T) \implies C[T'] \equiv T$
2. $C[T'] \equiv T \implies \exists C', T''.C' \equiv C \wedge T'' \equiv T' \wedge (T'',C') \in ext(T)$

Proof. In order to prove the first point, we prove the more general property

$$\forall i,j \in \mathbb{N}.\, \big((j,T',C) \in ext_\ell(i,T)\big) \implies C[T'] \equiv T$$

We prove it by induction on the structure of T. The two base cases are $T = S$ and $T = \big(S\big)^L$. We have $ext_\ell(i,T)$ equal to $\{(i,S,\Box)\}$ and $\{(i,\big(S\big)^L,\Box)\}$, respectively, then we have $\Box[S] \equiv S$ and $\Box[\big(S\big)^L] \equiv \big(S\big)^L$, respectively.

If $T \equiv \big(S\big)^L \rfloor T''$ then

$$ext_\ell(i,\big(S\big)^L \rfloor T'') = \{(i,\big(S\big)^L \rfloor T'',\Box)\} \cup \big(S\big)^L \rfloor \Box \circ ext_\ell(i+1,T').$$

Now we have two cases: in the first $(j,T',C) = (i,\big(S\big)^L \rfloor T'',\Box)$, hence $C[T'] \equiv \Box[\big(S\big)^L \rfloor T''] \equiv \big(S\big)^L \rfloor T'' \equiv T$. In the second case $(j,T',C) \in \big(S\big)^L \rfloor \Box \circ ext_\ell(i+1,T')$. By the induction hypothesis $\forall i',j' \in \mathbb{N}.(j',T''',C') \in ext_\ell(i',T'') \implies C'[T'''] \equiv T''$. This holds in particular for $i' = i+1$, then $T' = T'''$, $C = (\big(S\big)^L \rfloor \Box)[C']$, and $C[T'] \equiv \big(S\big)^L \rfloor C'[T'''] \equiv \big(S\big)^L \rfloor T'' \equiv T$.

If $T \equiv T_1 \,|\, T_2$ then

$$\begin{aligned} ext_\ell(i,T_1 \,|\, T_2) &= T_2 \,|\, \Box \circ ext_\ell(i,T_1) \\ &\quad \cup T_1 \,|\, \Box \circ ext_\ell(i,T_2) \cup \{(i,T_1^e \,|\, T_2^e, C_1^e[C_2^e]) \,|\, (i,T_j^e,C_j^e) \in ext_\ell(i,T_j), j \in \{1,2\}\}. \end{aligned}$$

Now, if $(j,T',C) \in T_2 \,|\, \Box \circ ext_\ell(i,T_1) \cup T_1 \,|\, \Box \circ ext_\ell(i,T_2)$ the proof is similar to the proof of the second subcase of the case $T \equiv \big(S\big)^L \rfloor T''$ described above. If $(j,T',C) \in \{(i,T_1^e \,|\, T_2^e, C_1^e[C_2^e]) \,|\, (i,T_h^e,C_h^e) \in ext_\ell(i,T_h), h \in \{1,2\}\}$, by the induction hypothesis we have $\forall i_h', j_h' \in \mathbb{N}.\, ((j_h',T_h^e,C_h^e) \in ext_\ell(i_h,T_h)) \implies C_h^e[T_h^e] \equiv T_h$. In the particular case of $i_1' = i_2' = i$ we have $T' = T_1^e \,|\, T_2^e$, and $C = C_1^e[C_2^e]$. By the second point of Lemma 8.9) we have $C[T'] \equiv C[\epsilon] \,|\, T'$, then $C[\epsilon] \,|\, T' \equiv C_1^e[C_2^e[\epsilon]] \,|\, T_1^e \,|\, T_2^e \equiv$ (by the first point of Lemma 8.9) $C_1^e[\epsilon] \,|\, C_2^e[\epsilon] \,|\, T_1^e \,|\, T_2^e \equiv$ (by the second point of Lemma 8.9) $C_1^e[T_1^e] \,|\, C_2^e[T_2^e] \equiv T_1 \,|\, T_2$.

We prove point 2 by induction on the structure of C. It is easy to see that $\forall T \in \mathcal{T}.(T,\Box) \in ext(T)$, and this proves the base case $C = \Box$. If $C = C_1 \,|\, T_1$, then $T \equiv C_1[T'] \,|\, T_1$. By induction hypothesis we have $\exists C_1', T''.C_1' \equiv C_1 \wedge T' \equiv T'' \wedge (T'',C_1') \in$

$ext(C_1[T'])$, that is $(0, T'', C_1') \in ext_\ell(0, C_1[T'])$. Now, by definition of ext_ℓ we have $\Box \,|\, T_1 \circ (0, T'', C_1') \in ext_\ell(0, T)$ that implies $(T'', C_1' \,|\, T_1) \in ext(T)$, with $T'' \equiv T'$ and $C_1' \,|\, T_1 \equiv C$. If $C = \big(S\big)^L \rfloor C_1$, then $T \equiv \big(S\big)^L \rfloor C_1[T']$. By induction hypothesis we have $\exists C_1', T''.C_1' \equiv C_1 \wedge T' \equiv T'' \wedge (T'', C_1') \in ext(C_1[T'])$, that is $(0, T'', C_1') \in ext_\ell(0, C_1[T'])$. Note that $\forall \widetilde{T}, \widetilde{T}', \widetilde{C}.(i, \widetilde{T}', \widetilde{C}) \in ext_\ell(k, \widetilde{T}) \iff (i+1, \widetilde{T}', \widetilde{C}) \in ext_\ell(k+1, \widetilde{T})$. Now, by definition of ext_ℓ we have $\big(S\big)^L \rfloor \Box \circ (1, T'', C_1') \in ext_\ell(0, T)$, that implies $(T'', \big(S\big)^L \rfloor C_1') \in ext(T)$, with $T'' \equiv T'$ and $\big(S\big)^L \rfloor C_1' \equiv C$. □

In order to show that the definition of $ext(T)$ is correct, we have to prove that $ext(T)$ contains exactly all the reactants in T. To prove this, we give a labeling function L which makes syntactically different all the reactants, and we show that computing $ext(T)$ is equivalent to computing the set of subterms of $L(T)$ modulo structural congruence, and then removing all labels.

Labels are strings over the alphabet $\{\cdot_0, \cdot_1, |_0, |_1, \rfloor_0, \rfloor_1\}$. We denote with λ the empty label, with Λ the set of all possible labels (including λ), and with ω a generic label in Λ. Let $\mathcal{E}'$ be a set of elementary constituents such that $\omega \in \mathcal{E}'$ and $\omega a \in \mathcal{E}'$ for any string ω and for all a in $\mathcal{E}$, we denote with $\mathcal{S}'$ and $\mathcal{T}'$ the sets of elementary sequences and terms built over $\mathcal{E}'$, respectively.

Definition 8.11 (Labeling/Unlabeling)**.** *The* labeling function $L : \mathcal{S} \cup \mathcal{T} \times \Lambda \mapsto \mathcal{S}' \cup \mathcal{T}'$ *and the* unlabeling function $L^{-1} : \mathcal{S}' \cup \mathcal{T}' \mapsto \mathcal{S} \cup \mathcal{T}$ *are recursively defined as follows:*

$$
\begin{array}{rcl@{\qquad}rcl}
L(\epsilon, \omega) &=& \omega & L^{-1}(\omega) &=& \epsilon \\
L(a, \omega) &=& \omega a \quad \forall a \in \mathcal{E} & L^{-1}(\omega a) &=& a \quad \forall a \in \mathcal{E} \\
L(S_1 \cdot S_2, \omega) &=& L(S_1, \omega\cdot_0) \cdot L(S_1, \omega\cdot_1) & L^{-1}(S_1 \cdot S_2) &=& L^{-1}(S_1) \cdot L^{-1}(S_2) \\
L(\big(S\big)^L, \omega) &=& \big(L(S, \omega)\big)^L & L^{-1}(\big(S\big)^L) &=& \big(L^{-1}(S)\big)^L \\
L(T_1 \,|\, T_2, \omega) &=& L(T_1, \omega|_0) \,|\, L(T_2, \omega|_1) & L^{-1}(T_1 \,|\, T_2) &=& L^{-1}(T_1) \,|\, L^{-1}(T_2) \\
L(\big(S\big)^L \rfloor T, \omega) &=& \big(L(S, \omega\rfloor_0)\big)^L \rfloor L(T, \omega\rfloor_1) & L^{-1}(\big(S\big)^L \rfloor T) &=& L^{-1}(\big(S\big)^L) \rfloor L^{-1}(T)
\end{array}
$$

We assume that L is always applied to minimal terms, namely terms in which ϵ appears only as the only element of non empty looping sequences (e.g. $\big(\epsilon\big)^L \rfloor b$). We extend L^{-1} to contexts by saying that $L^{-1}(\Box) = \Box$, to triples (i, T, C) by saying $L^{-1}((i, T, C)) = (i, L^{-1}(T), L^{-1}(C))$, and to multisets of such triples in the obvious way. An example of use of the labeling function is the following:

$$L(\big(a\big)^L \rfloor (b \,|\, b), \lambda) = \big(\rfloor_0 a\big)^L \rfloor (\rfloor_1|_0 b \mid \rfloor_1|_1 b)$$

It is easy to see that $\forall \omega \in \Lambda. L^{-1}(L(T, \omega)) = T$, however L^{-1} is not exactly the inverse of L because L^{-1} can be used to remove labels from labelled terms which are not in the image of L, as in this example:

$$L^{-1}(|_1 a \cdot \rfloor_0 b) = a \cdot b$$

Lemma 8.12. $L^{-1}(C' \circ (i, T, C)) = L^{-1}(C') \circ L^{-1}((i, T, C))$.

Proof. By the definitions of $\circ$ and of L^{-1} it is possible to derive:

$$
\begin{array}{l}
L^{-1}(C' \circ (i, T, C)) = L^{-1}((i, T, C'[C])) = \\
\qquad (i, L^{-1}(T), L^{-1}(C'[C])) = (i, L^{-1}(T), L^{-1}(C')[L^{-1}(C)])) = \\
\qquad\qquad L^{-1}(C') \circ (i, L^{-1}(T), [L^{-1}(C)])) = L^{-1}(C') \circ L^{-1}((i, T, C))
\end{array}
$$

□

In the following proposition we use the labeling and unlabeling functions to show that $ext(T)$ computes the expected multiset of pairs (T', C). The idea is to use the labeling function to distinguish among all the instances of the same pair (T', C) that can be extracted from T, and to show that $ext(T)$ extracts all of them. In the proposition we denote with $sub(T)$ the set of pairs (T', C) where each element is a representative of a congruence class in $\{(T'', C')|C'[T''] \equiv T, T'' \not\equiv \epsilon\}_{/\equiv}$ (where $\equiv$ is extended to pairs in the obvious way). In other words, $sub(T)$ contains one instance of each pair (T', C) that could be extracted from T, without distinguishing between structurally congruent pairs.

Proposition 8.13. $\forall \omega \in \Lambda. ext(T) = L^{-1}(sub(L(T, \omega)))$.

Proof. We first prove $\forall \omega \in \Lambda. ext(L(T, \omega)) = sub(L(T, \omega))$, namely that $ext(L(T, \omega))$ is a valid set of representatives for $\{(T'', C')|C'[T''] \equiv T, T'' \not\equiv \epsilon\}_{/\equiv}$. Lemma 8.10 ensures that for all $(T', C) \in sub(L(T, \omega))$ there exists at least one $(T'', C') \in ext(L(T, \omega))$ such that $T' \equiv T''$ and $C \equiv C'$, and vice–versa. In order to prove the equality of the two sets, we have to prove that such a pair (T'', C') is unique in $ext(L(T, \omega))$. This can be done by induction on the structure of T, and the only non trivial case is $T = T_1 \,|\, T_2$. In this case we have $ext_\ell(i, L(T_1 \,|\, T_2, \omega)) = ext_\ell(i, L(T_1, \omega|_0) \,|\, L(T_2, \omega|_1)) = M_1 \cup M_2 \cup M_3$, where $M_1 = L(T_2, \omega|_1) \,|\, \Box \circ ext_\ell(i, L(T_1, \omega|_1))$, $M_2 = L(T_1, \omega|_0) \,|\, \Box \circ ext_\ell(i, L(T_2, \omega|_1))$ and $M_3 = \{(i, T_1^e \,|\, T_2^e, C_1^e[C_2^e]) \,|\, (i, T_j^e, C_j^e) \in ext_\ell(i, L(T_j, \omega|_{j-1})), j \in \{1, 2\}\}$. As a consequence of the induction hypothesis, we have that in each multiset M_k each element is unique. Moreover, it is easy to see that the three multisets M_1, M_2 and M_3 are pairwise disjoint because the contexts appearing in their elements are obtained from different subterms of T, and thus they turn out to be syntactically different after the application of the labeling function.
Now, the proposition can be rewritten as $\forall \omega \in \Lambda. ext(T) = L^{-1}(ext(L(T, \omega)))$. We prove it by induction on the structure of T, and we show only the case $T = \big(S\big)^L \rfloor T'$ as the base cases are trivial and the case $T = T_1 | T_2$ is similar:
$L^{-1}(ext_\ell(i, L(\big(S\big)^L \rfloor T', \omega))) =$ (by def. of L)
$L^{-1}(ext_\ell(i, L(\big(S\big)^L, \omega\rfloor_0) \rfloor L(T', \omega\rfloor_1))) =$ (by def. of ext_ℓ)
$L^{-1}(\{(i, L(\big(S\big)^L \rfloor T', \omega), \Box)\}) \cup L^{-1}(L(\big(S\big)^L, \omega\rfloor_0) \rfloor \Box \circ ext_\ell(i + 1, L(T', \omega\rfloor_1)))$
$=$ (by Lemma 8.12)
$L^{-1}(\{(i, L(\big(S\big)^L \rfloor T', \omega), \Box)\}) \cup L^{-1}(L(\big(S\big)^L, \omega\rfloor_0) \rfloor \Box) \circ L^{-1}(ext_\ell(i + 1, L(T', \omega\rfloor_1)))$
$=$ (by definitions of L, L^{-1} and by induction hypothesis)$\{(i, \big(S\big)^L \rfloor T', \Box)\} \cup \big(S\big)^L \rfloor \Box \circ ext_\ell(i + 1, T')$. □

Example 8.14. Consider again the term $\big(a\big)^L \rfloor (b \,|\, b)$. The result obtained by computing $sub(L(\big(a\big)^L \rfloor (b \,|\, b), \lambda))$ is

$$\{(\big(\rfloor_0 a\big)^L \rfloor (\rfloor_1|_0 b \mid \rfloor_1|_1 b), \Box), (\rfloor_1|_0 b \mid \rfloor_1|_1 b, \big(\rfloor_0 a\big)^L \rfloor \Box), \\ (\rfloor_1|_0 b, \big(\rfloor_0 a\big)^L \rfloor (\Box \mid \rfloor_1|_1 b)), (\rfloor_1|_1 b, \big(\rfloor_0 a\big)^L \rfloor (\Box \mid \rfloor_1|_0 b))\}.$$

The unlabeling of this set is

$$\{(\big(a\big)^L \rfloor (b \,|\, b), \Box), (b \,|\, b, \big(a\big)^L \rfloor \Box), (b, \big(a\big)^L \rfloor (\Box \,|\, b)), (b, \big(a\big)^L \rfloor (\Box \,|\, b))\}$$

which is equal to $ext(\big(a\big)^L \rfloor (b \,|\, b))$.

8.1.3 The semantics of Stochastic CLS

The *ext* function computes the multiset of reactants of a term. We use such a function to compute the application rate of a ground rewrite rule. In particular, we compute the *application cardinality* of the rule, that is the number of reactants in the term in which the rule is applied that are equivalent to the left–hand side of the rule. This value will be multiplied by the kinetic constant of the reaction to obtain the application rate.

Definition 8.15 (Application Cardinality)**.** *Given a ground rewrite rule* $R = T_2 \overset{c}{\mapsto} T_2$ *and two terms* $T, T_r \in \mathcal{T}$*, the* application cardinality *of rule* R *leading from* T *to* T_r*,* $AC(R, T, T_r)$*, is defined as follows:*

$$AC(R, T, T_r) = |\left\{(T', C) \in ext(T) \text{ such that } T' \equiv T_1 \wedge C[T_2] \equiv T_r\right\}|.$$

As already mentioned, given a term T, a ground rewrite rule can be applied to different reactants of T. Hence, according to the reactant to which the rule is applied, the rewrite of T may result in different terms. For any reachable term, the application cardinality counts the number of reactants leading to it.

Example 8.16. Consider the ground rewrite rule $R = a \overset{c}{\mapsto} b$ and term $T = a \,|\, a \,|\, \left(m\right)^L \rfloor\, a$. The left part of the rule, consisting of the single element a, is contained three times in the set $ext(T)$, however the application of rule R in those three points gives rise to different terms. In particular, for the two elements (a, C) where $C = a \,|\, \left(m\right)^L \rfloor\, a \,|\, \square$ we have $T_r = C[T_2] = a \,|\, \left(m\right)^L \rfloor\, a \,|\, b$, and hence $AC(R, T, T_r) = 2$. On the other hand, for the element (a, C'), with $C' = a \,|\, a \,|\, \left(m\right)^L \rfloor\, \square$, we have $T'_r = C'[T_2] = a \,|\, a \,|\, \left(m\right)^L \rfloor\, b$, and hence $AC(R, T, T'_r) = 1$.

We now give the semantics of Stochastic CLS.

Definition 8.17 (Semantics)**.** *Given a finite set of rewrite rule schemata* $\mathcal{R}$*, the semantics of Stochastic CLS is the least labeled transition relation satisfying the following inference rule:*

$$\frac{R = T_1 \overset{c}{\mapsto} T_2 \in AR(\mathcal{R}, T) \qquad T \equiv C[T_1]}{T \xrightarrow{R, c \cdot AC(R,T,C[T_2])} C[T_2]}$$

The stochastic reduction semantics associates with each transition a rate which is the parameter of an exponential distribution that characterizes the stochastic behavior of the activity corresponding to the rewrite rule applied. The rate is obtained as the product of the rewriting rate constant and the application cardinality of the rule. The rewriting rate constant, obtained by instantiating the rewriting rate function of the schema from which the ground rewrite rule derives, expresses the contribution of the chosen instantiation, and the application cardinality expresses the number of reactants to which the rule can be applied and which give the same result. The higher is this value, the higher is the rate of the transition.

It is important to note that by removing the rate functions from the rule schemata we obtain rewrite rules for the standard CLS. The following proposition states that the semantics of Stochastic CLS is equivalent to the one of the standard CLS from the point of view of reachability of states.

Proposition 8.18. *Let $\mathcal{R}$ be a set of stochastic rewrite rule schemata, and let $\mathcal{R}'$ be the set of rewrite rules of the standard CLS obtained by removing all rate functions from the schemata in $\mathcal{R}$. It holds:*

$$T \to T' \iff T \xrightarrow{Rg,r} T'$$

Proof. If $T \to T'$, then an instantiation σ exists such that a rule in $\mathcal{R}'$ can be applied. The same instantiation σ can be applied to the corresponding rule schema in $\mathcal{R}$ and a stochastic ground rewrite rule is obtained. By such a ground rule we obtain $T \xrightarrow{Rg,r} T'$. The vice–versa is analogous. □

Our stochastic reduction semantics is essentially a *Continuous–time Markov Chain* (CTMC) [70], in the sense that the model obtained by applying the semantics to a given term T can be easily transformed into a CTMC.

As we recalled in Section 2.2, a CTMC can be defined as a triple $\langle \mathsf{S}, \mathsf{R}, \pi \rangle$, where S is the set of states, $\mathsf{R} : \mathsf{S} \times \mathsf{S} \mapsto \mathbb{R}^{\geq 0}$ is the *transition function*, and $\pi : \mathsf{S} \mapsto [0, 1]$ is the *starting distribution*. A state $s \in \mathsf{S}$ denotes a possible configuration of the described system. The system is assumed to pass from a configuration modeled by a state s to another one modeled by a state s' by consuming an exponentially distributed quantity of time, in which the parameter of the exponential distribution is $\mathsf{R}(s, s')$. The summation $\sum_{s' \in \mathsf{S}} \mathsf{R}(s, s')$ is called the *exit rate* of state s. Finally, the system is assumed to start from a configuration modeled by a state $s \in \mathsf{S}$ with probability $\pi(s)$, and $\sum_{s \in \mathsf{S}} \pi(s) = 1$. If the set of states of the CTMC is finite ($\mathsf{S} = \{s_1, \ldots, s_n\}$), then the transition function R can be represented as a square matrix of size n in which the element at position (i, j) is equal to $\mathsf{R}(s_i, s_j)$.

Many analysis techniques are available from mathematics and computer science for CTMCs. For example, if the set of states of the CTMC is finite, one can verify properties of the described system by using a probabilistic model checker such as PRISM [46]. For this reason we now show how to obtain a CTMC as a semantic model of Stochastic CLS.

The semantics of a term T can be transformed into a CTMC by considering CLS terms as states, by setting $\pi(T) = 1$ and by defining $\mathsf{R}(T_1, T_2)$ as the sum of the rates of all the transitions from T_1 to T_2 given by the semantics of the Stochastic CLS, namely:

$$\mathsf{R}(T_1, T_2) = \sum \left\{ r \mid T_1 \xrightarrow{Rg,r} T_2 \right\} = \sum_{Rg = Tg \xrightarrow{p} Tg' \in AR(\mathcal{R}, T_1)} p \cdot AC(Rg, T_1, T_2)$$

The set of states of the CTMC obtained by the semantics of a term T can be restricted to the set of CLS terms which are reachable from T. Obviously, if such a set of terms is finite, we obtain a finite state CTMC.

8.1.4 Simulating the Stochastic CLS

Given the CTMC of the stochastic reduction semantics, we can follow a standard simulation procedure that corresponds to Gillespie's simulation algorithm for chemical reactions [31]. Roughly speaking, the algorithm starts from the initial 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 s as parameter, and the next state s' is randomly chosen with a probability proportional to $\mathsf{R}(s, s')$.

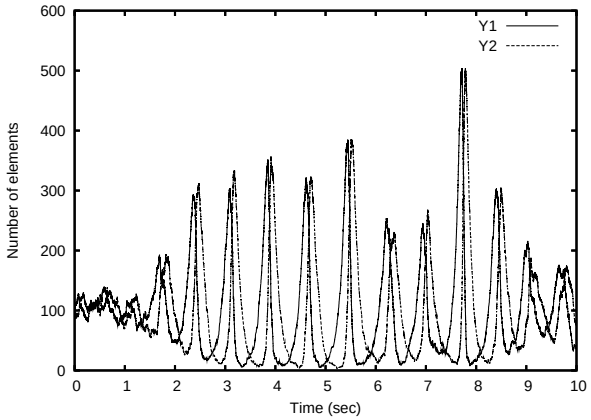


Figure 8.1: Results of simulation of the Lotka reactions.

Now we describe how the same approach can be applied to Stochastic CLS without the need of building the CTMC. A state of the simulation is a pair (T, t) where T is the current term and $t \in \mathbb{R}^{\geq 0}$ is the global clock. Assuming a finite set of rewrite rule schemata $\mathcal{R}$ and an initial term T_0, the initial state of the simulation is the pair $(T_0, 0)$.

Given a simulation state (T, t), from the stochastic reduction semantics, we have a finite set of transitions starting from T, namely the set of transitions $\{T \xrightarrow{Rg_i, r_i} T_i\}$, with $i \in [1, n]$, where r_i gives the rate of the i-th transition, and n is the number of transitions starting from T. Note that different transitions can be labeled by the same rewrite rule. Now, a simulation step transforms the state (T, t) into $(T_i, t + \tau)$ where τ is exponentially distributed with parameter $E = \sum_{i=1}^{n} r_i$ and i is chosen with probability $\frac{r_i}{E}$.

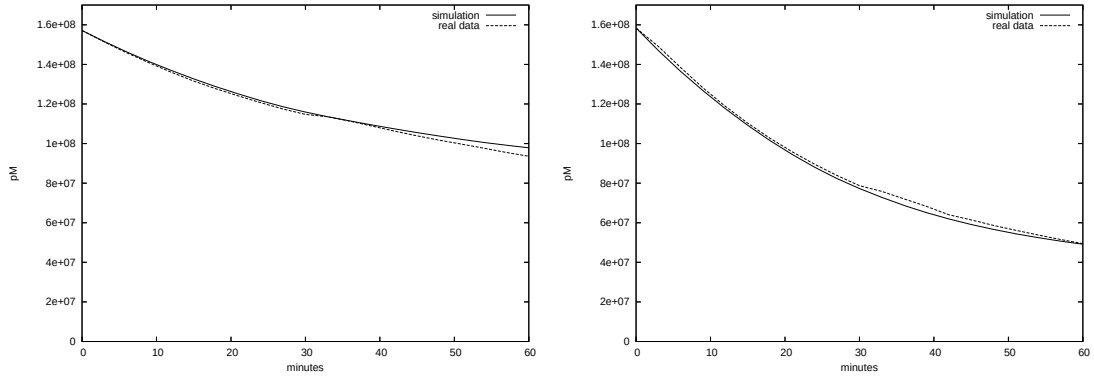
We developed a prototype simulator for the Stochastic CLS in C++ based on this simulation strategy. As a simple test, we simulated the well known Lotka reactions. Moreover, to compare the results of simulation with experimental data obtained by biologists, we simulated the reactions related to the activity of the Sorbytol Dehydrogenase enzyme in the calf eye [50] that we already studied, with different techniques, in [3, 6].

Example 8.19. The following Stochastic CLS rules model the chemical reactions known as the Lotka reactions:

$$Y_1 \overset{k_1}{\mapsto} Y_1 \,|\, Y_1 \qquad Y_1 \,|\, Y_2 \overset{k_2}{\mapsto} Y_2 \,|\, Y_2 \qquad Y_2 \overset{k_3}{\mapsto} \epsilon$$

where $k_1 = 10$, $k_2 = 0.1$ and $k_3 = 10$. In Figure 8.1 we show a simulation where the initial term contains the parallel composition of 100 Y_1 and 100 Y_2.

Example 8.20. The enzyme Sorbitol Dehydrogenase (SDH) catalyzes the reversible oxidation of Sorbitol and other polyalcohols to the corresponding keto–sugars (the accumulation of sorbitol in the calf eye has been proposed as the primary event in the development of sugar cataract in the calf). The rewrite rules modeling the reactions are shown in the



Setting	E	S	F	NADH	NADP	$E-NADH$	$E-NADP$
A	210	0	4×10^{11}	1.6×10^{8}	0	0	0
B	430	0	4×10^{11}	1.6×10^{8}	0	0	0

Figure 8.2: Sorbitol dehydrogenase: concentrations of $NADH$ with time varying. Simulations (solid lines) are compared with real experiments (dashed lines). The graph on the left corresponds to Setting A, while the graph on the right corresponds to Setting B. In the table are shown the initial quantities of the reactants.

following scheme:

$$E \,|\, NADH \underset{k_2}{\overset{k_1}{\rightleftharpoons}} ENADH \qquad E \overset{k_7}{\rightarrow} E_i$$

$$ENADH \,|\, F \underset{k_4}{\overset{k_3}{\rightleftharpoons}} ENAD^+ \,|\, S \qquad ENAD^+ \underset{k_6}{\overset{k_5}{\rightleftharpoons}} E \,|\, NAD^+$$

where E represents the enzyme Sorbitol dehydrogenase, S and F represent sorbitol and fructose, respectively, $NADH$ represents the nicotinamide adenine dinucleotide and NAD^+ is the oxidized form of $NADH$; $k_1, \ldots, k_7$ are the kinetic constants. Note that the enzyme degradation is modelled by the transformation of E into its inactive form E_i. In Figure 8.2 we show the initial values of the simulation and the results compared with the results obtained in vitro by biologists.

8.2 E.Coli Revised

Now we refine the model of the lactose operon in E.Coli given in Section 3.4 by including quantitative information. From the refinement we will obtain a model that can be used to perform simulations. We will use our prototype simulator of Stochastic CLS to simulate this gene regulation process. A detailed mathematical model of the regulation process can be found in [77]. It includes information on the influence of lactose degradation on the growth of the bacterium.

We give a Stochastic CLS model of the gene regulation process, with stochastic rates taken from [76]. As in Section 3.4, we model the membrane of the bacterium as the

looping sequence $\big(m\big)^L$, where the elementary constituent 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 element 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 elementary constituents $repr$ and $polym$, respectively. We model the mRNA of the lac Repressor as the elementary constituent $Irna$, a molecule of lactose as the elementary constituent $LACT$, and beta galactosidase, lactose permease and transacetylase enzymes as elementary constituents $betagal, perm$ and $transac$, respectively. Finally, since the three structural genes are transcribed into a single mRNA fragment (see Fig. 3.7) we model such mRNA as a single elementary constituent Rna.

The initial state of the bacterium when no lactose is present in the environment is modeled by the following term (where $n \times T$ stands for a parallel composition $T \,|\, \ldots \,|\, T$ of length n):

$$Ecoli \;::=\; \big(m\big)^L \rfloor\, (lacI\text{–}A \,|\, 30 \times polym \,|\, 100 \times repr) \tag{8.1}$$

The presence of lactose in the environment is modeled by composing $Ecoli$ in parallel with a number of $LACT$ elements as follows:

$$EcoliLact \;::=\; Ecoli \,|\, 100 \times LACT \tag{8.2}$$

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 rule schemata:

$$lacI \cdot \widetilde{x} \;\overset{0.02}{\longmapsto}\; lacI \cdot \widetilde{x} \,|\, Irna \tag{S1}$$

$$Irna \;\overset{0.1}{\longmapsto}\; Irna \,|\, repr \tag{S2}$$

$$polym \,|\, \widetilde{x} \cdot lacP \cdot \widetilde{y} \;\overset{0.1}{\longmapsto}\; \widetilde{x} \cdot PP \cdot \widetilde{y} \tag{S3}$$

$$\widetilde{x} \cdot PP \cdot \widetilde{y} \;\overset{0.01}{\longmapsto}\; polym \,|\, \widetilde{x} \cdot lacP \cdot \widetilde{y} \tag{S4}$$

$$\widetilde{x} \cdot PP \cdot lacO \cdot \widetilde{y} \;\overset{20.0}{\longmapsto}\; polym \,|\, Rna \,|\, \widetilde{x} \cdot lacP \cdot lacO \cdot \widetilde{y} \tag{S5}$$

$$Rna \;\overset{0.1}{\longmapsto}\; Rna \,|\, betagal \,|\, perm \,|\, transac \tag{S6}$$

$$repr \,|\, \widetilde{x} \cdot lacO \cdot \widetilde{y} \;\overset{1.0}{\longmapsto}\; \widetilde{x} \cdot RO \cdot \widetilde{y} \tag{S7}$$

$$\widetilde{x} \cdot RO \cdot \widetilde{y} \;\overset{0.01}{\longmapsto}\; repr \,|\, \widetilde{x} \cdot lacO \cdot \widetilde{y} \tag{S8}$$

$$repr \,|\, LACT \;\overset{0.005}{\longmapsto}\; RLACT \tag{S9}$$

$$RLACT \;\overset{0.1}{\longmapsto}\; repr \,|\, LACT \tag{S10}$$

Schemata (S1) and (S2) describe the transcription and translation of gene i into the lac Repressor (assumed for simplicity to be performed without the intervent of the RNA polymerase). Schemata (S3) and (S4) describe the binding (and unbinding) of the RNA polymerase to gene p. Schemata (S5) and (S6) 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. Schemata (S7) and (S8) describe the binding and unbinding, respectively, of the lac Repressor to gene o. Finally, schemata (S9) and (S10) describe the binding and unbinding, respectively, of the lactose to the lac Repressor.

The following schemata describe the behavior of the three enzymes for lactose degradation:

$$\left(\widetilde{x}\right)^{L} \rfloor (perm \,|\, X) \quad \overset{0.1\cdot f_1}{\longmapsto} \quad \left(perm \cdot \widetilde{x}\right)^{L} \rfloor X \tag{S11}$$

$$LACT \,|\, \left(perm \cdot \widetilde{x}\right)^{L} \rfloor X \quad \overset{0.001\cdot f_2}{\longmapsto} \quad \left(perm \cdot \widetilde{x}\right)^{L} \rfloor (LACT|X) \tag{S12}$$

$$betagal \,|\, LACT \quad \overset{0.001}{\longmapsto} \quad betagal \,|\, GLU \,|\, GAL \tag{S13}$$

where $f_1(\sigma) = occ(perm, \sigma(X)) + 1$, $f_2(\sigma) = occ(perm, \sigma(\widetilde{x})) + 1$ and $occ(a, T)$ is an in Example 8.4.

Schema (S11) describes the incorporation of the lactose permease in the membrane of the bacterium, schema (S12) the transporation of lactose from the environment to the interior performed by the lactose permease, and schema (S13) the decomposition of the lactose into glucose (denoted GLU) and galactose (denoted GAL) performed by the beta galactosidase.

The following schemata describe degradation of all the proteins and pieces of mRNA involved in the process:

$$perm \overset{0.001}{\mapsto} \epsilon \tag{S14}$$

$$betagal \overset{0.001}{\mapsto} \epsilon \tag{S15}$$

$$transac \overset{0.001}{\mapsto} \epsilon \tag{S16}$$

$$repr \overset{0.002}{\mapsto} \epsilon \tag{S17}$$

$$Irna \overset{0.01}{\mapsto} \epsilon \tag{S18}$$

$$Rna \overset{0.01}{\mapsto} \epsilon \tag{S19}$$

$$RLACT \overset{0.002}{\mapsto} LACT \tag{S20}$$

We recall that sequences are not allowed as context of application of the rules, hence the rule derived from schema (S14) cannot be applied to *perm* when it is an element of the looping sequence representing the membrane of the bacterium. This motivates the presence of the following final schema:

$$\left(perm \cdot \widetilde{x}\right)^{L} \rfloor X \quad \overset{0.001\cdot f_3}{\longmapsto} \quad \left(\widetilde{x}\right)^{L} \rfloor X \tag{S21}$$

where $f_3(\sigma) = occ(perm, \sigma(\widetilde{x})) + 1$.

8.2.1 Simulation Results

We simulated the evolution of the bacterium in the absence of lactose (modeled by the term *Ecoli* of Eq. (8.1)) and in the presence of 100 molecules of lactose in the environment (modeled by the term *EcoliLact* of Eq. (8.2)). The evolution of the two terms is given by the application of the set of rewrite rule schemata $\{(S1), \ldots, (S21)\}$.

In the first simulation we observed the variation in the number of lac Repressors, beta galactosidases, and lactose permeases, and, in the second simulation, also the speed of the entrance of the lactose from the environment into the bacterium and the speed of production of glucose molecules as the result of lactose degradation.

In Figure 8.3 we show the results of simulation when the lactose is absent. As the graph on the left shows, the number of lac Repressors inside the bacterium oscillates between 55 and 160. The graph on the right shows that the production of the beta galactosidase and lactose permease enzymes starts after more than 750 seconds and that the number of such enzymes in the bacterium is always smaller than 20. Moreover, this graph shows that the lactose permeases, once produced, become immediately part of the membrane of

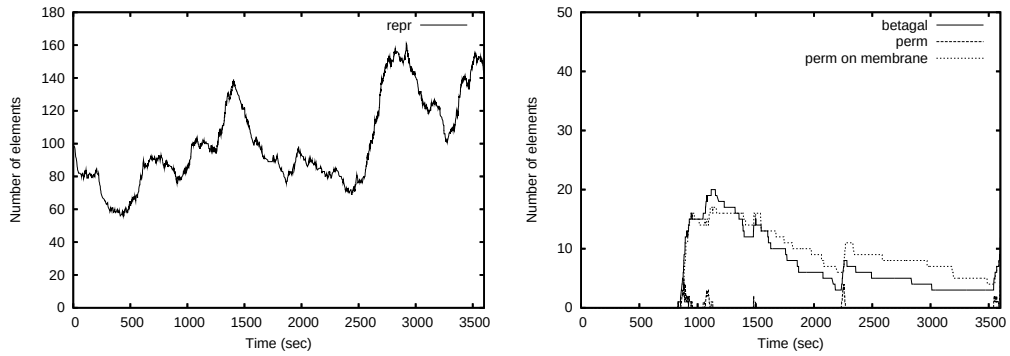


Figure 8.3: Results of simulation of the regulation process in the absence of lactose: variations in the number of lac Repressors over time (on the left), and in the number of beta galactosidase and lactose permease enzymes (on the right).

the bacterium, because the number of such enzymes not on the membrane remains always small.

In Figure 8.4 we show the results of the simulation when the lactose is present in the environment. In this simulation the production of the beta galactosidase and lactose permease enzymes start almost immediately (see the graph on the top–right).

We remark that the different times in the production of enzymes in the two simulations is not significant; in fact, the amount of time elapsed before the production of these enzymes does not depend on the presence of the lactose in the environment, because the lactose cannot enter the bacterium until some molecule of permease has been incorporated in the membrane.

Once some molecule of lactose permease joins the membrane, the lactose starts entering the bacterium. In fact, the graph on the bottom–left shows that the number of molecules in the environment rapidly decreases. Once entered, the lactose interacts with the lac Repressor: the graph on the top–left shows that about a half of the lac Repressors bind to lactose causing the number of free lac Repressors to become less than 40. In this situation the production of the beta galactosidase and lactose permease enzymes is favored; in fact the graph on the top–right shows that the number of such enzymes reaches values which oscillates around 30. At this stage, the lactose is decomposed by the beta galactosidase and, as the graph on the bottom–right shows, the production of glucose starts.

Once all the molecules of glucose have been decomposed, the number of lac Repressors increases, reaching the same values of the first simulation (see the graph on the top–left and Figure 8.3). The number of beta galactosidase and lactose permease enzymes, instead, does not decrease, and hence does not reach the values of the first simulation. This happens because the degradation of such enzymes, and of the mRNA from which they are translated, is a very slow process, which would take much more time than the time of the simulations we performed.

8.2.2 Finiteness of the Model

It is easy to see that, by applying repeatedly some rule schemata in the set $\{(S1), \ldots, (S21)\}$, it is possible to reach an infinite number of different terms. For instance, by applying

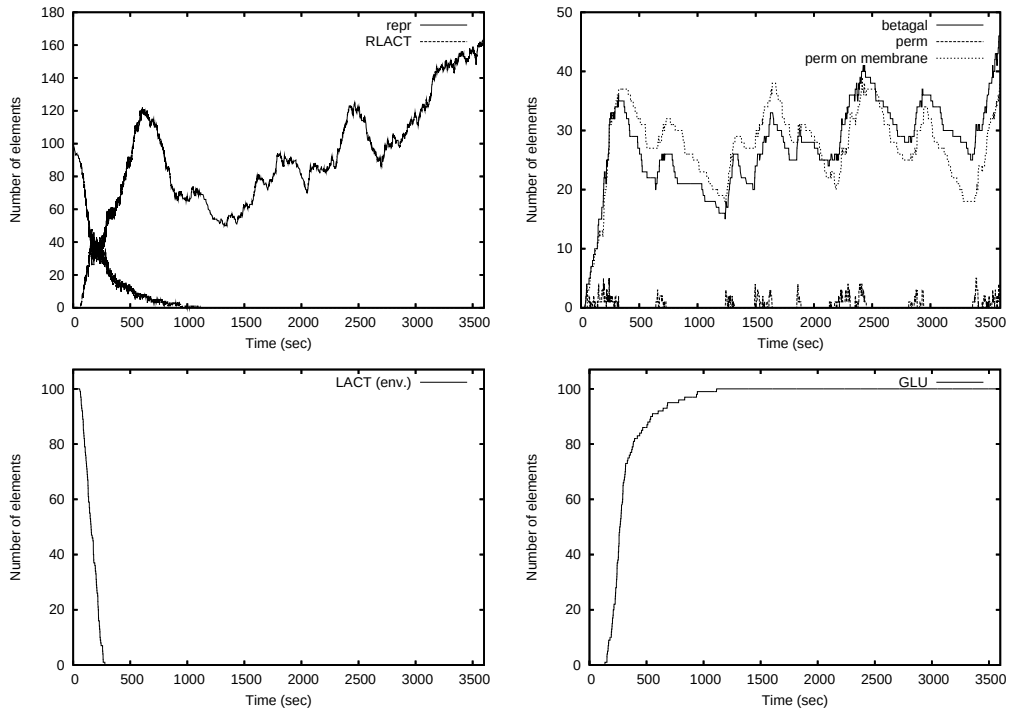


Figure 8.4: Results of simulation of the regulation process when lactose is present in the environment: variations in the number of lac Repressors over time (top–left), of beta galactosidase and lactose permease enzymes (top–right), of molecules of lactose in the environment (bottom–left), and of molecules of glucose (bottom–right).

repeatedly a ground rewrite rules derived from (S2) we obtain an infinite sequence of transitions in which, at each step, the number of repressors in the term is increased by one.

The set of reachable terms can be made finite by introducing upper bounds to the production of elements. This can be done, for example in the case of the repressor elements, by replacing rule schema (S2) with:

$$reprP \cdot \widetilde{x} \,|\, Irna \quad \overset{0.1}{\longmapsto} \quad Irna \,|\, repr \qquad \text{(S2')}$$

where $reprP$ denotes a potential repressor element. In this way the number of repressors that can be created is bounded by the number of potential repressors in the initial term. In order to have for (S2') always the same application rate as (S2), one must insert all the potential repressors of the initial term into a sequence, such as

$$reprP \cdot \ldots \cdot reprP \cdot reprT$$

where $reprT$ denotes a terminator for the sequence. Now, in order to correctly handle also the degradation of the repressors, one may replace rule schema (S17) with

$$repr \,|\, \widetilde{x} \cdot reprT \quad \overset{0.002}{\longmapsto} \quad reprP \cdot \widetilde{x} \cdot reprT \qquad \text{(S17')}$$

The same approach can be used for rule schemata (S1),(S5), and (S6), and for the corresponding degradation schemata.

As an alternative approach, one could include in the formalism conditions for the applicability of rules (as in Full–CLS, see Section 3.1), and in particular require that a rule is applied only if the occurrences of a particular element in the term are less than a given number.

Chapter 9

Translating Kohn's Molecular Interaction Maps into Stochastic CLS

The definition of a diagrammatic graphical language able to describe biochemical networks in a clearly visible and unambiguous way is an important step towards the understanding of cell regulatory mechanisms. One of the most well designed and rigidly defined proposals of graphical language are Kohn's Molecular Interaction Maps (MIMs) [1, 41, 43]. In these maps, biochemical components of bioregulatory networks are depicted using a notation similar to the "wiring diagrams" used in electronics, and various types of interactions that may occur between the components can be represented. Interactions includes complex formations, phosphorylations, enzyme catalysis, stimulation and inhibition of biochemical reactions, DNA transcription, etc. ...

The use of a single MIM diagram to describe all the many interactions in a biochemical network allows the tracing of pathways within the network, for instance with the aid of computer simulation. However, even if the meaning of MIM symbols is often clear and easy to understand, there is a lack of mathematical interpretation for some of them, hence some diagrams cannot be used directly as an input for a simulation tool. This is confirmed by the distinction made by Kohn in [41] between *heuristic maps*, that may include any symbol, and *explicit maps*, that may include a few symbols having a clear mathematical interpretation. The conclusion of Kohn is that only the latter should be used to perform simulations, by translating them into a list of chemical reactions.

In this chapter we face the problem of allowing the simulation of a larger set of diagrams than the explicit ones. In particular, we consider the Stochastic CLS formalism defined in the previous chapter and we show that by translating MIMs into Stochastic CLS terms and rewrite rules we can simulate more than the set of explicit diagrams. For the sake of simplicity, the translation will be presented mainly by specific examples by starting from basic diagrams (which essentially correspond to explicit diagrams) and by including step by step features like contingency symbols, membranes, multi–site DNA fragments and multi–domain species.

As regards related work, in [18] a simple example of MIM has been modeled using the Beta–binders formalism [63]. Moreover, other graphical languages for biochemical networks have been defined recently [19, 44, 57]. Among these, the notation introduced in

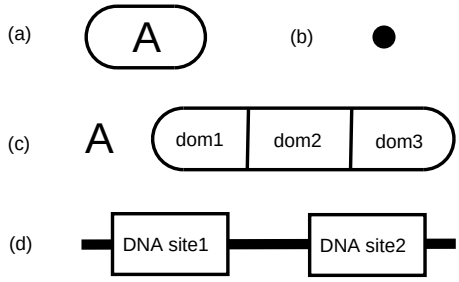


Figure 9.1: Species in MIMs.

[44] (which has been compared with MIMs in [42]) seems to be another promising proposal, as it has been used to model a real complex example of signalling pathway [56] and it is supported by useful software tools [30]. A different approach to the graphical description of biochemical networks based on graph rewriting is proposed in [23, 28].

In this chapter we recall the definition of Kohn's Molecular Interaction Maps (MIMs) and we show how them can be translated into Stochastic CLS. We refer to the definition of MIMs that can be found in [43], as it is the most recent and the most complete available in the literature. We present both MIMs and their translation incrementally, by showing first the diagrams for basic molecular interactions, then their extension with contingency symbols, then the extension with compartments, and finally the extensions with multi–site DNA fragments and multi–domain species.

A species in a MIM is depicted as a box containing the species name (Fig. 9.1.a). In the case of a DNA site, the box is placed over a thick line representing a DNA strand, and more than one site can be placed over the same line (Fig. 9.1.d). Multi–domain species, typically proteins, are depicted as boxes divided into slices, one slice for each domain (Fig. 9.1.c). Also a bullet (Fig. 9.1.b) is used to denote a species when it is the result of a reaction, and to denote different instances of the same species (see Fig 9.3).

9.1 Basic Diagrams

Basic MIM diagrams are composed by single–domain species and single–site DNA fragments related each other by some reaction symbols. Reaction symbols are arrows, and they are listed in Fig. 9.2. In the figure, arrow (a) connects two species and denotes the reversible binding of them; (b) points to one species and denotes a covalent modification (phosphorylation, acetylation, etc. . .), the type of the modification is usually written at the tail of the arrow; (c) connects two species and denotes a covalent binding; (d) connects two species and denotes a stoichiometric conversion, namely the species at the tail of the arrow disappears while the pointed one appears; (e) is like (d) without the loss of the species at the tail of the arrow; (f) connects a DNA strand and a species and denotes DNA transcription; (g) represents the cleavage of a covalent bond (see Fig. 9.3); finally, (h) is connected to a single species and represents its degradation.

An example of diagram is shown in Fig. 9.3. In the example, a DNA strand is transcribed into a piece of RNA that could be translated into enzyme E1 or could be degraded. Phosphorylation activates E1 which binds molecule A. The formed complex may be trans-

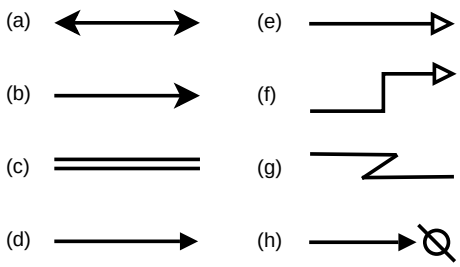


Figure 9.2: Reaction symbols in basic diagrams.

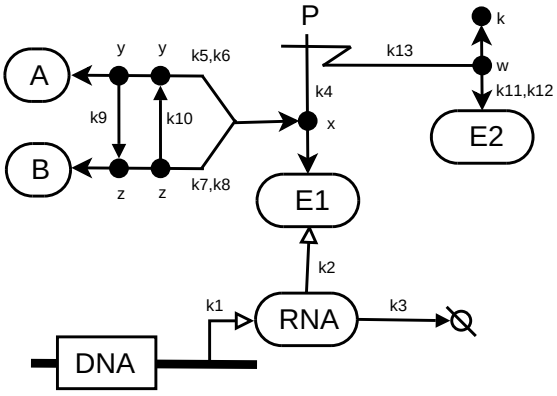


Figure 9.3: An example of basic diagram: x is E1 phosphorylated; y and z are the complexes E1:A and E1:B, respectively; k is another instance of E2; w is the homodimerization of E2. Labels k1,. . . ,k13 are kinetic constants: in reversible reactions the first constant in the one for binding, and the second is the one for dissociation of the reactants.

formed into the complex in which A has been replaced by B, and such a new bond can be broken releasing B and E1 phosphorylated (all the reactions involving A and B are reversible). Finally, E1 can be dephosphorylated, and hence deactivated, by a homodimer composed by two instances of E2.

Now we show how a basic diagram can be translated into a set of Stochastic CLS (ground) rewrite rules. We consider as the Stochastic CLS alphabet $\mathcal{E}$ the set of all species appearing in the diagram, including those denoted by bullet and obtained as the result of a reaction. In $\mathcal{E}$ we denote with A:B (without any ordering, hence A:B = B:A) the binding of species A and B, and we denote covalent modifications as follows: A_P denotes phosphorylated A, B_Ac denotes acetylated B, etc.... Reaction symbols can be translated into rewrite rules as shown in Figure 9.4 where c is Gillespie's *stochastic reaction constant* obtained from the corresponding kinetic constant k (see Section 2.3 for details). As an example, the map in Fig. 9.3 can now be translated into the following set of rewrite rules:

$$DNA \overset{c1}{\mapsto} DNA|RNA \quad RNA \overset{c2}{\mapsto} RNA|E1 \quad RNA \overset{c3}{\mapsto} \epsilon$$

$$E1 \overset{c4}{\mapsto} E1\_P \quad E1|A \overset{c5}{\mapsto} E1{:}A \quad E1{:}A \overset{c6}{\mapsto} E1|A \quad E1|B \overset{c7}{\mapsto} E1{:}B$$

$$E1{:}B \overset{c8}{\mapsto} E1|B \quad E1{:}A \overset{c9}{\mapsto} E1{:}B \quad E1{:}B \overset{c10}{\mapsto} E1{:}A$$

$$E2|E2 \overset{c11}{\mapsto} E2{:}E2 \quad E2{:}E2 \overset{c12}{\mapsto} E2|E2 \quad E2{:}E2|E1\_P \overset{c13}{\mapsto} E2{:}E2|E1$$

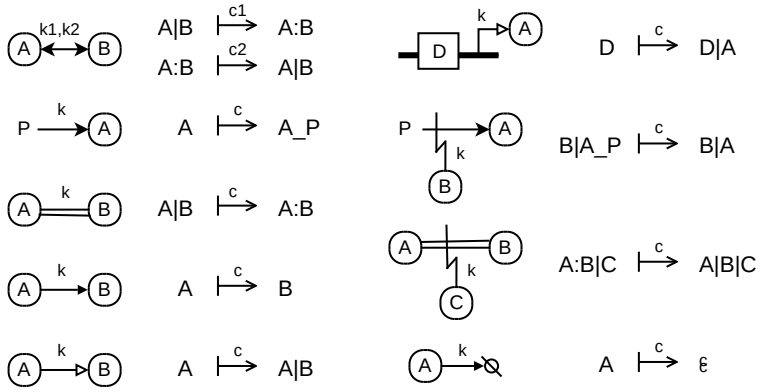


Figure 9.4: Translation of reaction symbols into rewrite rules.

Note that the homodimerization of E2 in the example has been translated into a rule (the eleventh) with two instances of E2 in the left hand side.

Now, in order to perform simulations, one has only to provide a Stochastic CLS term representing the initial state of the modeled system. A possible initial term in the given example could be DNA | E2 | E2 | E2 | E2 | E2 representing a state in which E1 has not been produced (yet) and five instances of E2 are present.

9.2 Contingency Symbols

Contingency symbols are arrows connecting a species and a reaction, and describing the influence of the species on the rate of the reaction connected with it. As pointed out in [43], contingency symbols may introduce ambiguities in the meaning of maps, hence the authors of that paper suggest avoiding them in diagrams that must be used as an input for simulation. However, we show that by giving those symbols a precise (limited) meaning, we can use them in simulations.

Contingency symbols are shown in Fig. 9.5: arrow (a) denotes stimulation of the pointed reaction; (b) is similar to (a), but requires that some instances of the species behind the arrow are present in the system to permit the pointed reaction to occur; (c) is inhibition of the pointed reaction; and (d) means that the species at the tail of the arrow is an enzyme catalyzing the pointed reaction. Now, our interpretation of the contingency symbols is the following: (a) replaces the kinetic constant of the pointed reaction with a smaller one; (b) sets the kinetic constant of the reaction and the reaction cannot occur if some instances of the species behind the arrow is not present; (c) is similar to (a) but introduces a constant bigger than the original one; (d) can be avoided and replaced with a few reaction symbols describing the interactions between the enzyme and the substrates (see [43] for details). Since (a) and (c) have the same behavior (they replace a kinetic constant with another one) we will use only (a) in what follows. As we shall see, the stimulation and inhibition arrows will have another role with a different meaning in gene regulation. When more than one contingency arrow points to the same reaction, some *state combination symbols* must be used to disambiguate the choice of the kinetic constant. A state combination symbol is a line connecting two species, and denotes the state in which

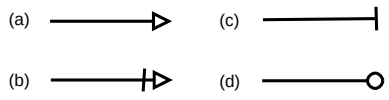


Figure 9.5: Contingency symbols.

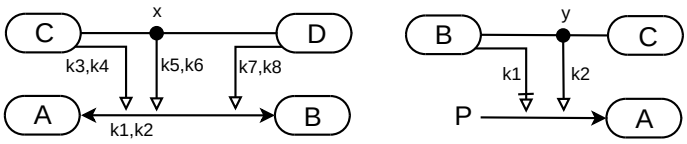


Figure 9.6: Two examples of usage of contingency symbols: x denotes the state in which both C and D are present in the system, and y the state in which both B and C are present.

some instances of both species are present in the system.

In Fig. 9.6 we show two examples of usage of contingency symbols. On the left, the reversible binding of species A and B is stimulated by species C and D. In the absence of both C and D the reaction rates are k1 and k2. When either C, or D, or both C and D are present, the reaction rates becomes the ones labeling the corresponding stimulation arrows. On the right, the phosphorylation of A is stimulated by B and C, with B required to permit the reaction to occur. The kinetic constant on the phosphorylation arrow is not present because the reaction cannot occur without the presence of B. For the same reason, also the simulation arrow coming from C is missing.

Now, the idea at the base of the translation of contingency symbols into Stochastic CLS is the use of a term variable in the rule of the stimulated reaction modeling the environment where the reaction occurs, and of a rate function in the rewrite rule which gives a different kinetic constant depending on which stimulating species are present in the instantiation of the variable. In order to ensure that the added term variable matches the whole reaction context, we have to enclose both sides of the obtained rewrite rule (and also the initial simulation term) into a $\left(\epsilon\right)^L \rfloor \Box$ context.

For example, the maps shown in Fig. 9.6 can be translated into Stochastic CLS rules as follows. From the diagram on the left we obtain:

$$\left(\epsilon\right)^L \rfloor (X \mid A \mid B) \overset{f_1}{\mapsto} \left(\epsilon\right)^L \rfloor (X \mid A{:}B)$$
$$\left(\epsilon\right)^L \rfloor (X \mid A{:}B) \overset{f_2}{\mapsto} \left(\epsilon\right)^L \rfloor (X \mid A \mid B)$$

$$f_1 = \begin{cases} c3 & \text{if } \sigma(X) \equiv C|T \\ & \text{and } T \not\equiv D|T' \\ c7 & \text{if } \sigma(X) \equiv D|T \\ & \text{and } T \not\equiv C|T' \\ c5 & \text{if } \sigma(X) \equiv C|D|T \\ c1 & \text{otherwise} \end{cases} \qquad f_2 = \begin{cases} c4 & \text{if } \sigma(X) \equiv C|T \\ & \text{and } T \not\equiv D|T' \\ c8 & \text{if } \sigma(X) \equiv D|T \\ & \text{and } T \not\equiv C|T' \\ c6 & \text{if } \sigma(X) \equiv C|D|T \\ c2 & \text{otherwise} \end{cases}$$

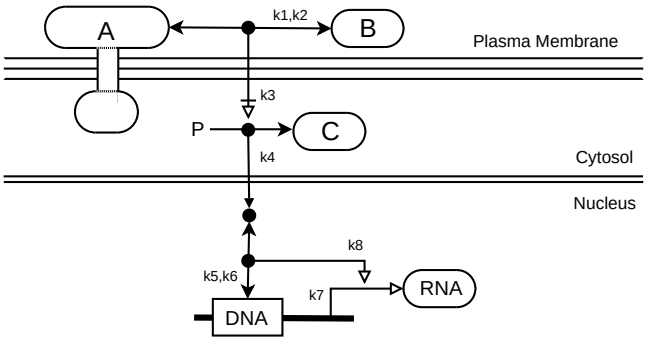


Figure 9.7: An example of signalling pathway: molecule A outside the cell binds to a receptor protein placed in the plasma membrane. This activates phosphorylation of C in the cytosol. Once phosphorylated, protein C enters the nucleus and binds to the DNA stimulating the transcription of some genes.

and from the diagram on the right:

$$\left(\epsilon\right)^L \rfloor (X \mid A) \stackrel{f}{\mapsto} \left(\epsilon\right)^L \rfloor (X \mid A\_P) \quad f = \begin{cases} c_1 & \text{if } \sigma(X) \equiv B|T \\ & \text{and } T \not\equiv C|T' \\ c_2 & \text{if } \sigma(X) \equiv B|C|T \\ 0 & \text{otherwise} \end{cases}$$

for some terms T, T' and where, as before, c_i denotes Gillespie's simulation constant obtained from k_i. An example of initial term for the first set of rules is $\left(\epsilon\right)^L \rfloor (A|A|A|B|B|B|C|D)$, and for the second is $\left(\epsilon\right)^L \rfloor (A|A|B|C)$.

We remark that our interpretation of the contingency symbols as means to replace kinetic constants could be in many cases not appropriate, for instance because this mechanism does not take into account the concentration of the stimulating species. A more precise interpretation would use as label of a contingency symbol a function from the concentration of the stimulating species to real values, and this function could be used in the translation of the symbol into Stochastic CLS. We avoided this interpretation for the sake of simplicity, and in what follows we show that however our choice is precise enough to model gene regulation systems.

9.3 Compartments

MIMs are often used to describe systems whose evolution may be given by intra– and extra–cellular interactions, by interactions mediated by proteins placed in the plasma membrane, and by interactions in the nucleus of an eukaryotic cell. Examples of systems which include all these kinds of interactions are signalling pathways. To correctly describe this division of the environment into different compartments, membranes are included in MIMs as shown in Fig. 9.7.

Membranes can be easily translated into Stochastic CLS by using looping sequences. We assume each membrane to have a unique identifying symbol in $\mathcal{E}$: a membrane is hence a looping sequence composed by such a symbol and by other symbols representing

the instances of the species of the diagram that are placed on the membrane. Looping sequences must be nested in accordance with the relative position of the membranes in the diagram, and the reactions of a molecule on a membrane must be translated by taking into account that the molecule symbol is placed in a looping sequence. As an explanation, we show the translation of the diagram of Fig. 9.7. We assume P and N to be the symbols identifying the plasma membrane and the membrane of the nucleus, respectively. The set of Stochastic CLS rules obtained from the translation is the following:

$$\left(P \cdot \widetilde{x} \cdot A \cdot \widetilde{y}\right)^L \rfloor X \mid B \overset{c1}{\mapsto} \left(P \cdot \widetilde{x} \cdot A{:}B \cdot \widetilde{y}\right)^L \rfloor X$$

$$\left(P \cdot \widetilde{x} \cdot A{:}B \cdot \widetilde{y}\right)^L \rfloor X \overset{c2}{\mapsto} \left(P \cdot \widetilde{x} \cdot A \cdot \widetilde{y}\right)^L \rfloor X \mid B$$

$$\left(P \cdot \widetilde{x}\right)^L \rfloor (X \mid C) \overset{f}{\mapsto} \left(P \cdot \widetilde{x}\right)^L \rfloor (X \mid C\_P)$$

$$\text{with } f = c3 \text{ if } A{:}B \in \widetilde{x}, \text{ and } f = 0 \text{ otherwise}$$

$$C\_P \mid \left(N \cdot \widetilde{x}\right)^L \rfloor X \overset{c4}{\mapsto} \left(N \cdot \widetilde{x}\right)^L \rfloor (C\_P \mid X)$$

$$\left(N \cdot \widetilde{x}\right)^L \rfloor (C\_P \mid DNA \mid X) \overset{c5}{\mapsto} \left(N \cdot \widetilde{x}\right)^L \rfloor (C\_P{:}DNA \mid X)$$

$$\left(N \cdot \widetilde{x}\right)^L \rfloor (C\_P{:}DNA \mid X) \overset{c6}{\mapsto} \left(N \cdot \widetilde{x}\right)^L \rfloor (C\_P \mid DNA \mid X)$$

$$\left(N \cdot \widetilde{x}\right)^L \rfloor (DNA \mid X) \overset{c7}{\mapsto} \left(N \cdot \widetilde{x}\right)^L \rfloor (DNA \mid RNA \mid X)$$

$$\left(N \cdot \widetilde{x}\right)^L \rfloor (C\_P{:}DNA \mid X) \overset{c8}{\mapsto} \left(N \cdot \widetilde{x}\right)^L \rfloor (C\_P{:}DNA \mid RNA \mid X)$$

The use of looping sequences representing membranes permits identifying exactly where the modeled reaction can occur. For instance, the reversible reaction modeled by the first two rules can occur only on the plasma membrane, and not, for instance, on the membrane of the nucleus. Moreover, looping sequences representing membranes take the place of $\left(\epsilon\right)^L$ in rules modeling contingency symbols, when in the translated diagram such symbols are inside a membrane. For instance, the phosphorylation described by the third rule is stimulated by the binding of A and B: here, the looping sequence $\left(P \cdot \widetilde{x}\right)^L$ plays both the roles of identifying the position where the reaction can occur, and of ensuring the maximal matching of variable X as usually done by $\left(\epsilon\right)^L$ in the translation of contingency symbols. We remark that the stimulation of the DNA transcription modeled by the last two rules is not modeled as usual: we explain gene regulation through binding of DNA sites in the next subsection. Finally, a possible initial term for the translation of the example in Fig. 9.7 is $B \mid B \mid B \mid \left(P \cdot A \cdot A\right)^L \rfloor (C \mid C \mid \left(N\right)^L \rfloor DNA)$.

9.4 Multi–Site DNA and Gene Regulation

Multi–site DNA species can be used in MIMs to model complex gene regulatory networks, in which promoter and inhibitor proteins may bind different DNA sites causing variations in the rate of transcription of the DNA. An example of regulatory network is shown in Fig. 9.8, where the transcription of some genes is regulated by two promoters P1 and P2, and one repressor R. In the diagram, the meaning of the stimulation symbols is as usual, while the inhibition symbol means that if an instance of R is bound to site1, then the transcription of the DNA cannot be performed.

We model a multi–site DNA fragment as a Stochastic CLS sequence composed by one element for each site. The regulation network can be translated into a set of rewrite rules,

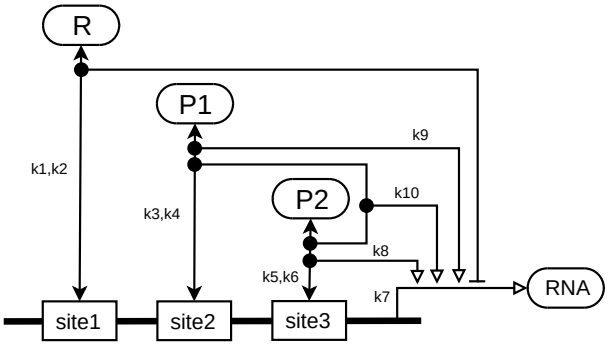


Figure 9.8: An example of gene regulation.

one for each possible combination of promoters and inhibitors bound to the DNA. The diagram in Fig. 9.8 can be translated into the following set of rules:

$$R \,|\, site1 \cdot \widetilde{x} \overset{c1}{\mapsto} site1{:}R \cdot \widetilde{x} \qquad site1{:}R \cdot \widetilde{x} \overset{c2}{\mapsto} R \,|\, site1 \cdot \widetilde{x}$$

$$P1 \,|\, \widetilde{x} \cdot site2 \cdot \widetilde{y} \overset{c3}{\mapsto} \widetilde{x} \cdot site2{:}P1 \cdot \widetilde{y}$$

$$\widetilde{x} \cdot site2{:}P1 \cdot \widetilde{y} \overset{c4}{\mapsto} P1 \,|\, \widetilde{x} \cdot site2 \cdot \widetilde{y}$$

$$P2 \,|\, \widetilde{x} \cdot site3 \overset{c5}{\mapsto} \widetilde{x} \cdot site3{:}P2 \qquad \widetilde{x} \cdot site3{:}P2 \overset{c6}{\mapsto} P2 \,|\, \widetilde{x} \cdot site3$$

$$site1 \cdot site2 \cdot site3 \overset{c7}{\mapsto} site1 \cdot site2 \cdot site3 \,|\, RNA$$

$$site1 \cdot site2{:}P1 \cdot site3 \overset{c8}{\mapsto} site1 \cdot site2{:}P1 \cdot site3 \,|\, RNA$$

$$site1 \cdot site2 \cdot site3{:}P2 \overset{c9}{\mapsto} site1 \cdot site2 \cdot site3{:}P2 \,|\, RNA$$

$$site1 \cdot site2{:}P1 \cdot site3{:}P2 \overset{c10}{\mapsto} site1 \cdot site2{:}P1 \cdot site3{:}P2 \,|\, RNA$$

The first six rules are the translation of the three reversible reactions appearing in the diagram. Each of these reactions regards only one site of the DNA, hence sequence variables have been used in the rules to allow them to be applied independently from the state of the other DNA sites. The last four rules describe the regulation process. One rules is present for each combination of promoters and reactants that allows the transcription, and no rule is present for those combinations for which transcription is forbidden, namely the ones in which the repressor is bound to site1. A possible initial term for this example of gene regulation is $site1 \cdot site2 \cdot site3 \,|\, R \,|\, R \,|\, P1 \,|\, P1 \,|\, P1 \,|\, P2 \,|\, P2$.

9.5 Multi–Domain Species

To model multi–domain species with Stochastic CLS one might think of using sequences as in the case of multi–site DNA strands. Unfortunately, the use of sequences in this case does not allow a correct modeling of the complexes that could be produced by the interactions between domains of different species. As an example, consider the diagram in Fig. 9.9, where two species A and B, each having two domains, interact each other by establishing a covalent bond. The two species could be modeled as the sequences $A1 \cdot A2$ and $B1 \cdot B2$, and the result of their phosphorylations could be modeled as $A1\_P \cdot A2$ and

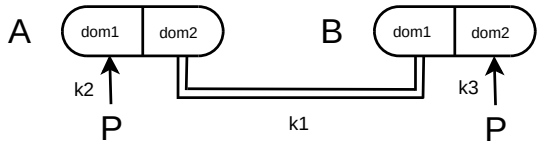


Figure 9.9: An example diagram with multi–domain species.

$B1 \cdot B2\_P$, but how to model the $A{:}B$ complex in a way that keeps in working order the domains not involved in the binding? The use of LCLS instead of CLS would help in this situation. In fact LCLS has been defined exactly to allow a easier description of these interactions at the domain level. By using CLS, the only general solution we have found to this problem is to avoid the use of sequences in the modeling of a multi–domain species in favor of the use of a different alphabet symbol for each possible state of the species.

For instance, in the translation of the diagram in Fig. 9.9, the alphabet $\mathcal{E}$ should contain the following symbols:

$$A \ , \ B \ , \ A\_P \ , \ B\_P \ , \ A{:}B \ , \ A\_P{:}B \ , \ A{:}B\_P \ , \ A\_P{:}B\_P.$$

Now, reaction symbols should be translated by taking into account all the possible states of the involved species in which the reaction may occur, and by generating rewrite rules for all these possible states. As regards the example in the figure, the rewrite rules obtained by translating the symbol of covalent binding are the following:

$$A \,|\, B \overset{c1}{\mapsto} A{:}B \qquad\qquad A\_P \,|\, B \overset{c1}{\mapsto} A\_P{:}B$$
$$A \,|\, B\_P \overset{c1}{\mapsto} A{:}B\_P \qquad\qquad A\_P \,|\, B\_P \overset{c1}{\mapsto} A\_P{:}B\_P$$

and the rules obtained from the translation of the two phosphorylations are the following:

$$A \overset{c2}{\mapsto} A\_P \qquad\qquad B \overset{c3}{\mapsto} B\_P$$
$$A{:}B \overset{c2}{\mapsto} A\_P{:}B \qquad\qquad A{:}B \overset{c3}{\mapsto} A{:}B\_P$$
$$A{:}B\_P \overset{c2}{\mapsto} A\_P{:}B\_P \qquad\qquad A\_P{:}B \overset{c3}{\mapsto} A\_P{:}B\_P$$

Chapter 10

Conclusions

In this thesis we have developed new formalisms for the description and the analysis of systems of Cell Biology. We have tried to obtain formalisms having simple notations, having the ability of describing biological systems at different levels of abstraction and being flexible enough to allow describing new kinds of phenomena without the need of defining extension of them. To obtain these results, we have based our formalisms on term rewriting and we have chosen to abstract biological structures into simpler structures (from the Computer Science point of view) such as (possibly circular) sequences of symbols. Moreover, we have not imposed any restriction on the rewrite rules one can define to model a biological system.

We have obtained a very expressive formalism, called Calculus of Looping Sequences (CLS), which can be used to model a wide range of biological systems (we have provided some guidelines for the modeling of biological entities and biological events in CLS). To show the expressiveness of CLS we have given several examples of models of real systems, and we have shown how two other well-established related formalisms can be encoded into CLS. We have proposed bisimulations as formal analysis tools for biological systems: we have defined them for CLS and for the simplest of Brane Calculi, we have compared the two definitions and used those of CLS to verify a causality property on the model of a real example of biological system. Moreover, we have faced the problem of modeling also quantitative aspects of biological systems, such as time and probabilities of the occurrences of events. In particular, we have defined a stochastic extension of CLS, we have developed a simulator based on this extended formalism and we have used it to analyze a real example of gene regulation. Finally, we have provided a translation of Kohn's Molecular Interaction Maps (MIMs) into our stochastic formalism, so to allow simulating systems described by using these maps.

We believe that the main feature of the formalisms we have proposed is in the fact that they are very general, but at the same time quite simple. Generality means that systems can be described at different abstraction levels and without restriction to any particular class of systems, and simplicity means that the notation of the formalism is readable and the semantics is compact. However, there are many improvements that can be made, and which we leave as future work.

A first improvement could be the introduction of some form of commutativity in the modeling of the objects constituting membranes, so to obtain a formalism that allow modeling membranes in a more natural manner. We have briefly considered an extension of

CLS with this feature in Chapter 4 with the definition of CLS+, but this extension could be investigated more in deep. Moreover, a further step towards a more complete description of biomolecular entities and membranes could be the description of their positions, sizes and shapes in a three–dimensional space.

Moreover, it would be interesting to study different rewrite rules formats for CLS under the point of view of both the biological and the computational expressive powers.

As regards quantitative aspects of biological systems, an extension of CLS could be defined which takes into account the size of and the distances between the components of the described systems. This would allow describing multi–cellular systems and growth phenomena, as embryos during their first stages of development.

Bibliography

[1] M.I. Aladjem, S. Pasa, S. Parodi, J.N. Weinstein, Y.Pommier and K.W. Kohn. "Molecular Interaction Maps–A Diagrammatic Graphical Language for Bioregulatory Networks". Science's STKE, volume 2004, number 222, pages pe8, 2004.

[2] R. Alur, C. Belta, F. Ivancic, V. Kumar, M. Mintz, G.J. Pappas, H. Rubin and J. Schug. "Hybrid Modeling and Simulation of Biomolecular Networks". Hybrid Systems: Computation and Control, LNCS 2034, pages 19–32, Springer, 2001.

[3] R. Barbuti, S. Cataudella, A. Maggiolo-Schettini, P. Milazzo and A. Troina. "A Probabilistic Model for Molecular Systems". Fundamenta Informaticae, volume 67, pages 13–27, 2005.

[4] R. Barbuti, A. Maggiolo-Schettini, and P. Milazzo. "Extending the Calculus of Looping Sequences to Model Protein Interaction at the Domain Level". Int. Symposium on Bioinformatics Research and Applications (ISBRA'07), LNBI 4463, pages 638–649, Springer, 2006.

[5] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, P. Tiberi and A. Troina. "Stochastic CLS for the Modeling and Simulation of Biological Systems". Submitted to Bioinformatics.

[6] R. Barbuti, A. Maggiolo–Schettini, P. Milazzo, and A. Troina. "An Alternative to Gillespie's Algorithm for the Simulation of Chemical Reactions". Computational Methods in Systems Biology (CMSB'05).

[7] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo and A. Troina. "A Calculus of Looping Sequences for Modelling Microbiological Systems". Fundamenta Informaticae, volume 72, pages 21–35, 2006.

[8] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. "Bisimulation Congruences in the Calculus of Looping Sequences". Int. Colloquium on Theoretical Aspects of Computing (ICTAC'06), LNCS 4281, pages 93–107, Springer, 2006.

[9] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. "Bisimulations in Calculi Modelling Membranes". Submitted to Formal Aspects of Computing.

[10] R. Barbuti, A. Maggiolo-Schettini, P. Milazzo, and A. Troina. "The Calculus of Looping Sequences for Modeling Biological Membranes". 8th Workshop on Membrane Computing (WMC8), LNCS, Springer, to appear.

[11] R. Blossey, L. Cardelli, and A. Phillips. "A Compositional Approach to the Stochastic Dynamics of Gene Networks", Transactions on Computational Systems Biology IV, LNCS 3939, pages 99–122, Springer, 2006.

[12] Caenorhabditis elegans WWW Server. web site. `http://elegans.swmed.edu/`.

[13] L. Cardelli. "Brane Calculi. Interactions of Biological Membranes". CMSB'04, LNCS 3082, pages 257–280, Springer, 2005.

[14] L. Cardelli and A.D. Gordon. "Mobile Ambients". Theoretical Computer Science, volume 240, number 1, pages 177–213, 2000.

[15] L. Cardelli and G. Păun. "A Universality Result for a (Mem)Brane Calculus Based on Mate/Drip Operations" Int. Journal of Foundations of Computer Science, volume 17, number 1, pages 49–68, 2006.

[16] N. Chabrier-Rivier, M. Chiaverini, V. Danos, F. Fages and V. Schachter. "Modeling and Querying Biomolecular Interaction Networks". Theoretical Computer Science, volume 325, number 1, pages 25-44, 2004.

[17] K.C. Chen, L. Calzone, A. Csikasz–Nagy, F.R. Cross, B. Novak, and J.J. Tyson. "Integrative Analysis of Cell Cycle Control in Budding Yeast". Molecular Biology of the Cell, volume 15, number 8, pages 3841–3862, 2004.

[18] F. Ciocchetta, C. Priami and P. Quaglia. "Modeling Kohn Interaction Maps with Beta-Binders: An Example". Transactions on Computational Systems Biology III, LNCS Subline, volume 3737, pages 22–48, Springer, 2005.

[19] D.L. Cook, J.F. Farley, S.J. Tapscott. "A Basis for a Visual Language for Describing, Archiving and Analyzing Functional Models of Complex Biological Systems". Genome Biology, volume 2, number 4, pages RESEARCH0012, 2001.

[20] M. Curti, P. Degano, C. Priami and C.T. Baldari. "Modelling Biochemical Pathways through Enhanced pi-calculus". Theoretical Computer Science, volume 325, number 1, pages 111–140, 2004.

[21] W.Damm and D. Harel. "LSCs: Breathing Life into Message Sequence Charts". Formal Methods in System Design, volume 19, number 1, 2001.

[22] Z. Dang and O.H. Ibarra. "On P Systems Operating in Sequential and Limited Parallel Modes", Workshop on Descriptional Complexity of Formal Systems, pages 164–177, 2004.

[23] V. Danos and C. Laneve. "Formal Molecular Biology". Theoretical Computer Science, volume 325, number 1, pages 69–110, 2004.

[24] V. Danos and S. Pradalier. "Projective Brane Calculus", Computational Methods in Systems Biology (CMSB'04), LNCS 3082, pages 134–148, Springer, 2005.

[25] N. Derschowitz. "Termination of Rewriting". Journal of Symbolic Computation, volume 3, pages 69–116, 1987.

[26] E–CELL web site: `http://ecell.sourceforge.net/`.

[27] S. Efroni, I.R. Choen, and D. Harel. "Toward Rigorous Comprehension of Biological Complexity: Modeling, Execution and Visualization of Thymic t–cell Maturation". Genome Research, volume 13, pages 2485–2497, 2003.

[28] J.R. Faeder, M.L. Blinov, W.S. Hlavacek. "Graphical Rule-Based Representation of Signal-Transduction Networks". Symposium on Applied Computing (SAC), ACM, pages 133–140, 2005.

[29] C. Flanagan and M. Abadi. "Object Types Against Races". CONCUR'99, LNCS 1664, pages 288-303, Springer, 1999.

[30] A. Funahashi, M. Morohashi and H. Kitano."CellDesigner: a Process Diagram Editor for Gene–Regulatory and Biochemical Networks". BIOSILICO, volume 1, number 5, pages 159–162, 2005.

[31] D. Gillespie. "Exact Stochastic Simulation of Coupled Chemical Reactions". Journal of Physical Chemistry, volume 81, pages 2340–2361, 1977.

[32] A. Gordon and P. Hankin. "A Concurrent Object Calculus: Reduction and Typing". High-Level Concurrent Languages (HLCL'98), Elsevier ENTCS, volume 16, number 3, 1998.

[33] N. Götz, U. Herzog, and M. Rattelbach. "TIPP – A Stochastic Process Algebra". Workshop on Process Algebras and Performance Modelling (PAPM'93), pages 31–36, 1993.

[34] D. Harel. "Statecharts: A Visual Formalism for Complex Systems". Science of Computer Programming, volume 8, number 3, pages 231–274, 1987.

[35] D. Harel. "A Grand Challenge: Full Reactive Modeling of a Multi-cellular Animal". Bulletin of the EATCS , European Association for Theoretical Computer Science" volume 81, pages 226–235, 2003.

[36] J. Hillston. "A Compositional Approach to Performance Modelling". Cambridge University Press, 1996.

[37] N. Kam, I.R. Cohen, and D. Harel. "The Immune System as a Reactive System: Modeling t–cell Activation with Statecharts". Symposia on Human Centric Computing Languages and Environments (HCC'01), page 15. IEEE Computer Society, 2001.

[38] N. Kam, D. Harel, H. Kugler, R. Marelly, A. Pnueli, E.J.A. Hubbard, and M.J. Stern. "Formal Modeling of C. elegans Development: A Scenario-Based Approach", Computational Methods in Systems Biology (CMSB'03)", LNCS 2602, pages 4–20, Springer, 2003.

[39] H. Kitano. "Foundations of Systems Biology". MIT Press, 2001.

[40] H. Kitano. "Systems Biology: a Brief Overview". Science, volume 295, pages 1662–1664, 2002.

[41] K.W. Kohn. "Molecular Interaction Maps as Information Organizers and Simulation Guides". CHAOS, volume 11, number 1, pages 84–97, 2001.

[42] K.W. Kohn and Mirit I. Aladjem. "Circuit Diagrams for Biological Networks". Molecular Systems Biology, doi: 10.1038:msb4100044, 2006.

[43] K.W. Kohn, M.I. Aladjem, J.N. Weinstein and Y. Pommier. "Molecular Interaction Maps of Bioregulatory Networks: A General Rubric for Systems Biology". Molecular Biology of the Cell, volume 17, pages 1–13, 2006.

[44] H. Kitano. "A Graphical Notation for Biochemical Networks". BIOSILICO, volume 1, number 5, pages 169–176, 2003.

[45] C. Kuttler. "Simulating Bacterial Transcription and Translation in a Stochastic Pi Calculus". Transactions on Computational Systems Biology VI, LNCS 4220, pages 113–149, Springer, 2006.

[46] M. Kwiatkowska, G. Norman, and D. Parker. "Probabilistic Symbolic Model Checking with PRISM: a Hybrid Approach". Int. Journal on Software Tools for Technology Transfer, volume 6, number 2, pages 128–142, 2004.

[47] C. Laneve and F. Tarissan. "A Simple Calculus for Proteins and Cells". Workshop on Membrane Computing and Biological Inspired Process Calculi (MeCBIC'06), to appear on ENTCS.

[48] P. Lecca and C. Priami. "Cell Cycle Control in Eukaryotes: a BioSpi Model". BioConcur 2003. Available as Technical Report DIT-03-045, University of Trento, 2003.

[49] J. Leifer and R. Milner. "Deriving Bisimulation Congruences for Reactive Systems". CONCUR'00, LNCS 1877, pages 243–258, Springer, 2000.

[50] I. Marini, L. Bucchioni, P. Borella, A. Del Corso and U. Mura. "Sorbitol Dehydrogenase from Bovine Lens: Purification and Properties". Archives of Biochemistry and Biophysics, volume 370, pages 383–391, 1997.

[51] H. Matsuno, A. Doi, M. Nagasaki and S. Miyano."Hybrid Petri Net Representation of Gene Regulatory Network". Pacific Symposium on Biocomputing, World Scientific Press, pages 341–352, 2000.

[52] P. Mendes. "GEPASI: A Software Package for Modelling the Dynamics, Steady States and Control of Biochemical and Other Systems". Computer Applications in the Biosciences, volume 9, number 5, pages 563–571, 1993.

[53] M. Merro and F. Zappa Nardelli. "Behavioural Theory of Mobile Ambients". Journal of the ACM, volume 52, number 6, pages 961–1023, 2005.

[54] R. Milner. "Communication and Concurrency". Prentice–Hall, 1989.

[55] R. Milner. "Communicating and Mobile Systems: the π–Calculus". Cambridge University Press, 1999.

[56] K. Oda, Y. Matsuoka, A. Funahashi and H. Kitano. "A Comprensive Pathway Map of Epidermal Growth Factor Receptor Signaling"- Molecular Systems Biology, doi: 10.1038:msb4100014, 2005.

[57] I. Pirson, N. Fortemaison, C. Jacobs, S. Dremier, J.E. Dumont and C.Maenhaut. "The Visual Display of Regulatory Information and Networks". Trends in Cell Biology, volume 10, pages 404–408, Elsevier Science, 2000.

[58] G. Păun. "Computing with Membranes". Journal of Computer and System Sciences, volume 61, number 1, pages 108–143, 2000.

[59] G. Păun. "Membrane Computing. An Introduction". Springer, 2002.

[60] G. Păun, G. Rozenberg. "A Guide to Membrane Computing". Theoretical Computer Science, volume 287, number 1, pages 73–100, 2002.

[61] G. Plotkin. "A Structural Approach to Operational Semantics". Technical Report DAIMI FM–19, University of Aarhus, Denmark, 1981.

[62] C. Priami. "Stochastic π–Calculus". The Computer Journal, volume 38, number 7, pages 578–589, 1995.

[63] C. Priami and P. Quaglia "Beta Binders for Biological Interactions". CMSB'04, LNCS 3082, pages 20–33, Springer, 2005.

[64] C. Priami, A. Regev, W. Silvermann, and E. Shapiro. "Application of a Stochastic Name–Passing Calculus to Representation and Simulation of Molecular Processes". Information Processing Letters, volume 80, pages 25–31, 2001.

[65] P. Prusinkiewicz, A Lindenmayer. "The Algorithmic Beauty of Plants". Springer, 1990.

[66] A. Regev, E.M. Panina, W. Silverman, L. Cardelli and E. Shapiro. "BioAmbients: An Abstraction for Biological Compartments". Theoretical Computer Science, volume 325, number 1, pages 141–167, 2004.

[67] A. Regev and E. Shapiro. "Cells as Computation". Nature, volume 419, page 343, 2002.

[68] A. Regev and E. Shapiro. "The π–Calculus as an Abstraction for Biomolecular Systems". Modelling in Molecular Biology, pages 219–266, Natural Computing Series, Springer, 2004.

[69] A. Regev, W. Silverman and E.Y. Shapiro. "Representation and Simulation of Biochemical Processes Using the pi-calculus Process Algebra". Pacific Symposium on Biocomputing, World Scientific Press, pages 459–470, 2001.

[70] S. Ross. "Stochastic Processes". John–Wiley, 1983.

[71] D. Sangiorgi. "Bisimulation for Higher–Order Process Calculi". Information and Computation, volume 131, pages 141–178, 1996

[72] P. Sewell. "From Rewrite Rules to Bisimulation Congruences". Theoretical Computer Science, volume 274, pages 183–230, 2002.

[73] StochSim web site: `http://www.anat.cam.ac.uk/~compcell/StochSim.html`.

[74] The P Systems web page: `http://psystems.disco.unimib.it/`.

[75] J. van Leeuwen (editor). "Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics". Elsevier and MIT Press, 1990.

[76] D. Wilkinson. "Stochastic Modelling for Systems Biology". Chapman & Hall/CRC, 2006.

[77] P. Wong, S. Gladney, and J.D. Keasling. "Mathematical Model of the lac Operon: Inducer Exclusion, Catabolite Repression, and Diauxic Growth on Glucose and Lactose". Biotechnology Progress, volume 13, pages 132–143, 1997