

UNIVERSITÀ DEGLI STUDI DI PISA
DIPARTIMENTO DI INFORMATICA
LAUREA SPECIALISTICA IN INFORMATICA

TESI DI LAUREA SPECIALISTICA

**An Intermediate Language
for the Simulation of Biological Systems**

Giulio Caravagna

SUPERVISOR
Prof. Roberto Barbuti

SUPERVISOR
Dott. Paolo Milazzo

a.a. 2006 - 2007

Abstract

In the last few years many formalisms, originally developed by computer scientists to model systems of interacting components, have been applied to Biology. Moreover, some new formalisms have been proposed to describe biomolecular and membrane interactions. All these formalisms can describe biological systems at different levels of abstraction.

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. It must be noticed that the development of simulators for these formalisms may not be easy, in particular also the definition of a stochastic semantics for those formalisms may not be trivial.

In this thesis we propose an extension of multiset rewriting, called *String MultiSet Rewriting (SMSR)*, in which multiset elements are strings and left hand sides of rewrite rules may contain an operator, called maximal matching operator, which allows representing the multiset of all strings having a common given prefix.

SMSR can be used as an intermediate language for simulation of higher level languages; here with the term high we refer to their ability of describing biological systems at different level of abstraction. On the one end, it is easy to develop simulators for SMSR, for instance by extending the GBS simulator. On the other hand, the maximal matching operator facilitates the translation of higher level languages, in particular those based on term rewriting. The idea is that a term can be seen as a tree, a tree can be seen as a set of strings representing all paths from root to leaves, and the replacement of a subtree becomes the replacement of a set of strings having a common prefix. As an example we start giving intuitions on the encoding of P-Systems and then we show how a formalism based on term rewriting, CLS+, can be translated into SMSR, and prove translation correctness and completeness.

Higher level formalisms could be translated into SMSR directly or via their translation into CLS+. In both cases one would have the possibility of using the simulator for SMSR to simulate high level descriptions.

Acknowledgments

Voglio ringraziare in primis coloro che mi hanno seguito durante la realizzazione di questa tesi, Roberto Barbuti, Andrea Maggiolo-Schettini e Paolo Milazzo. Ovviamente senza di loro questo lavoro non sarebbe stato possibile.

Un ringraziamento particolare va alla mia famiglia che mi ha privilegiato di tante attenzioni, mi ha seguito durante questi anni e mi è sempre stata vicina.

Un ringraziamento anche a tutti i compagni di laboratorio, non vi sto a elencare perchè siete stati tanti nel corso di tutti gli studi. Siete riusciti a sopportare la confusione che combino, siete molto pazienti!

Infine, per ultimi ma non certo per importanza, ringrazio tutti i miei amici, è un piacere sapere che si può sempre contare su di voi. Un ringraziamento particolare agli spettacolari membri del Brazebo, siamo bellissimi!

Giulio 
2007

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Main Contributions	3
1.3	Related Work	4
1.4	Structure of the Thesis	5
1.5	Published Material	6
2	Background	7
2.1	Notions of Biochemistry and Cell Biology	7
2.2	Notions of Probability Theory	9
2.3	Stochastic Simulation of Chemical Reactions	11
2.4	Transition Systems	13
3	MultiSet Rewriting	15
3.1	Definition of MSR	15
3.2	Expressiveness of MSR	18
3.3	Stochastic MSR	20
4	String MultiSet Rewriting	23
4.1	Definition of SMSR	23
4.2	Expressiveness of SMSR	31
4.3	Stochastic SMSR	34
5	SMSR as an Intermediate Language	37
5.1	Encoding P-Systems	38
5.1.1	Definition of P-Systems	38
5.1.2	Encoding P-Systems	40
5.2	Encoding the Calculus of Looping Sequences	43
5.2.1	Definition of CLS+	43
5.2.2	Encoding of CLS+	46
5.2.3	Example	52
5.2.4	Correctness and Completeness of the Encoding	53
6	Conclusions	57
	Bibliography	59

Chapter 1

Introduction

1.1 Motivation

Biochemistry, often conveniently described as the study of the chemistry of life, is a multifaceted 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 [31], Hybrid Systems [1] and the π -calculus [33, 15, 43]. Moreover, some new formalisms have been proposed to describe biomolecular and membrane interactions: among these we mention the κ -calculus [18], the biochemical Stochastic π -calculus [40], the Brane Calculi [10] and CLS [5].

Others, such as P-Systems [35], have been proposed as new biologically inspired computational models and have been later applied to the description of biological systems. For some of the mentioned formalisms specific simulators exist (e.g. SPiM [45] based on the Stochastic π -calculus, CytoSim and PSym [47] based on P-Systems, and SCLSM based on CLS [44, 46]). We refer to these formalisms as high level languages; with the term high we refer to their ability of describing biological systems at different levels of abstraction. Other formalisms have been recently proposed to take into account topological properties

of biological systems, LCLS [2] and TCLS [34] based on CLS, and SpacePi [22] base on the π -calculus.

The π -calculus and new calculi based on it, namely Beta Binders [39] and BioAmbients [41] 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. An example of direct application of a model for concurrency to biochemical systems has been introduced by Regev and Shapiro in [43, 42]. Their idea is to describe metabolic pathways as π -calculus processes and in [40] they showed how the stochastic variant of the model, defined by Priami in [38], can be used to represent both qualitative and quantitative aspects of the systems described. In [22] Ewald, John and Uhrmacher present a spatial extension to the π -calculus, the SpacePi calculus.

Moreover, Regev, Panina, Silverman, Cardelli and Shapiro in [41] defined the BioAmbients calculus, a model inspired by both the π -calculus and the Mobile Ambients calculus [11], which can be used to describe biochemical systems with a notion of compartments (as, for instance, membranes). However, these calculi offer very low-level interaction primitives, and this causes models to become very large and difficult to be read.

More details of membrane interactions have been considered by Cardelli in the definition of Brane Calculi [10], which are elegant formalisms for describing intricate biological processes involving membranes. Moreover, a refinement of Brane Calculi has been introduced by Danos and Pradalier in [19].

A pioneering formalism in the description of biological systems is the κ -calculus of Danos and Laneve [18]. It is a formal language for protein interactions, it is enriched with a very intuitive visual notation and 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 [29]. Brane Calculi, the κ -calculus and the formalism proposed in [14] 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.

In the tradition of automata and formal language theory, a more recent formalism are P-Systems, proposed by Păun [35, 36, 37]. P-Systems introduce the idea of membrane computing in the subject of natural computing. They represent a new computational paradigm which allows solving NP-complete problems in polynomial time (but in exponential space). The study of P-Systems has given rise to a huge amount of scientific works, recently P-Systems have been also applied to the description of biological systems (see [47] for a complete list of references). P-Systems have a simple notation and are not specialized to the description of a particular class of biological systems, but 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.

Moreover, Barbuti, Maggiolo-Schettini, Milazzo and Troina in [5] defined CLS; it is

a term rewriting language with many variants. 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. Elements of a circular sequence can rotate. A looping sequence can represent a membrane and the containment operator allows representing that some element is inside the membrane. Behavioral equivalence expressed by using bisimilarity for CLS systems has been defined in [6]. Moreover, a stochastic semantics of CLS has been proposed in [3]. Spatial extensions of CLS have been proposed by the same authors in [2] and by Pardini in [34]. Some well-established formalisms for the description of biological systems have been translated into CLS [32].

Finally, we mention some works by Harel [26][28], in which the challenging idea is introduced of modeling a full multi-cellular animal as a reactive system. The multi-cellular animal should be, specifically, the *C. elegans* nematode worm [8], which is complex, but well defined in terms of anatomy and genetics. Moreover, Harel proposes to use the languages of Statecharts [25] and Live Sequence Charts (LSC) [16], 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 [27] and [21].

From this discussion we conclude that there is a huge number of formalisms, most of them are suitable to describe different biological phenomena at different levels of abstraction; the implementation of simulators based on those formalisms is not trivial, in particular also the definition of their stochastic semantics may not be easy. In this thesis we present, through a process of refinement of a very simple modeling language, an intermediate language for the description of biological systems. Our goal is the definition of a general purpose language able to support the encoding of the previously mentioned formalisms. We would like that adding a stochastic semantics to this language would be easy, moreover a simulator based on this formalism should be efficient and, finally, this language should be suitable to support the encoding of as many formalisms as possible.

1.2 Main Contributions

Stochastic simulation of biomolecular systems traditionally is based on Gillespie's framework [24] which describes a system as a multiset of elements representing molecules. A system transformation due to a chemical reaction among molecules is described as the replacement in the multiset of the elements representing reactants with those representing products of the reaction.

Multisets and their transformations are easily implemented and many tools exist to the purpose. Moreover, multisets and their transformations are formalized as multiset rewriting systems [30].

In the last years the need has arisen to describe biological phenomena at system level, namely by ignoring structural and behavioral details of individual system components and by taking into account organization of components in compartments and their interaction capabilities.

Multiset rewriting does not allow descriptions at this high level and, consequently, many formalisms already mentioned, sometimes adaption of existing ones, have been pro-

posed. However, the development of simulators for high level descriptions may be not easy. Moreover, also the translation from a high level formalism to multiset rewriting which allows the use of existing simulators, may pose some difficulties.

In this thesis we propose an extension of multiset rewriting, called *String MultiSet Rewriting* (SMSR), in which multiset elements are strings and left hand sides of rewrite rules can contain an operator, called maximal matching operator, which allows representing the multiset of all strings having a common given prefix.

SMSR can be used as an intermediate language for simulation. On the one end, it is easy to develop simulators for SMSR, for instance by extending the GBS simulator [23]. On the other hand, the maximal matching operator facilitates the translation of higher level languages, in particular those based on term rewriting. The idea is that a term can be seen as a tree, a tree can be seen as a multiset of strings representing all the paths from root to leaves and the replacement of a subtree becomes the replacement of a set of strings having a common prefix. As an example we start giving intuitions on the encoding of P-Systems [35] and then we show how a formalism based on term rewriting, CLS+ [32], can be translated into SMSR, proving translation correctness and completeness.

Higher level formalisms, such as those mentioned before, could be translated into SMSR directly or via their translation into CLS+ along the lines described in [32, 5]. In both cases one would have the possibility of using the simulator for SMSR to simulate high level descriptions.

1.3 Related Work

We start mentioning a work by Versari which proposes in [48] the $\pi@$ calculus, an extension of the π -calculus with priorities and polyadic synchronization that turns out to be suitable to act as a core platform for the comparison of other calculi.

Polyadic synchronization is used to model localization of communication typical of the majority of bio-inspired calculi, which usually formalize it by the explicit introduction of compartments. Priority is exploited as a powerful mechanism for achieving atomicity, that is the completion, without overlapping, of complex atomic operations by the execution of several simple steps.

In the same work and in [49] are presented reasonable encodings of BioAmbients, Brane Calculi and catalytic P-Systems. The term reasonable is used by Versari to express the fact that the encoding functions have got to fulfill the notion of operational correspondence, characterized by two complementary properties: completeness and soundness. Furthermore, it has to preserve the degree of distribution of the source language, for instance the homomorphisms with respect to the parallel composition of terms must be preserved, and should not depend on the channel or on the compartment names of the term to be encoded.

As regards multiset rewriting systems, they have been already studied as formal modeling of biological systems in [30]. In particular, Cervesato in [12] presents, within the framework of linear logic, two multiset rewriting based formalisms: Propositional MultiSet Rewriting and First Order MultiSet Rewriting. The former is based on multisets of atomic objects which are modified by the application of multiset rewriting rules. This language strongly influenced the MSR formalism we will introduce as a formal modeling language for biochemical systems; the main difference between these languages are syn-

tactic. In particular, the syntax of MSR is more similar to that of process algebras. The latter, namely First Order MultiSet Rewriting, is based on multisets of objects that can carry structured values and are rewritten by parametric rewrite rules. First Order MultiSet Rewriting has been extended with the possibility of generating fresh data in [13] by Cervesato, Durgin, Lincoln, Mitchell and Scedrov.

Initially, we examined the possibility of using First Order MultiSet Rewriting as an intermediate language for the simulation of biological systems. However, we proposed a new formalism, namely the SMSR formalism, in which the structure of the objects is given by a string. We believe that strings are suitable to describe biological objects that can be modeled by using higher level languages and, furthermore, there exist many efficient algorithms for manipulating string that we could use to implement an efficient simulator based on SMSR. The syntax of SMSR, as it is based on MSR, is similar to that of process algebras. Furthermore, SMSR with respect to First Order MultiSet Rewriting, is enriched with a new operator, namely the maximal matching operator. With the use of maximal matching operator, which has got two different forms of application, is possible to represent multisets of objects with a given prefix. The use of the maximal matching operator together with a well-known encoding technique for term-based languages, facilitates the encoding of the previously mentioned formalisms.

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 and 4 we introduce two formalisms for the description and the simulation of biological systems, in Chapter 5, we show the encoding of two higher level formalism into the intermediate language we will define in this thesis.

- In Chapter 3 we present a very simple language, namely the MultiSet Rewriting formalism (MSR), for the simulation of mainly biochemical systems. This language is based on multisets of atomic objects that are modified by the application of rewriting rules; rewriting rules simply rewrite multisets. We examine its expressiveness with respect to Turing Machines and we give a proof of non Turing completeness for the formalism. Finally, we examine the possibility of adding a stochastic semantics to the formalism.
- In Chapter 4 we propose the String MultiSet Rewriting formalism (SMSR) as an intermediate language: SMSR is a natural extension of MSR. The choice of MSR as a basis for SMSR has two main motivations: the simple and clear syntax of MSR together with the possibility of easily developing a simulator for this formalism. SMSR is based on multisets of strings that are modified, as in MSR, by the application of rewriting rules. As regard the implementation of SMSR, there exist many efficient algorithms for manipulating string that we could use to implement an efficient simulator based on SMSR. The main features of SMSR are three: the possibility of defining rules with variables, the possibility of generating fresh data in the process

of application of the rules and the maximal matching operator. In particular this operator can be very useful when higher level languages are encoded, accordingly to a well-known encoding technique, into SMSR. At the end of the chapter we examine the expressiveness of SMSR with respect to Turing Machines and we give a proof of Turing completeness for the formalism. Finally, we give intuitions on the possibility of adding a stochastic semantics to the formalism.

- In Chapter 5 we discuss a general encoding technique to encode term-based languages into SMSR, in particular we give intuitions on the encoding of P-Systems, an established formalism for the description of biological systems based on membranes and contextual rules. We also encode the Calculus of Looping Sequences in one of its variants, namely the CLS+ formalism, into SMSR using such a technique. We formally define the required encoding functions for terms, patterns and rules of CLS+ and give a proof of correctness and completeness of the defined encoding.

Finally, in Chapter 6 we give some conclusions and discuss further work.

1.5 Published Material

The main ideas presented in this thesis, namely the definition of SMSR in Section 4.1 and the encoding of CLS+ into SMSR in Section 5.2.2, have been presented at a conference and can be found in [7]. 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. These 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:



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-versa.

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, \dots it holds $P(I_1 \cup I_2 \cup \dots) = \sum P(I_i)$

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 $x \in \mathbb{R}$.

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

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

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 $t_0 < t_1 < \dots < t_k$ and states s_0, \dots, s_k it holds

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

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 $\langle S, R, \pi \rangle$, where*

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

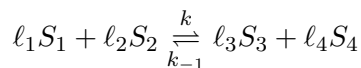
The system is assumed to pass from a configuration modeled by a state s to another one modeled by a state s' by consuming an exponentially distributed quantity of time, in which the parameter of the exponential distribution is $R(s, s')$. The summation $\sum_{s' \in S} R(s, s')$ is called the *exit rate* of state s . A transition from s to s' happens with probability

$$P(s, s') = \frac{R(s, s')}{E(s)}$$

Finally, the system is assumed to start from a configuration modeled by a state $s \in S$ with probability $\pi(s)$, and $\sum_{s \in S} \pi(s) = 1$. If the set of states of the CTMC is finite ($S = \{s_1, \dots, s_n\}$), 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 $R(s_i, s_j)$.

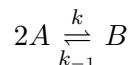
2.3 Stochastic Simulation of Chemical Reactions

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:



where S_1, \dots, S_4 are molecules, ℓ_1, \dots, ℓ_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



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:

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

where S_1, \dots, S_ρ are all the distinct molecular reactants of the reaction.

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

In [24] 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



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



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, \dots, S_n are the only molecules that may appear in a chemical solution, a *state* of the simulation is a tuple (X_1, \dots, X_n) representing a solution containing X_i molecules S_i for each i in $1, \dots, n$. Given a state (X_1, \dots, X_n) , a set of reactions R_1, \dots, 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:

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

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 . Notice that there exist also variants to the Gillespie's algorithm and it is worth mentioning [4].

2.4 Transition Systems

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 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, \dots , 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 $s_1, \dots, s_n \in S$ such that $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n \rightarrow s$.

An 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, \dots , L is a set of labels ranged over by l, l_0, l_1, \dots 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 an *LTS* the label of a transition usually denotes the event that has caused the transition. *LTS*s 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. There exist many notion of behavioral equivalence, it is worth mentioning the bisimulation equivalence in both its strong and weak definition; however, we omit to define these equivalences in this thesis.

Chapter 3

MultiSet Rewriting

3.1 Definition of MSR

In this section we introduce *MultiSet Rewriting* (MSR); already exist some formal definitions of this formalism, for instance the Propositional MultiSet Rewriting in [12], although we will introduce a new one with a syntax similar to process algebras. We will define the syntax of terms, a structural congruence relation over terms, rewriting rules and an operational semantics. At the end of this chapter the expressiveness with respect to Turing Machines will be investigated, and, in the last section, we will discuss the possibility of adding a stochastic behavior to MSR. Some examples will be shown in this chapter to make the exposition clearer.

A state of the system will be represented by a term of MSR; in order to define terms we assume a infinite set of atomic objects $\mathcal{A} = \{a, b, c, \dots\}$ and we define multisets of atomic objects in \mathcal{A} with the following *syntax*:

Definition 3.1. (Terms)

Multisets M over a infinite set of atomic objects \mathcal{A} are defined by the following grammar:

$$\begin{aligned} M &::= A \mid M \mid M \mid \epsilon \\ A &::= a \mid b \mid c \mid \dots \end{aligned}$$

where \mid is the usual union on multisets and ϵ is the empty multiset. The universe of all multisets is denoted by \mathcal{M} .

Multisets of atomic objects can be constructed by means of the union operator \mid . We use the notation \mid for multiset union instead of the more usual notation \cup to have a notation similar to that of process calculi.

Well formed multisets can be used to represent aggregation of elements with repetitions. As an example, the aggregation of three objects a and two objects b is denoted by the multiset $a \mid a \mid a \mid b \mid b$. Of course the order of appearance of the objects inside the multiset is meaningless, in particular any permutation of the ordering of objects represents the same multiset, for instance $a \mid b$ is the same as $b \mid a$ for any a or b . There are also other properties of operators defined in the syntax of MSR; all these properties are endowed in a *structural congruence relation* on terms of MSR stating which syntactically different terms represent the same multiset.

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

$$M_1|M_2 \equiv M_2|M_1 \quad M_1|(M_2|M_3) \equiv (M_1|M_2)|M_3 \quad M|\epsilon \equiv M$$

Axioms of the structural congruence simply state the associativity and commutativity of $|$ and the neutral role of ϵ . With this relation is now clear that the commutativity of $|$ provides the property that we previously stated; in particular, referring to the previous example we have that

$$a | a | a | b | b \equiv b | a | b | a | a \equiv \epsilon | a | a | a | b | b \equiv \dots$$

Notice that, due the neutral role of ϵ with respect to $|$, given a multiset M there exist infinite multisets structurally congruent to M .

We can formally define the algebraic multiset operations \in , \subseteq , \setminus , \cup and \cap for MSR terms as follows

- given $a \in \mathcal{A}$ and $M \in \mathcal{M}$ we have that $a \in M \Leftrightarrow \exists M' \in \mathcal{M}. M \equiv a | M'$;
- given $M, M' \in \mathcal{M}$ we have that $M \subseteq M' \Leftrightarrow \forall a \in M. a \in M'$;
- given $M', M'' \in \mathcal{M}$ we have that $M' \setminus M'' = M \Leftrightarrow M \equiv M' | M''$;
- given $M, M' \in \mathcal{M}$ we have that $M \cup M' = M | M'$;
- given $M', M'' \in \mathcal{M}$ we have that $M' \cap M'' = M \Leftrightarrow M \equiv a_1 | \dots | a_n$ and $\forall i = 1, \dots, n. a_i \in M' \wedge a_i \in M''$.

Example 3.3. A simple chemical solution containing one molecule of water (H_2O), one molecule of hydrogen two (H_2) and one molecule of oxygen (O) can be represented using the multiset

$$H_2O | H_2 | O$$

built over the alphabet $\{H_2O, H_2, O\}$.

We can now define *rewriting rules* for MSR: intuitively each rule will consist of two multisets, the former will be referred to as the multiset of reactants, and the latter as the multiset of products. Formally we define rewrite rules as follows:

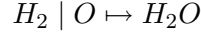
Definition 3.4. (Rewrite Rules) *A rewrite rule is a pair (M, M') , denoted by $M \mapsto M'$, where $M, M' \in \mathcal{M}$, $M \neq \epsilon$. With \mathcal{R} we denote the set of all possible rewriting rules.*

Intuitively the rule $a | b \mapsto c$ states that whenever it would be applied to a multiset the reactants a and b will be replaced with the product c . If in the multiset to which the rule is applied there is more than one a or b object, then the objects to be replaced will be chosen non-deterministically.

Formally, as regards the application of a rule $M \mapsto M'$ to a multiset N , we have that:

- the rule can be applied if and only if all the reactants in M appear in N , namely $M \subseteq N$;
- whenever the rule can be applied to $N \equiv M|N'$ then the result of its application is the multiset $M'|N'$ obtained by computing the multiset $(N \setminus M) \cup M'$.

Example 3.5. A simple chemical reaction stating that one molecule of hydrogen two and one molecule of oxygen reacting together are transformed into a molecule of water is described by the MSR rule



The result of applying this rule to the multiset given in Example 3.3, namely $H_2O \mid H_2 \mid O$, is



Notice that the same rule cannot be applied any more to the new multiset because $H_2 \mid O \not\subseteq H_2O \mid H_2O$.

After these considerations on rules and multisets we can define an *operational semantics* for MSR providing the required features.

Definition 3.6. (Semantics) *Given a finite set of rewriting rules $R \subset \mathcal{R}$, the semantics of MSR is the least relation \rightarrow closed with respect to \equiv and satisfying the following inference rule:*

$$\frac{M \mapsto M' \in R}{M'' \mid M \rightarrow M'' \mid M'}$$

Notice that the closure of \rightarrow with respect to \equiv implicitly yields the following inference rule

$$\frac{T \equiv M \rightarrow M' \equiv T'}{T \rightarrow T'}$$

We show now a simple example of derivation of the semantics.

Example 3.7. Given a multiset M and a single rule R such that

$$M \equiv a \mid b \mid a \mid c \quad R : b \mid a \mapsto c$$

we show one application of the semantics of MSR to M :

$$\frac{b \mid a \mapsto c}{\underbrace{b \mid a}_{\text{removed reactants}} \mid a \mid c \rightarrow \underbrace{c}_{\text{inserted products}} \mid a \mid c}$$

Notice that there are two ways of applying the rule to M because it contains two occurrences of a , however, independently from the chosen occurrence, we get always the same resulting multiset modulo the structural congruence relation \equiv .

Having defined the syntax and the semantics of MSR, we can now define an *MSR system* as follows:

Definition 3.8. (MultiSet Rewriting System)

Given a multiset $M \in \mathcal{M}$ and a finite set of rules $R \subset \mathcal{R}$ we define an MSR system as the pair (M, R) . The time-evolution (an MSR computation) of such a system is described by the closure of the transition relation \rightarrow when applied to M and R .

An MSR model of a biological system is represented using an MSR system having, as initial state, a multiset representing the initial solution, and, as rewriting rules, the *events* that may happen in the biological system.

3.2 Expressiveness of MSR

In this section we examine the possibility of using multiset rewriting as an intermediate language for their simulation.

As may it be intuitive MSR is suitable for the simulation of biological systems based on biochemistry or, more in general, those describing cell biology. There exist systems that cannot be simulated by MSR computations, in particular we have that high-level formalisms mentioned in Chapter 1 can describe a wider class of biological systems. The greater expressiveness of the mentioned formalisms, from a computer science point of view, can be seen as the fact that those formalisms are as expressive as Turing Machines [32]. The next theorem states the non Turing completeness of MSR. As we cannot expect to be able to use in the simulations a language which is less expressive than the one we want to simulate, we will have to define a more expressive formalism.

Theorem 3.9. *MultiSet Rewriting is not Turing complete.*

Proof. The main idea is the encoding of each MSR computation in a computation of a Place/Transition Petri Nets. As Petri Nets has been proved to be non Turing complete, then also MSR will be proved to be non Turing complete.

A Petri Net P is a 5-tuple $P = (S, T, F, M_0, W)$, where

- S is a set of places
- T is a set of transitions
- $F \subseteq (S \times T) \cup (T \times S)$ is a set of arcs
- $M_0 : S \mapsto \mathbb{N}$ is the initial marking for P
- $W : F \mapsto \mathbb{N}^+$ is a set of arch weights

We show how to encode a generic MSR system $(M, \{R_1, \dots, R_k\})$ into a Petri Net P ; in particular starting from the MSR system we define the structure of the net, namely S and T , we then encode the initial state M into the marking M_0 , and, finally, we build the set F and the weights W from rules in $\{R_1, \dots, R_k\}$.

Net structure: Let $A \subseteq \mathcal{A}$ denote the finite set of all symbols of the alphabet \mathcal{A} that appear in the whole MSR system we want to encode, that is all the atomic objects that appear in both M and $\{R_1, \dots, R_k\}$. We can define the set of places in the net P as follows: for each $a \in A$ we define a place $\bigcirc_a \in S$. We also enrich S with a place representing the ϵ symbol that is not included in \mathcal{A} , thus $\bigcirc_\epsilon \in S$. As regards rules, starting from rules $\{R_1, \dots, R_k\}$ we define the set of transitions $T = \{\square_1, \dots, \square_k\}$.

Initial marking: Let N_a denote, for any symbol $a \in A$, the number of occurrences of a that appear in multiset M (in other words the algebraic cardinality of a in M); then we can define the marking M_0 as $M_0(\bigcirc_a) = N_a$. For all the other symbols $\bar{a} \in A$ that appear in the rules but not in M , we define $M_0(\bigcirc_{\bar{a}}) = 0$.

Arcs: for each rule R_w in the MSR system of the form $a_1 \mid \dots \mid a_m \mapsto b_1 \mid \dots \mid b_n$ where $a_i, b_j \in A$, we add to F the arcs $(\bigcirc_{a_i}, \square_w)$ for any $i = 1, \dots, m \wedge a_i \neq \epsilon$ and the arcs $(\square_w, \bigcirc_{b_j})$ for any $j = 1, \dots, n$. Let $M_{a,w}$ denote, for any object in A , the number

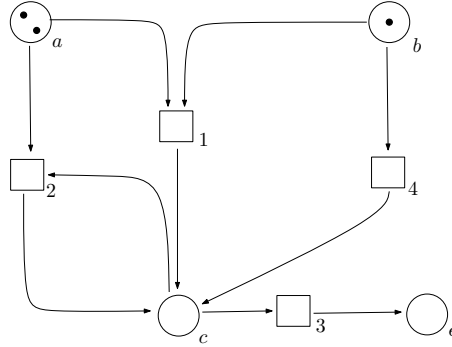


Figure 3.1: Example of encoding a MSR system into a Petri Net

of occurrences of a that appear into the multiset of reactants of rule R_w , and $M'_{a,w}$ the number of occurrences of a that appear in the multiset of products of rule R_w . Then we can define W as $W(a, \square_w) = M_{a,w}$ and as $W(\square_w, a) = M'_{a,w}$.

The MSR system is now encoded into the Petri Net and the computation on the net is able to *emulate* the behavior of the system. \square

Example 3.10. Let M be the multiset $a \mid a \mid b$; given the rules $\{R_1, R_2, R_3, R_4\}$ such that

$$\begin{aligned} R_1 : a \mid b &\mapsto c & R_2 : c \mid a &\mapsto c \\ R_3 : c &\mapsto \epsilon & R_4 : b &\mapsto c \end{aligned}$$

with the MSR system $(M, \{R_1, R_2, R_3, R_4\})$ we have that we can associate the following Petri Net $P = (S, T, F, M_0, W)$ where

- $\{\bigcirc_a, \bigcirc_b, \bigcirc_c, \bigcirc_\epsilon\}$ is the set of places S
- $\{\square_1, \square_2, \square_3, \square_4\}$ is the set of transitions T
- the set of arcs given, for instance, by rule R_1 is $\{((\bigcirc_a, \square_1), 1), ((\bigcirc_b, \square_1), 1), ((\square_1, \bigcirc_c), 1)\}$
- $M_0(\bigcirc_a) = 2$, $M_0(\bigcirc_b) = 1$ and $M_0(\bigcirc_c) = M_0(\bigcirc_\epsilon) = 0$ is the initial marking for P
- the set of arch weights given, for instance, by rule R_1 is $\{((a, \square_1), 1), ((b, \square_1), 1), ((\square_1, c), 1)\}$.

A graphical representation of the Petri Net P is show in Figure 3.1.

Summarizing, we introduced MSR as a formal language for the description of some biological systems, we defined formally its semantics and proved an important result on its expressiveness.

We proved the non Turing completeness of MSR; we will use this language as a basis for the definition of an intermediate language for the description of biological systems. The language that we define will have to be proved to be Turing complete.

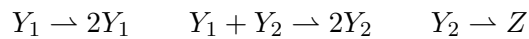
We want to notice that, even if not suitable as an intermediate language, the MSR formalism can be used in practice in the development of simulation softwares for biological systems; as example MSR as been used as the formal definition of the semantics

of the Generic Biological Simulator [23, 9]. Such a simulator can describe a wide class of biological systems, in particular in [9] has been used to simulate biochemical systems and evolutionistic systems based on sexual selection (a phenomenon known as sympatric speciation).

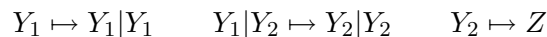
We conclude this section with a simple example of a biochemical system, the Lotka–Volterra system, which can be easily modeled using a MSR system.

Example 3.11. The Lotka–Volterra equations, also known as the *predator-prey* equations, are a pair of first order, non-linear, differential equations frequently used to describe the dynamics of biological systems in which two species interact, one a predator and one its prey. They were proposed independently by Alfred J. Lotka and Vito Volterra in the early 1900’s.

The differential equations can be seen as the following chemical reactions



whose can be easily encoded into the following multiset rewriting system by using alphabet $\{Y_1, Y_2, Z\}$



The initial state for such a simulation, defined by a generic aggregation where $Y_1 = n_1$, $Y_2 = n_2$ and $Z = n_3$, can be encoded into the following multiset rewriting term

$$Y_1 \times n_1 \mid Y_2 \times n_2 \mid Z \times n_3$$

where with $T \times n$ we denote the repetition of T for n times.

3.3 Stochastic MSR

In this section we examine the possibility of adding a stochastic semantics to MSR.

We recall that adding a stochastic behavior to a modeling language is necessary in order to use a simulation algorithm to describe the time–evolution of a modeled biological system. We examined in Section 2.3 the well established Gillespie’s algorithm [24] and in this section we show how to enrich MSR with a stochastic semantics inspired by such an algorithm. From the stochastic semantics a *Continuous Time Markov Chain* (CTMC for short) can be easily derived. Given a CTMC associated to an MSR system we can use model checking techniques to verify properties on the model and, furthermore, we can use analytical tools to compute properties on the CTMC, for instance the *steady states* of the system.

For most of the higher level languages there exist a stochastic semantics but, as we previously said, adding such a semantic may not be trivial. In particular, we note that in the process of computing the rate of a rule, the most frequent problem that must be solved is represented by computing the number of occurrences of the reactants of the rule that appear in the current state of the simulation. As higher level languages are term–based formalisms, then both the state of the computation and the reactants of the rules are expressed as terms or patterns of the language. In this case the problem of counting the number of occurrences of the reactants becomes the problem of searching the number of occurrences of a sub–term inside a term. Here the sub–term are the reactants of the rule

and the term is the state of the computation. Alternatively, this problem can be seen as the problem of searching a sub-tree inside a tree. In particular, the sub-term and the term correspond to the sub-tree and to the tree, respectively.

Furthermore, as almost all the higher level languages are equipped with a structural congruence relation, then the sub-trees we are going to search may change accordingly to the equivalences introduced by such a relation. This last consideration may introduce difficulties for both the definition and the implementation of the stochastic semantics of the language.

Surely, the outlined problem can be solved using naive algorithms, although we can note that, as it has to be solved at each step of a computation and for each rule of the system, then would be very important to solve it efficiently.

We now focus on the problem in the context of multiset rewriting; in particular, we will show that solving such a problem referring to multisets rewriting formalisms is quite trivial.

In order to formally define the problem let us define a multiset M built over an alphabet \mathcal{A} as a *cardinality function*

$$m : \mathcal{A} \rightarrow \mathbb{N}$$

For each object $a \in \mathcal{A}$ the value $m(a)$ represents the number of occurrences of a in M . With this representation of multisets, a rule with reactants represented by the multiset m can be applied to a multiset n if and only if $\forall a \in \mathcal{A}. m(a) \leq n(a)$.

Example 3.12. Given the multiset

$$M \equiv a|a|a|b|b$$

built over $\mathcal{A} = \{a, b, c\}$, M can be represented as the set of pairs

$$m = \{(a, 3), (b, 2), (c, 0)\}$$

because the distinct objects of \mathcal{A} that appear in M are a and b with cardinality 3 and 2, respectively. The cardinality function then states that $m(a) = 3$, $m(b) = 2$ and $m(c) = 0$.

We can note that, the problem of computing the number of occurrences of the reactants inside the multiset representing the current state of a simulation, is immediately solved computing the values of the cardinality functions.

Formally, with respect to the Gillespie's algorithm we have that, given an MSR computation in which the current state is represented by the multiset

$$N = n$$

then the rate of an MSR rule $R : (M, M', k)$ enriched with a kinetic constant $k \in \mathbb{R}$, is given by

$$Rate(R, N) = k \prod_{a \in \mathcal{A}} \binom{n(a)}{m(a)}$$

when $M = m$.

Example 3.13. Given the multiset

$$n = \{(a, 3), (b, 2), (c, 0)\}$$

the rate of a rule $R : (M, M', k)$ where

$$m = \{(a, 1), (b, 2), (c, 0)\}$$

is given by

$$Rate(R, N) = k \prod_{v \in \{a, b\}} \binom{n(v)}{m(v)} = k \binom{n(a)}{m(a)} \binom{n(b)}{m(b)} = k \binom{3}{1} \binom{2}{2} = k \frac{3!}{1!2!} = 3k$$

We can now define the stochastic semantics of MSR with the following inference rule.

$$\frac{R : (M, M', k) \quad M \subseteq M_1 \quad M_2 = (M_1 \setminus M) \cup M'}{M_1 \xrightarrow{R, Rate(R, M_1)} M_2}$$

Here the labels on the transition represent the identifier of the applied rule, namely R , and the rate of the transition, namely the result of computing the value $Rate(R, M_1)$.

The CTMC associated to the transition system generated by this semantics can be obtained trivially. Formally, given a state s of the transition system, let $T = \{(R_i, r_i)\}$ be the set of labels of the transitions of the semantics that lead to the same state s' , namely the set of transitions $\xrightarrow{R_i, r_i}$ with $r_i = Rate(R_i, s)$, then we can define in the CTMC a transition from s to s' with following rate.

$$R(s, s') = k \sum_{(R_i, r_i) \in T} r_i$$

Summarizing, we examined the problem of adding a stochastic semantics to multiset rewriting formalisms and we discovered that there exist a representation of multisets which is suitable to trivially solve the problem. Then we gave a stochastic semantics with respect to the Gillespie's algorithm and we showed how to derive a CTMC from such a semantics. As already said, we will use the main ideas of MSR to formally define an intermediate language for the simulation of biological systems.

Chapter 4

String MultiSet Rewriting

4.1 Definition of SMSR

In this section we introduce the *String MultiSet Rewriting* (SMSR) formalism as a natural extension of the MSR formalism introduced in Chapter 3. We will define the syntax of terms and a structural congruence relation on them; then we will introduce rewriting rules and we will define an operational semantics describing the evolution of terms by means of rewriting rules application.

SMSR, accordingly to the core of the MSR formalism, will be a multiset rewriting formalism based on multisets of objects that will be rewritten by the application of rewriting rules. As regards the objects that we will store into the multisets we have an important difference with respect to MSR; while in the case of MSR we used atomic objects, in the case of SMSR objects will be strings built over an alphabet. The use of strings instead of atomic objects provides an interesting feature; a string can represent a biological object with a certain structure which can be encoded into the string itself, furthermore any change on a string can be seen as a change in the structure of the object represented by the string itself.

The rewriting rules of SMSR will be, as in the case of MSR, pairs of multisets; rules will be enriched with variables and with a new syntactic operator, the maximal matching operator. Furthermore, inspired by the possibility of generating fresh names as in some process algebras [15, 43] and in First Order Multiset Rewriting [12], we will introduce in SMSR the possibility of generating fresh data in the process of application of a rule containing a certain kind of variables. In this section we will introduce, using a lot of examples to make the exposition clearer, the formal definition of SMSR which will provide the outlined features.

Finally, as regards the expressiveness of SMSR and the considerations made in Section 3.2 about the expressiveness of MSR, we will prove in the next section the Turing completeness of this new formalism.

In order to define strings we start with assuming an infinite alphabet $\mathcal{E} = \{e_1, e_2, \dots\}$. The objects we will store in the multisets will be strings built over the alphabet \mathcal{E} .

We can now define the *syntax* of SMSR terms representing the state of an SMSR computation as follows:

Definition 4.1. (*Terms*) Multisets M and strings S over an alphabet \mathcal{E} are defined by

the following grammar:

$$\begin{aligned} M &::= S \mid M \mid M \\ S &::= \epsilon \mid e \mid S \cdot S \end{aligned}$$

where ϵ is the empty string, e is a generic element of \mathcal{E} , \cdot is string concatenation, and \mid is multiset union. We denote the set of all multisets as \mathcal{M} and the set of all strings as \mathcal{S} .

Strings over \mathcal{E} can be constructed by means of the concatenation operator \cdot , with ϵ representing the concatenation of zero elements. Multisets of strings can be constructed by means of the union operator \mid ; notice that this operator is the same used in MSR.

As we made for MSR, we endow SMSR with a *structural congruence relation* on terms; in particular we express the associativity of \cdot and \mid , the commutativity of the latter, and the neutral role of ϵ .

Definition 4.2. (Structural Congruence) *The structural congruence relation \equiv is the least congruence relation on multisets satisfying following axioms:*

$$\begin{aligned} S_1 \cdot (S_2 \cdot S_3) &\equiv (S_1 \cdot S_2) \cdot S_3 & S \cdot \epsilon &\equiv S \\ M_1 \mid M_2 &\equiv M_2 \mid M_1 & M_1 \mid (M_2 \mid M_3) &\equiv (M_1 \mid M_2) \mid M_3 & M \mid \epsilon &\equiv M \end{aligned}$$

Note that the definition of the canonical algebraic multisets operations \in , \subseteq , \setminus , \cup and \cap on MSR terms given in Section 3.1 can be trivially extended to the domain of SMSR terms.

Example 4.3. Given an alphabet such that

$$\mathcal{E} = \{a, b, c\}$$

there exist infinite different multisets that can built over \mathcal{E} . For instance some of them may be

$$M \equiv a \cdot b \cdot c \mid a \cdot b \mid a \mid c \quad M' \equiv a \cdot c \mid b \cdot b \mid a \cdot a \cdot c \quad M'' \equiv a \mid a \mid c$$

In this simple example M'' is both a term of SMSR and of MSR while, M and M' are only SMSR terms; in general given an alphabet \mathcal{E} every MSR term built over \mathcal{E} is also an SMSR term, but not viceversa.

Now we are going to introduce SMSR *patterns*. Intuitively, patterns are terms enriched with variables and with a maximal matching operator, this operator is the main novelty of SMSR. Patterns will be used to define rewriting rules, and both variables and the maximal matching operator will be instantiated in the process of application of a rule in order to match a subterm of the term to which the rule is applied.

We assume an infinite set of variables \mathcal{V} such that

$$\mathcal{V} = \mathcal{V}_{\mathcal{E}} \cup \mathcal{V}_{\mathcal{S}} \cup \mathcal{V}_{\mathcal{M}}$$

where:

- $\mathcal{V}_{\mathcal{E}} = \{x, y, z, \dots\}$ denotes the infinite set of *element* variables; in the process of instantiating such a variable the instantiation value will be of type element, namely will be an element of the set $\mathcal{E} \cup \{\epsilon\}$.

- $\mathcal{V}_S = \{\tilde{x}, \tilde{y}, \tilde{z}, \dots\}$ denotes the infinite set of *sequence* variables; in the process of instantiating such a variable the instantiation value will be of type string, namely will be a string built over the set $\mathcal{E} \cup \{\epsilon\}$.
- $\mathcal{V}_M = \{X, Y, Z, \dots\}$ denotes the infinite set of *multiset* variables; these variables will be used together with the maximal matching operator. In the following we will formally explain their meaning.

We can now introduce SMSR patterns over the alphabet \mathcal{E} and the infinite set of variables \mathcal{V} as follows:

Definition 4.4. (*Patterns*) Multiset patterns MP and string patterns SP over an alphabet \mathcal{E} and an infinite set of variables \mathcal{V} are defined by the following grammar:

$$\begin{aligned} MP & ::= SP \mid MP \mid MP \mid \{SP\}_X \mid \{SP\}_{SP} \\ SP & ::= \epsilon \mid e \mid SP \cdot SP \mid \tilde{x} \mid x \end{aligned}$$

where ϵ is the empty string, e is a generic element of \mathcal{E} , \tilde{x} , x and X are generic string, element and multiset variables, \cdot is string concatenation, \mid is multiset union and $\{_ \}__$ is the maximal matching operator. We denote the set of all multiset patterns as \mathcal{MP} and the set of all string patterns as \mathcal{SP} .

We assume the algebraic multiset operations \in , \subseteq , \setminus , \cup and \cap to be trivially extended to SMSR patterns. Patterns are used to define rewrite rules of SMSR. Analogously as in MSR a rewrite rule is essentially a pair of multisets; in particular in the case of MSR a rule was a pair of terms while in the case of SMSR a rule is a pair of multiset patterns. As always the first describes the term that is modified by an application of the rule and the second describes how the term changes after the application.

Definition 4.5. (*Rewrite Rule*) A rewrite rule R is a pair (M, M') , denoted by $M \mapsto M'$, where $M, M' \in \mathcal{MP}$ and $M \neq \epsilon$. With \mathcal{R} we denote the set of all possible rewrite rules.

Before defining the semantics of SMSR we make some considerations on the syntax of multiset patterns; in particular we focus on the meaning of variables and on the meaning of the maximal matching operator.

Intuitively, variables in patterns allow a rewrite rule to be applicable to any term that can be obtained by replacing such variables with proper elements or strings.

Example 4.6. Given the following rules

$$R_1 : a \cdot x \mapsto \dots \quad R_2 : a \cdot \tilde{x} \mapsto \dots$$

where $x \in \mathcal{V}_E$ and $\tilde{x} \in \mathcal{V}_S$, we can point out that with respect to the following multisets

$$M \equiv a \cdot b \quad M' \equiv a \cdot a \cdot c$$

the following propositions hold:

- R_1 can be applied to M when its element variable x is instantiated with value b

- R_2 can be applied to M when its sequence variable \tilde{x} is instantiated with value b (an element is a string of length one)
- R_2 can be applied to M' when its sequence variable \tilde{x} is instantiated with value $a \cdot c$
- R_1 cannot be applied to M' because its variable x cannot be instantiated to a string. In general a rule of the form $a \cdot b \mapsto c$ cannot be applied to the multiset $a \cdot b \cdot b$.

As regards the maximal matching operator we can notice that there exist two different forms for its use, namely

$$\{\{SP\}\}_X \qquad \{\{SP\}\}_{SP}$$

where $SP \in \mathcal{SP}$ and $X \in \mathcal{V}_M$; intuitively both the forms represent a multiset of strings which have as prefix the same instantiation of the string pattern SP .

In particular, as regards the first form, $\{\{SP\}\}_X$, we have that, whenever we will apply to a multiset M a rule containing such an operator, then this instance of the maximal matching operator will represent the multiset of *all* the strings in M which have as prefix the instantiation of the string pattern SP . This behavior justifies the name for the operator, in particular the word *maximal* means that the operator represents the multiset of *all* the strings with a given prefix; formally such an operator represents the multiset of all the strings S' of M satisfying the following proposition

$$S' \in M \wedge \exists S'' \in \mathcal{S}. S' \equiv S \cdot S''$$

where S is the instantiation for the prefix SP . Semantics of SMSR will guarantee such a behavior for this form of the maximal matching operator.

Example 4.7. Given the following multiset

$$M \equiv a \cdot b \mid a \cdot b \cdot a \mid a \cdot c \cdot b \mid a \cdot c \cdot c$$

and these two instances of the maximal matching operator

$$\{\{a \cdot b\}\}_X \qquad \{\{a \cdot x\}\}_X$$

we have that, with respect to M , $\{\{a \cdot b\}\}_X$ represents the multiset of all the strings with prefix $a \cdot b$, namely $a \cdot b \mid a \cdot b \cdot a$. Differently, $\{\{a \cdot x\}\}_X$ represents both the multiset $a \cdot b \mid a \cdot b \cdot a$ and the multiset $a \cdot c \cdot b \mid a \cdot c \cdot c$ dependently from the instantiation of its variable x ; in particular will represent the former when x is instantiated with the value b , and the latter when x is instantiated with the value c .

As regards the second form of the maximal matching operator, namely $\{\{SP_1\}\}_{SP_2}$, there is a syntactic difference on the subscript with respect to the previously defined form; in particular in the first one we had a multiset variable while in this one we have a sequence pattern. With this form of the operator we want to express a proposition with an existential quantifier, namely

$$\exists! S_1 \cdot S_2 \in M$$

where S_1 and S_2 are the instantiations of the patterns SP_1 and SP_2 , respectively. In other words this form of the maximal matching operator will represent the *singleton* of the *only*

string $S_1 \cdot S_2$ in M ; note that if in M there will be 0 or $n > 1$ strings $S_1 \cdot S_2$ then the rule containing one instance of this form of the operator cannot be applied. In the case there would be k syntactically equivalent occurrences of this form of the operator, then the rule would be applied if and only if there exist exactly k strings $S_1 \cdot S_2$ in M . As already mentioned, the semantics of SMSR will provide the expected behavior for this operator in both its forms.

Example 4.8. Given the following multiset

$$M \equiv a \cdot b \cdot a \mid a \cdot c \cdot c \mid a \cdot c \cdot c$$

and these two instances of the maximal matching operator

$$\{a \cdot b\}_a \qquad \{a \cdot c\}_c$$

we have that $\{a \cdot b\}_a$ will represent the singleton $a \cdot b \cdot a$ of M while $\{a \cdot c\}_c$ will represent the singleton $a \cdot c \cdot c$ of M . Notice that, as we said, if we had a rule containing the pattern $\{a \cdot b\}_a$ then that rule could be applied to M because the number of occurrences of the singleton $a \cdot b \cdot a$ in M is one; differently if the rule contained one pattern $\{a \cdot c\}_c$ then it could not be applied to M because the number of occurrences of the singleton $a \cdot c \cdot c$ is two. If a rule contained two instances of the pattern $\{a \cdot c\}_c$ to M , then it could be applied to M .

In order to formally define the behavior of the maximal matching operator we must introduce an auxiliary function; we recursively define a *pattern expansion function* $\langle _ \rangle_-$. Such a function will be used to replace all the occurrences of the maximal matching operator with a pattern which will not contain any instances of the operator. In particular we will have that in the case of the first form of the maximal matching operator the function transforms each maximal matching operator into the union of string patterns, all with the same prefix S followed by different sequence variables. We call *n-expansion* of $\{S\}_X$ the replacement of a maximal matching operator $\{S\}_X$ by a union of n string patterns $S \cdot \tilde{x}_1 \mid \dots \mid S \cdot \tilde{x}_n$. The value of n for the expansion of each maximal matching operator will be given by an auxiliary function $\rho : \mathcal{V}_S \rightarrow \mathbb{N}$, which is a parameter of the pattern expansion function. In the case of the second form of the maximal matching operator, $\{SP_1\}_{SP_2}$, the expansion function gives $SP_1 \cdot SP_2$.

Definition 4.9. (Pattern Expansion Function) *The pattern expansion function $\langle _ \rangle_- : \mathcal{MP} \times (\mathcal{V}_M \rightarrow \mathbb{N}) \rightarrow \mathcal{MP}$ is recursively defined as follows:*

$$\begin{aligned} \langle SP \rangle_\rho &= SP \\ \langle \{SP\}_X \rangle_\rho &= SP \cdot \tilde{x}_1 \mid \dots \mid SP \cdot \tilde{x}_{\rho(X)} \quad \text{where } \tilde{x}_i \in \mathcal{V}_S \\ \langle \{SP_1\}_{SP_2} \rangle_\rho &= SP_1 \cdot SP_2 \\ \langle MP_1 \mid MP_2 \rangle_\rho &= \langle MP_1 \rangle_\rho \mid \langle MP_2 \rangle_\rho \end{aligned}$$

Notice that the pattern expansion function behaves as the identity function over string patterns and also distributes over the operator $_ \mid _$. Furthermore as the function will compute the n -expansion of a maximal matching operator of the first form, then the expanded pattern will represent exactly n strings given a prefix S ; note that in the process

of application of a rule which wants to rewrite all the strings with given prefix, then the semantics of SMSR will provide the choice of the proper ρ function to be used in the expansion.

Finally we can note that multiset variables are replaced using a finite set of sequence variables during patterns expansion. From this consideration multiset variables can be seen as placeholders which occur in the first form of the maximal matching operator.

Example 4.10. Given a multiset $M \equiv a \cdot \tilde{x} \mid \{\{a \cdot b\}_X \mid c$ and $\rho(X) = 3$ we have that

$$\begin{aligned} \langle M \rangle_\rho &= \langle a \cdot \tilde{x} \mid \{\{a \cdot b\}_X \mid c \rangle_\rho \\ &= \langle a \cdot \tilde{x} \rangle_\rho \mid \langle \{\{a \cdot b\}_X \rangle_\rho \mid \langle c \rangle_\rho \\ &= a \cdot \tilde{x} \mid a \cdot b \cdot \tilde{x}_1 \mid a \cdot b \cdot \tilde{x}_2 \mid a \cdot b \cdot \tilde{x}_3 \mid c \end{aligned}$$

Notice that this multiset pattern obtained by the expansion represents only $\rho(X) =$ strings with prefix $a \cdot b$.

Let us now define the following functions:

$Var : \mathcal{MP} \mapsto \wp(\mathcal{V}_\mathcal{E} \cup \mathcal{V}_\mathcal{S})$ such that $Var(M)$ denotes the set of variables that appear in multiset M , formally

$$\begin{aligned} Var(e) &= Var(\epsilon) = \emptyset & \forall e \in \mathcal{E} \\ Var(v) &= \{v\} & \forall v \in \mathcal{V} \\ Var(S \cdot S') &= Var(S) \cup Var(S') \\ Var(\{\{S\}_X) &= Var(S) \cup \{\tilde{x}_i \mid i \in \mathbb{N}\} \\ Var(\{\{S\}_{S'}) &= Var(S) \cup Var(S') \\ Var(M \mid M') &= Var(M) \cup Var(M') \end{aligned}$$

Note that, by the expansion of the maximal matching operator, we have that $Var(\{\{SP\}_X) = Var(SP) \cup \{\tilde{x}_i \mid i \in \mathbb{N}\}$ where $\{\tilde{x}_i \mid i \in \mathbb{N}\}$ is an infinite set of sequence variables. This yields the fact that if there exist at least one occurrence of the first form of the maximal matching operator in a pattern P , then the set $Var(P)$ is infinite. This is due to the fact that, by the definition of the expansion function, we know that a multiset variable X will be replaced with a finite set of sequence variables $\{\tilde{x}_1, \dots, \tilde{x}_n\}$ when $\rho(X) = n$; however we can note that even if the value of n is finite, it cannot be known a priori because the ρ function we choose during a computation will change over time dependently on the current state of the computation.

$Symbols : \mathcal{MP} \mapsto \wp(\mathcal{E})$ such that $Symbols(M)$ denotes the set of elements of \mathcal{E} that appear in multiset M , formally

$$\begin{aligned} Symbols(e) &= \{e\} & \forall e \in \mathcal{E} \\ Symbols(v) &= \emptyset & \forall v \in \mathcal{V} \\ Symbols(S \cdot S') &= Symbols(S) \cup Symbols(S') \\ Symbols(\{\{S\}_X) &= Symbols(S) \\ Symbols(\{\{S\}_{S'}) &= Symbols(S) \cup Symbols(S') \\ Symbols(M \mid M') &= Symbols(M) \cup Symbols(M') \end{aligned}$$

We assume $Symbols$ to be trivially extended to sets of SMSR terms and we define the set of *fresh* symbols for a multiset M as the infinite set $\mathcal{E} \setminus Symbols(M)$.

An *instantiation* is a function $\sigma : \mathcal{V}_E \cup \mathcal{V}_S \rightarrow \mathcal{M}$; we denote by Σ the set of all possible instantiations. Given $M \in \mathcal{MP}$, with $M\sigma$ we denote the multiset obtained by replacing each occurrence of variable v appearing in M with the corresponding instantiation $\sigma(v)$, for instance given a multiset $M \equiv a \cdot \tilde{x}$ and a σ such that $\sigma(\tilde{x}) = b \cdot a$ then $M\sigma \equiv a \cdot b \cdot a$.

We denote with $\sigma(V)$ the set $\{T \mid \sigma(v) = T, v \in V\}$. A multiset M is *ground* if and only if $Var(M) = \emptyset$; a rule $M \mapsto M'$ is ground if and only if both M and M' are ground. We write $Var(R)$ for $Var(M) \cup Var(M')$ and $Symbols(R)$ for $Symbols(M) \cup Symbols(M')$.

Notice that, in a rewrite rule $M \mapsto M'$ we have no constraints on $Var(M)$ and $Var(M')$. In particular, some variables may exist that appear only in M' and not in M . We call these variables *free* and the others, namely those appearing in M , *bound*. Formally, $FV : \mathcal{R} \mapsto \wp(\mathcal{V})$ and $BV : \mathcal{R} \mapsto \wp(\mathcal{V})$ are the functions

$$\begin{aligned} FV(M \mapsto M') &= \{v \mid v \in Var(M') \wedge v \notin Var(M)\} \\ BV(M \mapsto M') &= Var(R) \setminus FV(M, M') = Var(M). \end{aligned}$$

Free variables are related to the generation of fresh data, in particular their instantiation in the process of application of a rule R to a multiset M will be such that $Symbols(\sigma(FV(R))) \cap (Symbols(M) \cup Symbols(R)) = \emptyset$. The semantics of SMSR will provide the choice of the proper instantiation function.

Example 4.11. Given a multiset M and a rule R such that

$$M \equiv a \cdot b \mid c \quad R : a \cdot \tilde{x} \mapsto a \cdot y$$

we have that

$$\begin{aligned} Var(M) &= \emptyset & Var(R) &= \{\tilde{x}, y\} & BV(R) &= \{\tilde{x}\} \\ Symbols(M) &= \{a, b, c\} & Symbols(R) &= \{a\} & FV(R) &= \{y\} \end{aligned}$$

the result of the first application of R to M , when $\sigma = \{(y, d), (\tilde{x}, b)\}$, is

$$a \cdot d \mid c$$

Note that the variable y is instantiated with a fresh symbol d which satisfies the constraint

$$\{d\} \cap (\{a, b, c\} \cup \{a\}) = \emptyset$$

and that any further application of such a rule will need to generate again a fresh symbol. We could get for example the multiset $a \cdot e \mid c$ when applying again the rule R to $a \cdot d \mid c$ with $\sigma' = \{(y, e), (\tilde{x}, d)\}$.

We can now define a formal operational semantics satisfying the required features.

Definition 4.12. (Semantics) Given a finite set of rewrite rules $\mathcal{R} \subset \mathfrak{R}$ the semantics of SMSR is the least labeled transition relation $\xrightarrow{\xi, \zeta}$ closed with respect to \equiv and satisfying the following inference rules:

$$(1) \quad \frac{R : M_1 \mapsto M_2 \in \mathcal{R} \quad \sigma \in \Sigma \quad \exists \rho. (\langle M_1 \rangle_\rho)\sigma \equiv M \wedge (\langle M_2 \rangle_\rho)\sigma \equiv M' \quad (Symbols(\sigma(BV(R))) \cup Symbols(R)) \cap Symbols(\sigma(FV(R))) = \emptyset}{M \xrightarrow{\{SP\sigma \mid \{SP\}_- \in M'\}, Symbols(\sigma(FV(R)))} M'}$$

$$(2) \quad \frac{M \xrightarrow{\xi, \zeta} M' \quad \forall S \in \xi. \nexists S' \in \mathcal{S}. (S \cdot S') \in M'' \quad \zeta \cap Symbols(M'') = \emptyset}{M'' \mid M \xrightarrow{\xi, \zeta} M'' \mid M'}$$

The semantics rules use the concepts of patterns expansion and instantiation.

As regards the behavior of the maximal matching operator, $\{\{SP\}_\perp$, we have that semantic rule (1) expresses that each occurrence of such an operator in a rewrite rule must be expanded and the instantiation of SP by σ , $SP\sigma$, must be recorded as the first label of the transition in the conclusion of the semantic rule. Such a set of labels, denoted by ξ in semantic rule (2), will be used to provide the maximality, in the previously explained sense, required by the maximal matching operator; in particular the constraint

$$\forall S \in \xi. \nexists S' \in \mathcal{S}. (S \cdot S') \in M''$$

when satisfied, guarantees that no strings with a prefix in ξ have not been rewritten and still are contained into multiset M'' .

As regards the generation of fresh data, the constraint in semantic rule (1)

$$(Symbols(\sigma(BV(R))) \cup Symbols(R)) \cap Symbols(\sigma(FV(R))) = \emptyset$$

states that the symbols used in the instantiation of free variables cannot appear in either the symbols used in the instantiation of bound variables or in the symbols that appear in the rewriting rule. Notice that this constraint is not sufficient to guarantee that the set of symbols used in the term representing the current state of the simulation is disjoint from the set of symbols used in the instantiation of free variables. To solve this problem semantic rule (2) uses the constraint

$$\zeta \cap Symbols(M'') = \emptyset$$

that, when satisfied, guarantees the required feature.

As for MSR systems we can now define *SMSR systems* as follows.

Definition 4.13. (*String MultiSet Rewriting System*)

Given a multiset $M \in \mathcal{M}$ and a finite set of rules $R \subset \mathcal{R}$ we define an SMSR system as the pair (M, R) . The time-evolution (an SMSR computation) of such a system is described by the closure of the transition relation $\xrightarrow{\xi, \zeta}$ when applied to M and R .

Analogously to MSR, an SMSR model of a biological system will be represented by using an SMSR system having as initial state a multiset representing the initial solution and will have rewriting rules representing the *events* that may happen in the biological system.

We show now one example of application of such a semantics to an SMSR system; in particular in this example we show only how the semantics provides correctness of the expansion of maximality operator.

Example 4.14. Given a multiset M and a single rule R such that

$$M \equiv a \cdot b \mid b \mid a \cdot a \mid c \qquad R : b \mid \{a\}_X \mapsto \epsilon$$

we show now two different applications of the semantics of SMSR to M :

- the *incorrect* derivation is the following

$$\begin{array}{c}
\sigma = \{(\widetilde{x}_1, a)\} \quad ((b|\{a\}_X)_\rho)\sigma \equiv b|a \cdot a \quad (\langle \epsilon \rangle_\rho)\sigma \equiv \epsilon \\
\rho = \{(X, 1)\} \quad (\{a\} \cup \{a, b\}) \cap = \emptyset \\
(1) \frac{}{a \cdot a \mapsto \epsilon \quad \forall S \in \{a\}. \exists S'. a \cdot S' \in a \cdot b|b|c \quad \emptyset \cap \{a, b, c\} = \emptyset} \\
(2) \frac{}{a \cdot b|b|a \cdot a|c \rightarrow a \cdot b|b|c}
\end{array}$$

and is incorrect because, even if semantic rule (1) can be successfully applied, rule (2) imposes constraints on ξ and M'' that cannot be satisfied with the choice in (2) of $M \equiv b | a \cdot a$. In particular the constraint

$$\forall S \in \{a\}. \exists S'. a \cdot S' \in a \cdot b|b|c$$

is not satisfied when choosing $S' \equiv b$ because $a \cdot b \in a \cdot b|b|c \equiv M''$.

- the *correct* derivation is the following

$$\begin{array}{c}
\sigma = \{(\widetilde{x}_1, a), (\widetilde{x}_2, b)\} \quad ((b|\{a\}_X)_\rho)\sigma \equiv b|a \cdot a|a \cdot b \quad (\langle \epsilon \rangle_\rho)\sigma \equiv \epsilon \\
\rho = \{(X, 2)\} \quad (\{a, b\} \cup \{a, b\}) \cap = \emptyset \\
(1) \frac{}{a \cdot a|a \cdot b \mapsto \epsilon \quad \forall S \in \{a\}. \exists S'. a \cdot S' \in b|c \quad \emptyset \cap \{b, c\} = \emptyset} \\
(2) \frac{}{a \cdot b|b|a \cdot a|c \rightarrow b|c}
\end{array}$$

and is the only correct one with respect to the semantics of SMSR because provides all the required features for operator $\{a\}_X$

4.2 Expressiveness of SMSR

We now examine the expressiveness of SMSR, in particular we show now the Turing completeness of this formalism.

Theorem 4.15. *String MultiSet Rewriting is Turing-Complete.*

Proof. Let us assume a Turing Machine

$$\langle \Sigma \cup \{\#, \diamond\}, Q, \widehat{s} \in Q, F \subseteq Q, \delta \subseteq Q \times \Sigma \cup \{\#\} \rightarrow Q \times \Sigma \cup \{\#\} \times \{L, R, -\} \rangle$$

where

- Σ is an alphabet of symbols enriched with two special symbols $\#$ and \diamond ;
- Q is the set of states in which the machine can be, in particular \widehat{s} is the initial state of the machine and F is a set of states of acceptance;
- δ is the transition relation which describes the behavior of the machine; in particular, given the state s the machine is currently in and the symbol σ it is reading on the tape the relation δ tells the machine what to do. Formally, given $\delta(q, \sigma) = (q', \sigma', M)$, the machine behaves as follows:

1. it writes symbol σ' in place of σ ;
2. it moves the head in the direction described by M ; in particular moves left if $M = L$, right if $M = R$ or makes a non-move if $M = -$;

3. it updates the state of the machine to q' .

A computation of a machine will end successfully when in a state $q \in F$ and reading a symbol σ there are no transitions in δ , namely $((q, \sigma), (q', \sigma', M)) \notin \delta$ for any q' , σ' and M .

The tape will have a sequence of elements in Σ and will have an infinite sequence of blank symbols $\#$ on the left and on the right of the first and of the last symbols of Σ , respectively. The tape will have the following structure

$$\dots \# \# \# \# \# \sigma_1 \sigma_2 \dots \sigma_{n-1} \sigma_n \# \# \# \# \# \dots$$

We now encode a computation of a Turing Machine encoding a configuration of the machine, thus both the tape and the current state in which the machine is, and encoding the transition relation δ . In order to encode the machine we assume an alphabet

$$\mathcal{E} = \Sigma \cup \{\#\} \cup Q \cup \{\triangleright_L, \triangleright_C, \triangleright_R\}$$

and the machine is encoded as follows:

Tape: in the encoding of the tape we will use the following idea: we split the tape into three different strings.

In particular suppose the current tape is

$$\dots \# \# \# \# \# \sigma_1 \dots \sigma_{k-1} \underline{\sigma_k} \sigma_{k+1} \dots \sigma_n \# \# \# \# \# \dots$$

with $\sigma_i \in \Sigma$ and with the current marker pointing $\underline{\sigma_k}$, we will encode the tape into the multiset

$$\triangleright_L \cdot \sigma_1 \cdot \dots \cdot \sigma_{k-1} \mid \triangleright_C \cdot \sigma_k \mid \triangleright_R \cdot \sigma_{k+1} \cdot \dots \cdot \sigma_n$$

Notice that the alphabet is enriched with three new symbols \triangleright_L , \triangleright_C and \triangleright_R denoting the encoding of the portion of tape to the left of current marker (\triangleright_L), the current marker (\triangleright_C) and the portion of tape to the right of the marker (\triangleright_R).

State: the current state $q \in Q$ in which the machine is encoded into the multiset q .

Summarizing, the encoding of the initial configuration of a Turing Machine with input tape

$$\dots \# \# \# \# \# \sigma_1 \dots \sigma_{k-1} \underline{\sigma_k} \sigma_{k+1} \dots \sigma_n \# \# \# \# \# \dots$$

is the SMSR multiset

$$\widehat{s} \mid \triangleright_L \cdot \sigma_1 \cdot \dots \cdot \sigma_{k-1} \mid \triangleright_C \cdot \sigma_k \mid \triangleright_R \cdot \sigma_{k+1} \cdot \dots \cdot \sigma_n$$

Transitions: we show now how to encode the transition relation δ ; in particular, given $((q, \sigma), (q', \sigma', M)) \in \delta$, we add to our SMSR system a rewrite rule accordingly to this schema:

- if $M = -$ we add

$$q \mid \triangleright_C \cdot \sigma \mapsto q' \mid \triangleright_C \cdot \sigma'$$

- if $M = L$ we add

$$q \mid \triangleright_C \cdot \sigma \mid \triangleright_L \cdot \tilde{x} \cdot x \mid \triangleright_R \cdot \tilde{y} \mapsto q' \mid \triangleright_C \cdot x \mid \triangleright_L \cdot \tilde{x} \mid \triangleright_R \cdot \sigma' \cdot \tilde{y}$$

- if $M = R$ we add

$$q \mid \triangleright_C \cdot \sigma \mid \triangleright_L \cdot \tilde{x} \mid \triangleright_R \cdot y \cdot \tilde{y} \mapsto q' \mid \triangleright_C \cdot y \mid \triangleright_L \cdot \tilde{x} \cdot \sigma' \mid \triangleright_R \cdot \tilde{y}$$

For all the transitions in which the required symbol or the rewritten symbol is $\#$, we encode those transitions accordingly to the given schema but the symbol $\#$ is replaced by the empty symbol ϵ .

The Turing Machine is now encoded into an SMSR system and the SMSR computation is able to *emulate* the behavior of the machine. \square

Example 4.16. Let $T = \langle \{a, b\} \cup \{\#\}, \{0, 1\}, 0, \{1\}, \delta \rangle$ be a Turing machine such that

$$R_1 : ((0, a), (0, b, R)) \in \delta$$

$$R_2 : ((0, b), (1, b, -)) \in \delta$$

$$R_3 : ((1, b), (1, a, R)) \in \delta$$

This machine given a tape of symbols a and b simply rewrites all the a -symbols with b -symbols, and viceversa. Let the machine be in initial state 0 with the following tape

$$\dots \# \# \underline{a} a b b \# \# \dots$$

where the marker is on symbol \underline{a} ; we can encode this configuration in the following multiset

$$M \equiv 0 \mid \triangleright_L \mid \triangleright_C \cdot a \mid \triangleright_R \cdot a \cdot b \cdot b$$

With respect to the encoding of the transition relation of the machine we have the following SMSR rewrite rules:

$$R'_1 : 0 \mid \triangleright_L \cdot \tilde{x} \mid \triangleright_C \cdot a \mid \triangleright_R \cdot y \cdot \tilde{y} \mapsto 0 \mid \triangleright_L \cdot \tilde{x} \cdot b \mid \triangleright_C \cdot y \mid \triangleright_R \cdot \tilde{y}$$

$$R'_2 : 0 \mid \triangleright_C \cdot b \mapsto 1 \mid \triangleright_C \cdot b$$

$$R'_3 : 1 \mid \triangleright_L \cdot \tilde{x} \mid \triangleright_C \cdot b \mid \triangleright_R \cdot y \cdot \tilde{y} \mapsto 1 \mid \triangleright_L \cdot \tilde{x} \cdot a \mid \triangleright_C \cdot y \mid \triangleright_R \cdot \tilde{y}$$

The only acceptance computation executed by the Turing machine is the following:

$$\begin{aligned} 0 &\equiv \underline{a} a b b \# \# \dots \\ \rightsquigarrow_{R_1} 0 &: b \underline{a} b b \# \# \dots \\ \rightsquigarrow_{R_1} 0 &: b b \underline{b} b \# \# \dots \\ \rightsquigarrow_{R_2} 1 &: b b \underline{b} b \# \# \dots \\ \rightsquigarrow_{R_3} 1 &: b b a \underline{b} \# \# \dots \\ \rightsquigarrow_{R_3} 1 &: b b a b \underline{\#} \# \# \dots \end{aligned}$$

where notation $n : t$ denotes the current state n of the machine and the tape t and the notation \rightsquigarrow_{R_i} denotes the application of the transition $R_i \in \delta$.

The corresponding SMSR computation emulated by the SMSR system $(M, \{R'_1, R'_2, R'_3\})$ is the following:

$$\begin{aligned}
M &: 0 \mid \triangleright_L \mid \triangleright_C \cdot a \mid \triangleright_R \cdot a \cdot b \cdot b \\
&\xrightarrow{R'_1} \emptyset, \emptyset \quad 0 \mid \triangleright_L \cdot b \mid \triangleright_C \cdot a \mid \triangleright_R \cdot b \cdot b \\
&\xrightarrow{R'_1} \emptyset, \emptyset \quad 0 \mid \triangleright_L \cdot b \cdot b \mid \triangleright_C \cdot b \mid \triangleright_R \cdot b \\
&\xrightarrow{R'_2} \emptyset, \emptyset \quad 1 \mid \triangleright_L \cdot b \cdot b \mid \triangleright_C \cdot b \mid \triangleright_R \cdot b \\
&\xrightarrow{R'_3} \emptyset, \emptyset \quad 1 \mid \triangleright_L \cdot b \cdot b \cdot a \mid \triangleright_C \cdot b \mid \triangleright_R \\
&\xrightarrow{R'_3} \emptyset, \emptyset \quad 1 \mid \triangleright_L \cdot b \cdot b \cdot a \cdot a \mid \triangleright_C \mid \triangleright_R
\end{aligned}$$

Notice that also the SMSR computation halts here because there are no rules that can be applied to the resulting multiset.

Summarizing, we started introducing MSR as a formal language for the description of some biological systems and then extended it introducing the SMSR formalism.

We got interesting features for this formalism, in particular the possibility of dealing with strings that may be seen as structured objects, the use of variables in the rewriting rules, the introduction of a new operator, namely the maximal matching operator, and, finally, the possibility of generating fresh data during computation.

We also gave a proof of Turing completeness for this formalism. In the next chapter, after presenting a general encoding technique for term-based languages, we will see at different levels of details the encoding of two of the mentioned higher level languages: the P-Systems and CLS+.

4.3 Stochastic SMSR

As we examined the possibility of adding a stochastic behavior to MSR systems in Section 3.3, we now examine the same possibility in the context of SMSR.

In the case of SMSR we have that, analogously to the case of MSR, the state of a computation is represented by a ground multiset but, differently from MSR, rewriting rules contain variables and instances of the maximal matching operator.

We must deal with both of these two differences; firstly we will examine the case of variables and then the case of the operator.

As regards variables we have that a rule of SMSR will have to be instantiated whenever it has to be applied; in particular, given an instantiation function σ , we will have to compute the related ground rule with respect to σ . Intuitively, this can be seen as a SMSR rule were a kind of meta-rule which represents all the possible ground rules that can be obtained instantiating the meta-rule for all the possible instantiation function.

Example 4.17. Let M be the following multiset

$$M \equiv a \mid a \cdot b \mid a \cdot c$$

and let R be the following rewrite rule

$$a \cdot x \mid \tilde{y} \mapsto \epsilon$$

We have that, dependently on the instantiation function, there exist many ground rules that can be applied to M :

$$\begin{aligned} a \cdot b \mid a \mapsto \epsilon & \quad \text{when } \sigma(x) = b, \sigma(\tilde{y}) = a \\ a \cdot b \mid a \cdot c \mapsto \epsilon & \quad \text{when } \sigma(x) = b, \sigma(\tilde{y}) = a \cdot c \\ a \cdot c \mid a \mapsto \epsilon & \quad \text{when } \sigma(x) = c, \sigma(\tilde{y}) = a \\ a \cdot c \mid a \cdot b \mapsto \epsilon & \quad \text{when } \sigma(x) = c, \sigma(\tilde{y}) = a \cdot b \\ & \quad \dots \end{aligned}$$

Note that some of them, for instance the second and the forth, may represent equivalent ground rewrite rules.

The process of generating all the ground rewrite rules that can be obtained from a SMSR rule because is quite trivial; let us extend SMSR rules to triples (M, M', f) where $f : \Sigma \rightarrow \mathbb{R}$ is a function which computes, dependently on the instantiation function σ of the variables that appear in the rule, a real value $f(\sigma)$.

Note that, while in the case of MSR where we had a kinetic constant for rules, in this case as we have got meta-rules, we add a function which will compute the kinetic constant dependently on the instantiation function. The kinetic constant used in the MSR stochastic rules can be embedded in the definition of the function f .

As regards the generation of all the ground rewrite rules we have that, given a meta-rule, there exist a finite set of ground rules that can be generated with respect to the current state of the simulation, to all the instantiation function and to all the pattern expansions of the reactants. As already said we will not give a formal definition of this process. Note that, when we have obtained all the ground rewrite rules from a meta-rule, then we have a finite set of MSR rewriting rules in which the kinetic constant is given be the value $f(\sigma)$. Then we can use the stochastic semantics of MSR to simulate the evolution of the SMSR system.

We now make some considerations on the maximal matching operator, in particular we examine, using a simple example, how it influences the rate of a rule. Let N be the current state of a simulation such that

$$N \equiv a \cdot a \mid a \cdot a \mid a \cdot b \quad \text{alternatively } N = \{(a \cdot a, 2), (a \cdot b, 1)\}$$

where the multiset, accordingly to the alternative representation introduced in Section 3.3, can be represented using a function from the language of the strings over $\mathcal{E} \cup \{\epsilon\}$ to \mathbb{N} . Furthermore, let $R = (M, M', f)$ be a rule with reactants

$$M \equiv \{a\}_X$$

where M represents the multiset of all the strings in N with a as prefix, namely all the strings in N . If we have a ρ function such that $\rho(X) = 3$, then the expansion of M with respect to ρ is

$$\langle M \rangle_\rho \equiv a \cdot \tilde{x}_1 \mid a \cdot \tilde{x}_2 \mid a \cdot \tilde{x}_3$$

Let σ be the following instantiation function

$$\sigma = \{(\tilde{x}_1, a), (\tilde{x}_2, a), (\tilde{x}_3, b)\}$$

Then the instantiation of the expansion of M with respect to σ is the ground multiset

$$\langle\langle M \rangle\rangle_\rho \sigma \equiv a \cdot a \mid a \cdot a \mid a \cdot b \quad \text{alternatively } \{(a \cdot a, 2), (a \cdot b, 1)\}$$

We can note that, as we said into Section 3.3, the rate is computed with respect to the following formula:

$$f(\sigma) \prod_{v \in \{a \cdot a, a \cdot b\}} \binom{n(v)}{m(v)} = f(\sigma) \binom{n(a \cdot a)}{m(a \cdot a)} \binom{n(a \cdot b)}{m(a \cdot b)} = f(\sigma) \binom{2}{2} \binom{1}{1} = f(\sigma)$$

We can note, from this simple example, an interesting property for the operator: all the binomial coefficients of all the strings obtained by the instantiation of the expansion of the maximal matching operator are equal to one. This is quite intuitive because there exist one way to pick n items from a set of n items.

This is due to the fact that the semantics of SMSR provides the correct behavior for the operator, namely provides a maximality expansion for the pattern M . Furthermore this property holds even in the case of the other form of the operator. For instance replace M with the pattern $\{a\}_b$ to see that the property holds. Note that it still holds if there are multiple occurrences of the operators in both their forms.

Chapter 5

SMSR as an Intermediate Language

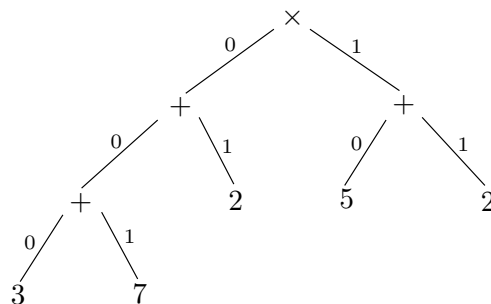
We start this chapter discussing on the encoding of term-based languages into SMSR. In particular the problem we want to face is the encoding of a term into a multiset of strings.

The ideas we will outline in this introduction describe a well-known encoding technique firstly used in [15, 20].

We start with making a simple consideration: with each term its *abstract syntax tree* can always be associated. For instance, given the arithmetic expression represented by the term

$$3 + 7 + 2 \times 5 + 2$$

we assume both the precedence of \times with respect to $+$ and the left-associativity of operator $+$. The abstract syntax tree for term is



where the labels 0 or 1 denote the left-child or the right-child of a node, respectively.

Intuitively, if we encode each path in the abstract syntax tree starting from the root and ending in a leaf with a string over an alphabet \mathcal{E} , then we have a multiset representation of the tree. The alphabet we choose is $\mathcal{E} = \{2, 3, 5, 7, +_0, +_1, \times_0, \times_1\}$.

In this case we get that the multiset

$$\begin{aligned} & \times_0 \cdot +_0 \cdot +_0 \cdot 3 \mid \\ & \times_0 \cdot +_0 \cdot +_1 \cdot 7 \mid \\ & \times_0 \cdot +_1 \cdot 2 \mid \\ & \times_1 \cdot +_0 \cdot 5 \mid \\ & \times_1 \cdot +_1 \cdot 2 \end{aligned}$$

represents all the paths from \times to the leafs in the abstract syntax tree for term. Note that, for all the paths in the abstract syntax tree that share a sub-path, then they share a prefix in the encoding. For instance as the path from \times to 3 and the path from \times to 7 have as sub-path the path from \times to their common father, then they have the same prefix $\times_0 \cdot +_0$.

We will encode now, using such a technique, sequential P-Systems. We will not give formal definitions of the encoding functions because the encoding of P-System is shown as a simple example of application of this encoding technique.

In the last section of the chapter we will give, formally, the encoding of the Calculus of Looping Sequences in one of its variants, namely the CLS+ formalism.

5.1 Encoding P-Systems

5.1.1 Definition of 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, or they pass through membranes, in two directions: they 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 5.1 (P-Systems). *A P-System is a tuple*

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

where:

- 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 $1, 2, \dots, n$ of μ .
- R_i with $1 \leq i \leq n$ are finite sets of evolution rules associated with the regions $1, 2, \dots, n$ of μ . An evolution rule is a pair (u, v) where u is a string over V and v is a string over $(V \times \{\text{here, out}\}) \cup (V \times \{\text{in}_j \mid 1 \leq j \leq n\}) \cup \{\delta\}$ 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: $(r_1, r_2) \in \rho_i$ 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 [35].

Example 5.2. Consider the following P-System:

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

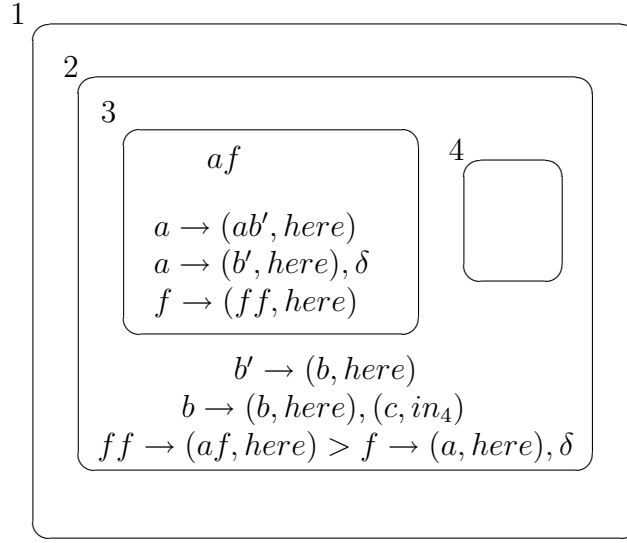


Figure 5.1: Example of a P-System generating n^2 , with $n \geq 1$.

where:

$$\begin{aligned}
 V &= \{a, b, b', c, f\} \\
 \mu &= \{(1, 2), (2, 3), (2, 4)\} \\
 w_1 &= \emptyset, R_1 = \emptyset, \rho_1 = \emptyset \\
 w_2 &= \emptyset \\
 R_2 &= \{r_1 : b' \rightarrow (b, \text{here}), r_2 : b \rightarrow (b, \text{here})(c, \text{in}_4), \\
 &\quad r_3 : (ff, \text{here}) \rightarrow (af, \text{here}), r_4 : f \rightarrow (a, \text{here})\delta\} \\
 \rho_2 &= \{r_3 > r_4\} \\
 w_3 &= \emptyset, R_3 = \{a \rightarrow (ab, \text{here}), a \rightarrow (b', \text{here})\delta, f \rightarrow (ff, \text{here})\}, \rho_3 = \emptyset \\
 w_4 &= \emptyset, R_4 = \emptyset, \rho_4 = \emptyset
 \end{aligned}$$

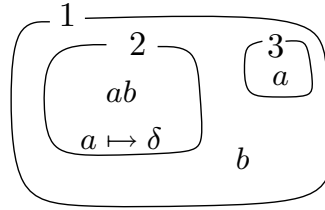
The system is shown in Figure 5.1

5.1.2 Encoding P-Systems

In this section we will give intuitions on the encoding of sequential P-Systems [17] into SMSR. Sequential P-Systems represent a variant of P-Systems introduced in Section 5.1.1; in particular, they have a sequential semantics rather than a maximally parallel one.

We will not give a formal definition of the encoding functions and a proof of equivalence of the formalisms, but we will present the encoding by referring to a running example.

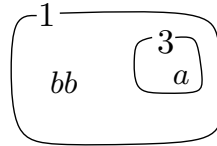
Let P be the following P-System



where:

- the membrane 1 contains an object b and both the membranes 2, 3 and their content;
- the membrane 3 contains an object a and no other nested membranes;
- the membrane 2 contains an object a and an object b but no other nested membranes;
- in this P-System there is just one rule contextually to membrane 2, namely $a \rightarrow \delta$.
Such a rule, when applied, will destroy both membrane 2 and the object a , while object b will move into membrane 1.

The result of applying the only rule of P-System P is the following P-System P' :



Referring to the encoding technique that we have previously examined, we can encode P into the multiset

$$M \equiv 1 \cdot b \mid 1 \cdot 2 \cdot a \mid 1 \cdot 2 \cdot b \mid 1 \cdot 3 \cdot a$$

where $\mathcal{E} = \{1, 2, 3, a, b\}$ and

- $1 \cdot b$ denotes the containment of the b object in the membrane 1;
- $1 \cdot 2 \cdot a$ denotes the containment of the a object in the membrane 2; notice that as the membrane 2 is nested into the membrane 1, then the encoding of its content will have as a prefix the encoding of the membrane 1, namely the string 1;
- $1 \cdot 2 \cdot b$ denotes the containment of the b object into the membrane 2;
- $1 \cdot 3 \cdot a$ denotes the containment of the a object into the membrane 3 which is contained into the membrane 1;

Note that the identifier of each membrane $i \in \mathbb{N}$ is encoded by using the identity function. This is due to the fact that, as already said, in P-Systems the name of a membrane is univocally determined by its natural number. We will see that in the encoding of CLS+ this is different.

We examine now the encoding of the only rule of the P-System P , in particular we show three different encodings of the rule, but only one will be correct with respect to the semantics of the P-Systems:

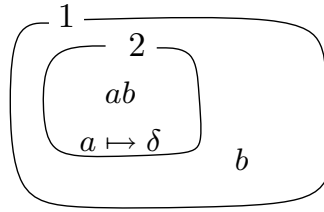
- Let the rule be encoded into the following SMSR rewriting rule

$$\tilde{x} \cdot a \mapsto \epsilon$$

if we apply this rule to M using as instantiation a function $\sigma = \{(\tilde{x}, 1 \cdot 3)\}$, then we get the multiset

$$M \equiv 1 \cdot b \mid 1 \cdot 2 \cdot a \mid 1 \cdot 2 \cdot b$$

which corresponds, when reversing the encoding, to the following P-System



Note that this P-System is completely different from P' , and this is due to the fact that we have used a sequence variable \tilde{x} to encode the prefix of the string a to be rewritten. This is obviously not correct because, as variables can be instantiated to any value, we lose the context of application of the rule in the P-System. In other words, using a variable we express a rule that can be applied everywhere inside the P-System and this is very different from the expected behavior. We will have then to express explicitly the context of application of the rule, namely $1 \cdot 2$.

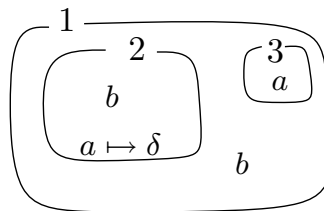
- Let the rule be encoded into the following SMSR rewriting rule

$$1 \cdot 2 \cdot a \mapsto \epsilon$$

if we apply this rule to M we get the multiset

$$M' \equiv 1 \cdot b \mid 1 \cdot 2 \cdot b \mid 1 \cdot 3 \cdot a$$

which corresponds to the following P-System



Note that this P-System is different from the required one, namely P' , and this is due to the fact that we have not expressed the movement of the object b contained in the membrane 2 inside membrane 1. We need to be able to express the movement of all the content of membrane 2 which is not destroyed from inside the membrane to outside the membrane.

- Let the rule be encoded into the following SMSR rewriting rule

$$1 \cdot 2 \cdot a \mid \{1 \cdot 2\}_X \mapsto \{1\}_X$$

If we apply this rule to M we get the multiset

$$M' \equiv 1 \cdot b \mid 1 \cdot b \mid 1 \cdot 3 \cdot a$$

which corresponds to the desired P-System P' . This is given by the fact that the maximal matching operator $\{1 \cdot 2\}_X$ represents all the content of the membrane 2 which is exactly the term $1 \cdot 2 \cdot a$ because the term $1 \cdot 2 \cdot b$ explicitly appears in the reactants of the rule.

Furthermore, as in the multiset of the products of the rule we rewrite the term $\{1 \cdot 2\}_X$ into the term $\{1\}_X$, then we change the prefix of the object $1 \cdot 2 \cdot b$ to $1 \cdot b$ which corresponds to the expected behavior of the P-System. Note that this methodology would have been correct even if the membrane 2 had contained other membranes or even if we would have moved its content into a membrane different from 1. In this last case the rewritten prefix should have been changed accordingly to the encoding of the destination membrane.

Summarizing, we introduced P-Systems and we discussed on the possibility of encoding P-Systems into SMSR systems. We saw that the required encoding is trivial to be defined; moreover, P-Systems with a sequential semantics can be simulated by SMSR systems.

We will show now the formal encoding of the CLS+ formalism.

5.2 Encoding the Calculus of Looping Sequences

5.2.1 Definition of CLS+

In this section we recall the Calculus of Looping Sequences (CLS) in one of its variants, CLS+[32, 5]. CLS+ is essentially based on term rewriting, hence a CLS+ model consists of a term and a set of rewrite rules. The term is intended to represent the structure of the modeled system, and the rewrite rules the events that may cause the system to evolve.

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

Definition 5.3. (Terms) Terms T , Branes B , and Sequences S of CLS+ are given by the following grammar:

$$\begin{aligned} T & ::= S \mid (B)^L \rfloor T \mid T \mid T \\ B & ::= S \mid B \mid B \\ S & ::= \epsilon \mid a \mid S \cdot S \end{aligned}$$

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.

In CLS+ we have a sequencing operator \cdot , a looping operator $(_)^L$, a parallel composition operator \mid and a containment operator \rfloor . Sequencing can be used to concatenate

elements of the alphabet \mathcal{E} . The empty sequence ϵ denotes the concatenation of zero symbols. A term can be either a sequence, or a looping sequence (that is the application of the looping operator to a parallel composition of sequences) containing another term, or the parallel composition of two terms. By definition, looping and containment are always applied together, hence we can consider them as a single binary operator $(-)^L \rfloor -$ which applies to one brane and one term. Brackets can be used to indicate the order of application of the operators, and we assume $(-)^L \rfloor -$ to have precedence over $-|-$.

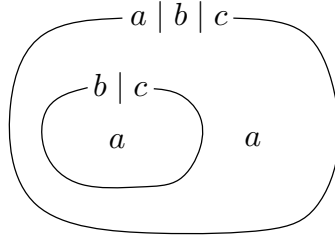
Example 5.4. Consider the CLS+ term

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

It represents the nesting of two membranes where

- the top-level membrane contains a multiset of objects in its looping sequence, namely $a | b | c$;
- the top-level membrane contains both an object a and another membrane;
- the nested membrane contains a multiset of objects in its looping sequence, namely $b | c$;
- the nested membrane contains only one object a .

A visual representation of this CLS+ term is the following



In CLS+ we may have syntactically different terms representing the same structure. The structural congruence relation of CLS+ expresses associativity of both \cdot and $|$, commutativity of the latter and the neutral role of ϵ with respect to all the operators.

Definition 5.5. (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:*

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

$$S_1 \equiv_S S_2 \text{ implies } S_1 \equiv_B S_2$$

$$B_1 | B_2 \equiv_B B_2 | B_1 \quad B_1 | (B_2 | B_3) \equiv_B (B_1 | B_2) | B_3 \quad B | \epsilon \equiv_B B$$

$$S_1 \equiv_S S_2 \text{ implies } S_1 \equiv_T S_2 \quad B_1 \equiv_B B_2 \text{ implies } (B_1)^L \rfloor T \equiv_T (B_2)^L \rfloor T$$

$$T_1 | T_2 \equiv_T T_2 | T_1 \quad T_1 | (T_2 | T_3) \equiv_T (T_1 | T_2) | T_3 \quad T | \epsilon \equiv_T T \quad (\epsilon)^L \rfloor \epsilon \equiv \epsilon$$

Rewrite rules will be defined essentially as pairs of terms, in which the first term describes the portion of the system in which the event modeled by the rule may occur, and the second term describes how that portion of the system changes when the event occurs. In the terms of a rewrite rule we allow the use of variables. As a consequence, a rule will be applicable to all terms which can be obtained by properly instantiating variables. Variables can be of four kinds associated with terms, branes, sequences, and alphabet elements, respectively. We assume a set of term variables TV ranged over by X, Y, Z, \dots , a set of brane variables BV ranged over by $\bar{x}, \bar{y}, \bar{z}, \dots$, a set of sequence variables SV ranged over by $\tilde{x}, \tilde{y}, \tilde{z}, \dots$, and a set of element variables \mathcal{X} ranged over by x, y, z, \dots . All these sets are possibly infinite and pairwise disjoint. We denote by \mathcal{V} the set of all variables, $\mathcal{V} = TV \cup BV \cup SV \cup \mathcal{X}$, and with ρ a generic variable of \mathcal{V} . Hence, a pattern is a term which may include variables and a rewrite rule is a pair of patterns.

Definition 5.6. (Patterns) Patterns P , brane patterns BP and sequence patterns SP of $CLS+$ are given by the following grammar:

$$\begin{aligned} P & ::= SP \mid (BP)^L \rfloor P \mid P \mid P \mid X \\ BP & ::= SP \mid BP \mid BP \mid \bar{x} \\ SP & ::= \epsilon \mid a \mid SP \cdot SP \mid \tilde{x} \mid x \end{aligned}$$

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

Definition 5.7. (Rewrite Rules) A rewrite rule is a pair of patterns (P_1, P_2) , written as $P_1 \mapsto P_2$, where $P_1, P_2 \in \mathcal{P}$, $P_1 \neq \epsilon$ and such that $\text{Var}(P_2) \subseteq \text{Var}(P_1)$. We denote with \mathfrak{R} the infinite set of all the possible rewrite rules.

Example 5.8. Let R be the following $CLS+$ rule

$$((a)^L \rfloor X) \mid ((Y)^L \rfloor Z) \mapsto (a \mid Y)^L \rfloor X \mid Z$$

This rule states that, whenever there exist two juxtaposed membranes in a $CLS+$ term, if one of them has got an a object as its looping sequence, then the two membranes merge. In particular they are rewritten into a single membrane containing the union of both the starting looping sequences as its looping sequence, and containing also the merging of the objects contained in both membranes.

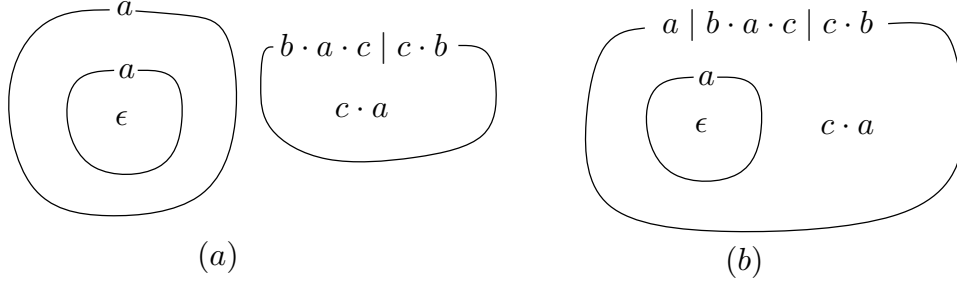
As expected, if we apply rule R to a term T such that

$$T \equiv ((a)^L \rfloor (a)^L \rfloor \epsilon) \mid (b \cdot a \cdot c \mid c \cdot b)^L \rfloor c \cdot a$$

we get

$$T \equiv (a \mid b \cdot a \cdot c \mid c \cdot b)^L \rfloor (a)^L \rfloor \epsilon \mid c \cdot a$$

In the following figure is given a graphical representation the term T (a) before the application of the rule and (b) after:



In CLS+ we have some rules that are applicable everywhere in a term, while others cannot be applied to branes. For instance, a rule such as $a | b \mapsto (a)^L \rfloor b$ cannot be applied to the elements of a looping (as in $(a | b)^L \rfloor c$) because the result of the application would not be a syntactically correct CLS+ term ($((a)^L \rfloor b)^L \rfloor c$). 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 $\mathfrak{R}_{\mathcal{B}} \subset \mathfrak{R}$ 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 5.9. (Semantics) *Given a set of rewrite rules $\mathcal{R} \subseteq \mathfrak{R}$, and a set of brane rules $\mathcal{R}_{\mathcal{B}} \subseteq \mathcal{R}$, such that $(\mathcal{R} \setminus \mathcal{R}_{\mathcal{B}}) \cap \mathfrak{R}_{\mathcal{B}} = \emptyset$, the semantics of CLS is the least transition relation \rightarrow on terms closed under \equiv , and satisfying the following inference rules:*

$$\frac{(P_1, P_2) \in \mathcal{R} \quad P_1 \sigma \neq \epsilon \quad \sigma \in \Sigma}{P_1 \sigma \rightarrow P_2 \sigma} \quad \frac{T_1 \rightarrow T_2}{T | T_1 \rightarrow T | T_2} \quad \frac{T_1 \rightarrow T_2}{(B)^L \rfloor T_1 \rightarrow (B)^L \rfloor T_2}$$

$$\frac{(BP_1, BP_2) \in \mathcal{R}_{\mathcal{B}} \quad BP_1 \sigma \neq \epsilon \quad \sigma \in \Sigma}{BP_1 \sigma \rightarrow_{\mathcal{B}} BP_2 \sigma} \quad \frac{B_1 \rightarrow_{\mathcal{B}} B_2}{B | B_1 \rightarrow_{\mathcal{B}} B | B_2}$$

$$\frac{B_1 \rightarrow_{\mathcal{B}} B_2}{(B_1)^L \rfloor T \rightarrow (B_2)^L \rfloor T}$$

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.

The relation $\rightarrow_{\mathcal{B}}$ 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.

5.2.2 Encoding of CLS+

Firstly we notice that, for the sake of simplicity, we assume a slight restriction on the syntax of CLS+ patterns. In particular, we require that term and brane variables appear always inside the operands of some $(-)^L \rfloor -$ operator. In other words, we avoid rewrite rules having the following forms:

$$X | \dots \mapsto \dots \quad \bar{x} | \dots \mapsto \dots$$

The reason for this restriction is that, as we shall see, we will use the maximal matching operator of SMSR to encode term and brane variables of CLS+. This means that the

translation of a term or of a brane variable will be always instantiated in a maximal way. This is correct with respect to the semantics of CLS+ for variables that appear in the operand of a looping and containment operator as shown in the following example.

Example 5.10. Given CLS+ terms

$$T \equiv (m)^L \rfloor (a \mid b \mid c) \qquad T' \equiv a \mid b \mid c$$

and rules

$$R : (m)^L \rfloor (a \mid X) \mapsto \epsilon \qquad R' : a \mid X \mapsto \epsilon$$

we have only the following result by applying R to T :

$$T \xrightarrow{X=b|c} (m)^L \rfloor (\epsilon)$$

and this is the only application with maximality instantiation for X . Differently, as regards the application of R' to T' , we have

$$T \xrightarrow{X=b|c} \epsilon \quad T \xrightarrow{X=b} c \quad T \xrightarrow{X=c} b \quad T \xrightarrow{X=\epsilon} b \mid c$$

and the only application with maximality instantiation for X is $T' \xrightarrow{X=b|c} \epsilon$.

This restriction could be avoided by defining also a non-maximal matching operator in SMSR; the definition of the semantics of such an operator would be quite straightforward.

We will give two encoding functions that map CLS+ terms and patterns into SMSR terms and patterns, respectively. These functions will be used to translate both the rewrite rules and the initial term of a CLS+ model. The encoding functions are defined, accordingly to the general encoding technique presented at the beginning of this chapter, recursively on the structure of the CLS+ term or pattern, hence they perform a complete visit of the abstract syntax tree of such a term or pattern from root to leaves.

The tree we take into account is a view on the abstract syntax tree of CLS+ terms, in particular we will encode CLS+ sequences into SMSR strings, namely we will have as leafs CLS+ sequences; furthermore we will not encode the $_ \mid _$ operator. This last choice is due to the fact that the commutativity of the operator, if encoded, would have yield a more complex encoding.

While performing this visit, the encoding functions construct one SMSR string for each path from the root to one leaf and, eventually, they concatenate a string corresponding to the leaf to the string corresponding to its path in the tree. The result of the translation of a CLS+ pattern is hence a multiset of strings. We have a multiset of strings instead of a set because, as we shall see, the encoding will consider only the nodes of the abstract syntax tree corresponding to applications of the looping operator.

We assume that each element $e \in \mathcal{E}_{\text{CLS+}}$, where $\mathcal{E}_{\text{CLS+}}$ is the alphabet of CLS+, is also contained in the alphabet of SMSR. Moreover, we assume that for each element variable x and sequence variable \tilde{x} of CLS+, the same x and \tilde{x} exist in $\mathcal{V}_{\mathcal{E}}$ and $\mathcal{V}_{\mathcal{S}}$, respectively. We assume also that $\mathcal{V}_{\mathcal{S}}$ contains a special variable Δ that will be used in the encoding of rewrite rules. Finally, for each brane variable \bar{x} and term variable X of CLS+ we assume the existence of \bar{x} and X in $\mathcal{V}_{\mathcal{M}}$ and of infinitely many variables \bar{x}_i and X_i , for all $i \in \mathbb{N}$, in $\mathcal{V}_{\mathcal{S}}$.

In order to encode paths in the abstract syntax tree of CLS+ terms and patterns we assume that the SMSR alphabet \mathcal{E} contains two symbols $\bar{\lambda}$ and λ and all the natural numbers \mathbb{N} . The two symbols $\bar{\lambda}$ and λ are used to distinguish between the branes and the contents in an application of the looping operator, and the natural numbers are used to distinguish two different applications of the looping operator. The two symbols $\bar{\lambda}$ and λ will be always followed by either a natural number or an element variable. To simplify the notation we will write λ_i and λ_x for $\lambda \cdot i$ and $\lambda \cdot x$, respectively, and the same with $\bar{\lambda}$ instead of λ .

We now define the encoding of CLS+ terms into SMSR multisets and of CLS+ patterns into SMSR multiset patterns. In the definitions we will use an auxiliary injection function $\triangleright : \mathcal{SP} \times \mathcal{MP} \rightarrow \mathcal{MP}$ that inserts a sequence pattern SP as a prefix of all the elements of a multiset pattern MP . The injection function is represented with infix notation and is recursively defined as follows:

$$\begin{aligned} SP_1 \triangleright SP_2 &= SP_1 \cdot SP_2 \\ SP \triangleright (MP_1 \mid MP_2) &= (SP \triangleright MP_1) \mid (SP \triangleright MP_2) \\ SP_1 \triangleright \{\{SP_2\}_X\} &= \{\{SP_1 \cdot SP_2\}_X\} \\ SP_1 \triangleright \{\{SP_2\}_{SP_3}\} &= \{\{SP_1 \cdot SP_2\}_{SP_3}\} \end{aligned}$$

Example 5.11. The injection of the string $a \cdot b$ into the multiset $M \equiv a \mid \{\{a \cdot c\}_X\} \mid \{\{c\}_b\}$ is

$$a \cdot b \triangleright M = a \cdot b \cdot a \mid \{\{a \cdot b \cdot a \cdot c\}_X\} \mid \{\{a \cdot b \cdot c\}_b\}$$

We define the encoding of CLS+ terms. The encoding technique is the same as the one used in [15, 20] to define enhanced semantics for the study of causality properties. Here, the idea is to represent a path in the abstract syntax tree of CLS+ term as a sequence of $\bar{\lambda}_i$ and λ_i symbols representing applications of the looping operator. As already mentioned, we do not use any symbol to represent applications of both the parallel composition operator \mid of CLS+ as it is directly translated into multiset union of SMSR and for the sequencing operator \cdot of CLS+ that is directly translated into SMSR string concatenation.

Definition 5.12. (Encoding of terms) *The encoding of CLS+ terms into multisets of strings is given by the function $\lfloor _ \rfloor : \mathcal{T} \rightarrow \mathcal{M} \times \wp(\mathbb{N})$ recursively defined as follows:*

$$\begin{aligned} \lfloor S \rfloor &= (S, \emptyset) \\ \lfloor T_1 \mid T_2 \rfloor &= (M_1 \mid M_2, I_1 \cup I_2) \quad \text{where } \lfloor T_i \rfloor = (M_i, I_i) \text{ and } I_1 \cap I_2 = \emptyset \\ \lfloor (B)^L \rfloor T \rfloor &= (\bar{\lambda}_i \triangleright M_1 \mid \lambda_i \triangleright M_2, I_1 \cup I_2) \\ &\quad \text{where } \lfloor B \rfloor = (M_1, I_1), \lfloor T \rfloor = (M_2, I_2) \text{ and } i \in \mathbb{N} \setminus (I_1 \cup I_2) \end{aligned}$$

The encoding of terms translates a CLS+ term T into a pair (M, I) , where M is the actual result of the translation, namely the SMSR multiset corresponding to T , and I is the set of natural numbers occurring in M . Such a set of numbers is used in the definition of the encoding to ensure that different applications of the looping operator in T will be translated into occurrences of $\bar{\lambda}_i$ and λ_i having different indexes. In what follows, we will ignore this set of natural numbers and we will use $\lfloor T \rfloor$ to denote only the SMSR multiset M .

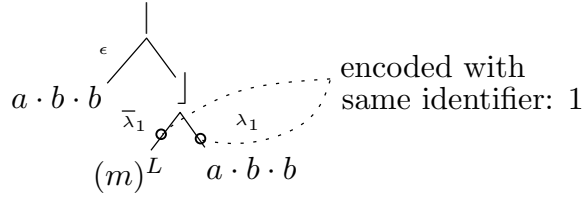


Figure 5.2: Sample encoding

Example 5.13. Given a CLS+ term

$$T \equiv a \cdot b \cdot b \mid (m)^L \rfloor (a \cdot b \cdot b)$$

its encoding is

$$a \cdot b \cdot b \mid \overline{\lambda_1} \cdot m \mid \lambda_1 \cdot a \cdot b \cdot b$$

having used the singleton $\{1\}$ as set of natural numbers. Notice that the two sequences $a \cdot b \cdot b$ of T have a different encoding of their prefixes because they have different paths in the view on the abstract syntax tree as shown in Figure 5.2.

Now we define the encoding of CLS+ patterns into SMSR multiset patterns. This encoding will be used to translate CLS+ rewrite rules into rewrite rules of SMSR.

Definition 5.14. (Encoding of patterns) *The encoding of CLS+ patterns into multiset patterns is given by the function $\llbracket _ \rrbracket : \mathcal{P} \rightarrow \mathcal{MP} \times \mathcal{V}_{\mathcal{E}} \times \mathcal{V}_{\mathcal{E}}$ recursively defined as follows:*

$$\llbracket SP \rrbracket = (SP, \emptyset, \text{Var}(SP))$$

$$\llbracket v \rrbracket = (\{\epsilon\}_v, \emptyset, \{v_i \mid i \in \mathbb{N}\}) \quad \forall v \in TV \cup BV$$

$$\begin{aligned} \llbracket P_1 \mid P_2 \rrbracket &= (MP_1 \mid MP_2, \Sigma_1 \cup \Sigma_2, \Sigma'_1 \cup \Sigma'_2) \\ &\text{where } \llbracket P_i \rrbracket = (MP_i, \Sigma_i, \Sigma'_i) \text{ and } \Sigma_1 \cap \Sigma_2 = \emptyset \end{aligned}$$

$$\begin{aligned} \llbracket (BP)^L \rfloor P \rrbracket &= (\overline{\lambda_x} \triangleright MP_1 \mid \lambda_x \triangleright MP_2, \{x\} \cup \Sigma_2, \Sigma'_1 \cup \Sigma'_2) \\ &\text{where } \llbracket BP \rrbracket = (MP_1, \emptyset, \Sigma'_1), \llbracket P \rrbracket = (MP_2, \Sigma_2, \Sigma'_2), \\ &\text{and } x \in \mathcal{V}_{\mathcal{E}} \setminus (\Sigma_1 \cup \Sigma'_1 \cup \Sigma_2 \cup \Sigma'_2) \end{aligned}$$

where the auxiliary encoding function $\llbracket _ \rrbracket : \mathcal{P} \rightarrow \mathcal{MP} \times \mathcal{V}_{\mathcal{E}} \times \mathcal{V}_{\mathcal{E}}$ is recursively defined as follows:

$$\llbracket SP \rrbracket = (\{\epsilon\}_{SP}, \emptyset, \text{Var}(SP))$$

$$\llbracket v \rrbracket = (\{\epsilon\}_v, \emptyset, \{v_i \mid i \in \mathbb{N}\}) \quad \forall v \in TV \cup BV$$

$$\begin{aligned} \llbracket P_1 \mid P_2 \rrbracket &= (MP_1 \mid MP_2, \Sigma_1 \cup \Sigma_2, \Sigma'_1 \cup \Sigma'_2) \\ &\text{where } \llbracket P_i \rrbracket = (MP_i, \Sigma_i, \Sigma'_i) \text{ and } \Sigma_1 \cap \Sigma_2 = \emptyset \end{aligned}$$

$$\begin{aligned} \llbracket (BP)^L \rfloor P \rrbracket &= (\overline{\lambda_x} \triangleright MP_1 \mid \lambda_x \triangleright MP_2, \{x\} \cup \Sigma_2, \Sigma'_1 \cup \Sigma'_2) \\ &\text{where } \llbracket BP \rrbracket = (MP_1, \emptyset, \Sigma'_1), \llbracket P \rrbracket = (MP_2, \Sigma_2, \Sigma'_2), \\ &\text{and } x \in \mathcal{V}_{\mathcal{E}} \setminus (\Sigma_1 \cup \Sigma'_1 \cup \Sigma_2 \cup \Sigma'_2) \end{aligned}$$

The encoding of patterns translates a CLS+ pattern P into a triple (MP, Σ, Σ') , where MP is the actual result of the translation, namely the SMSR multiset pattern corresponding to P , the set Σ contains all the element variables that are used in MP as subscripts of $\bar{\lambda}$ and λ symbols, and the set Σ' contains all the other variables that may appear in MP . The set Σ is used to ensure that different applications of the looping operator in P will be translated into occurrences of symbols $\bar{\lambda}$ and λ having different subscripts. The set Σ' , instead, will be used in the following to translate CLS+ rewrite rules. In what follows, when we do not represent explicitly the triple (MP, Σ, Σ') obtained from the encoding, we will use $\llbracket T \rrbracket$ and $\langle T \rangle$ to denote only the SMSR multiset pattern MP .

Example 5.15. Given the CLS+ pattern

$$a \cdot \tilde{x} \mid (m \cdot \tilde{y})^L \mid (X \mid a)$$

its encoding is the SMSR pattern

$$\Delta \cdot a \cdot \tilde{x} \mid \Delta \cdot \bar{\lambda}_x \cdot m \cdot \tilde{y} \mid \{\Delta \cdot \lambda_x\}_X \mid \{\Delta \cdot \lambda_x\}_a$$

where the set of variables used is $\{x, y, \Delta\} \subset \mathcal{V}$.

Now, a CLS+ model consisting in an initial term T and a set of rewrite rules $\mathcal{R} = \{P_1 \mapsto P'_1, \dots, P_n \mapsto P'_n\}$ can be translated into a SMSR model consisting in the initial term $\llbracket T \rrbracket$ and in a set of rewrite rules derived from \mathcal{R} . The translation of CLS+ rewrite rules uses the encoding $\llbracket - \rrbracket$ and is different in the two cases of brane rules and non-brane rules.

The translation of a CLS+ brane rule $BP_1 \mapsto BP_2 \in \mathfrak{R}_B$ is simple as brane rules can be applied everywhere in a CLS+ term. Consequently, the SMSR rule corresponding to $BP_1 \mapsto BP_2$ is

$$\Delta \triangleright MP_1 \mapsto \Delta \triangleright MP_2 \tag{5.1}$$

where $\llbracket BP_1 \rrbracket = (MP_1, \Sigma_1, \Sigma'_1)$, $\llbracket BP_2 \rrbracket = (MP_2, \Sigma_2, \Sigma'_2)$, $\Sigma_1 \cap \Sigma_2 = \emptyset$ and $\Delta \notin \Sigma_1 \cup \Sigma'_1 \cup \Sigma_2 \cup \Sigma'_2$.

Note that, as shown in Figure 5.3, the instantiation of the special variable Δ added in MP_1 and in MP_2 will represent, during the application of R , the path from the root of the abstract syntax tree to the point in the tree in which the rule will be applied.

Example 5.16. Given the brane rule

$$a \cdot \tilde{x} \mid b \cdot \tilde{x} \mapsto a \cdot b \cdot \tilde{x} \cdot \tilde{x}$$

its encoding is

$$\Delta \cdot a \cdot \tilde{x} \mid \Delta \cdot b \cdot \tilde{x} \mapsto \Delta \cdot a \cdot b \cdot \tilde{x} \cdot \tilde{x}$$

The case of the translation of a non-brane CLS+ rule, namely of a rule $P_1 \mapsto P_2 \notin \mathfrak{R}_B$, is slightly more complicated as such a rule in CLS+ can be applied only either to the components of a parallel composition at the top level of the term, or to the components of a parallel composition inside some looping sequence. In order to obtain the corresponding result after translation into SMSR, we translate the rule $P_1 \mapsto P_2$ into two SMSR rewrite rules, namely:

$$MP_1 \mapsto MP_2 \quad \text{and} \quad \Delta \cdot \lambda_x \triangleright MP_1 \mapsto \Delta \cdot \lambda_x \triangleright MP_2$$

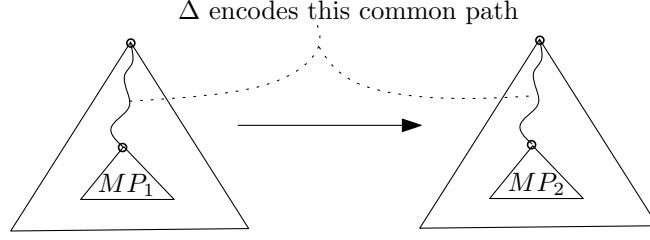


Figure 5.3: Visual representation of the variable Δ in the application of rewrite rules.

where $\llbracket P_1 \rrbracket = (MP_1, \Sigma_1, \Sigma'_1)$, $\llbracket P_2 \rrbracket = (MP_2, \Sigma_2, \Sigma'_2)$, $\Sigma_1 \cap \Sigma_2 = \emptyset$ and $\{\Delta, x\} \cap (\Sigma_1 \cup \Sigma'_1 \cup \Sigma_2 \cup \Sigma'_2) = \emptyset$.

The first of the two SMSR rules will be applicable only to strings representing components of a parallel composition at the top level of the CLS+ term, and the second SMSR rule only to strings representing components of a parallel composition inside some looping sequence in the CLS+ term.

Example 5.17. Given non-brane rule

$$(a)^L \rfloor X \mapsto X$$

its encoding are the following rules

$$\begin{aligned} \{\{\Delta \cdot \lambda_y \cdot \bar{\lambda}_x\}_a \mid \{\{\Delta \cdot \lambda_y \cdot \lambda_x\}_X \mapsto \{\{\Delta \cdot \lambda_y\}_X \\ \{\{\bar{\lambda}_x\}_a \mid \{\{\lambda_x\}_X \mapsto \{\{\epsilon\}_X \end{aligned}$$

It is clear that the first rule can be applied only inside the encoding of the containment of a membrane, this because the injected prefix is $\Delta \cdot \lambda_y$; differently the second rule can be applied only at top-level in a term because it does not contain the Δ variable.

Summarizing, we have defined the encoding of CLS+ into the SMSR formalism using mainly two encoding functions: one for terms and the other for patterns. We also have treated differently the encoding of brane rules and non-brane rules in order to have a correct behavior with respect of the CLS+ semantics.

Before showing an example of an SMSR computation encoding a CLS+ one and before giving a formal proof of equivalence of CLS+ and SMSR, we examine whether the property on the maximal matching operator outlined in Section 4.3 has implications or not on the encoding.

In particular, we can note that the maximal matching operator is used, for instance, to encode term variables of CLS+. Those variables are used, generally, to express the content of a part of a membrane and to express the movement of the content itself. For instance, given a rule which states that a membrane is destroyed and all its content is spread outside the membrane, namely

$$(\tilde{y})^L \rfloor X \mapsto X$$

we would like that the rate of the rule would depend only on the number of membranes with looping sequence equal to $\sigma(\tilde{y})$, for any function σ . We know that the term variable

X used inside such a rule would be encoded by using an instance of the maximal matching operator. Furthermore, the property states that all the binomial coefficients of all the strings obtained by the instantiation of the expansion of the maximal matching operator are equal to one. Then the maximal matching operator, namely the encoding of the term variable X , does not influence the rate of such a rule, and this is exactly what we expected.

5.2.3 Example

As an example we show the encoding and some steps of evolution of a simple CLS+ model where T represents the initial state of the computation and $\{R_1, R_2\}$ is the set of CLS+ rules.

$$T \equiv ((a)^L \rfloor (a)) \mid ((b)^L \rfloor (b))$$

$$(R_1) \quad a \mid b \mapsto c$$

$$(R_2) \quad ((a)^L \rfloor (X)) \mid ((b)^L \rfloor (Y)) \mapsto (a \mid b)^L \rfloor (X \mid Y)$$

The term T represents two cells: one with membrane a and containing the sequence a , the other with membrane b and containing the sequence b . Brane rule R_1 states that a parallel composition of sequences a and b is rewritten into a sequence c . Non-brane rule R_2 describes the fusion of a cell with membrane a with a neighbor cell with membrane b ; the resulting cell will have both a and b on the membrane and its content will be the merge of the content of both cells.

A possible CLS+ evolution for T is

$$\begin{aligned} T &\equiv ((a)^L \rfloor (a)) \mid ((b)^L \rfloor (b)) \\ &\rightarrow (a \mid b)^L \rfloor (a \mid b) \quad \text{applying rule } (R_2) \\ &\rightarrow (a \mid b)^L \rfloor (c) \quad \text{applying rule } (R_1) \\ &\rightarrow (c)^L \rfloor (c) \quad \text{applying rule } (R_1) \end{aligned}$$

As previously defined, we have that T is encoded into the multiset

$$M \equiv \bar{\lambda}_1 \cdot a \mid \lambda_1 \cdot a \mid \bar{\lambda}_2 \cdot b \mid \lambda_2 \cdot b$$

and rules are encoded into the following rules

- (1) $\Delta \cdot a \mid \Delta \cdot b \mapsto \Delta \cdot c$
- (2) $\{\{\Delta \cdot \lambda_w \cdot \bar{\lambda}_x\}_a \mid \{\{\Delta \cdot \lambda_w \cdot \lambda_x\}_X \mid \{\{\Delta \cdot \lambda_w \cdot \bar{\lambda}_y\}_b \mid \{\{\Delta \cdot \lambda_w \cdot \lambda_y\}_Y\}\}\} \\ \mapsto \{\{\Delta \cdot \lambda_w \cdot \bar{\lambda}_z\}_a \mid \{\{\Delta \cdot \lambda_w \cdot \bar{\lambda}_z\}_b \mid \{\{\Delta \cdot \lambda_w \cdot \lambda_z\}_X \mid \{\{\Delta \cdot \lambda_w \cdot \lambda_z\}_Y\}\}\}$
- (3) $\{\{\bar{\lambda}_x\}_a \mid \{\{\lambda_x\}_X \mid \{\{\bar{\lambda}_y\}_b \mid \{\{\lambda_y\}_Y\}\}\} \mapsto \{\{\bar{\lambda}_z\}_a \mid \{\{\bar{\lambda}_z\}_b \mid \{\{\lambda_z\}_X \mid \{\{\lambda_z\}_Y\}\}\}$

where (1) is the result of encoding the brane rule R_1 and rules (2) and (3) are the result of encoding the non-brane rule R_2 . We note that using the maximal matching operator

in the encoding of looping operators leads to the fact that rule (2) will not be applicable to the encoding of a term like $(a \mid b)^L \rfloor \dots$

A SMSR computation corresponding to the shown CLS+ computation is

$$\begin{aligned}
T &\equiv \bar{\lambda}_1 \cdot a \mid \lambda_1 \cdot a \mid \bar{\lambda}_2 \cdot b \mid \lambda_2 \cdot b \\
&\rightarrow \bar{\lambda}_3 \cdot a \mid \bar{\lambda}_3 \cdot b \mid \lambda_3 \cdot a \mid \lambda_3 \cdot b \quad \text{applying rule (3)} \\
&\rightarrow \bar{\lambda}_3 \cdot a \mid \bar{\lambda}_3 \cdot b \mid \lambda_3 \cdot c \quad \text{applying rule (1)} \\
&\rightarrow \bar{\lambda}_3 \cdot c \mid \lambda_3 \cdot c \quad \text{applying rule (1)}
\end{aligned}$$

5.2.4 Correctness and Completeness of the Encoding

In this section we prove the equivalence of the semantics of SMSR and CLS+ with respect to the encoding previously defined.

We start with showing how SMSR instantiation functions $\sigma_{[\cdot]}$ can be obtained from a CLS instantiation function σ through the encoding function $\llbracket _ \rrbracket$. This can be done as follows:

$$\begin{aligned}
&\forall v \in SV \cup EV \Rightarrow \sigma_{[\cdot]}(v) = \llbracket \sigma(v) \rrbracket \\
&\forall v \in TV \cup BV. \llbracket \sigma(v) \rrbracket = S_1 \mid \dots \mid S_n \Rightarrow \forall i = 1, \dots, n. \sigma_{[\cdot]}(v_i) = S_i
\end{aligned}$$

Note that for a function σ there exist infinite $\sigma_{[\cdot]}$ that satisfy the given construction.

We then prove the following lemma that will be used in the proof of equivalence.

Lemma 5.18. *For any CLS+ pattern P and any instantiation function $\sigma \in \Sigma$ a function ρ exists such that $\llbracket P\sigma \rrbracket \equiv \langle \llbracket P \rrbracket \rangle_{\rho\sigma_{[\cdot]}}$.*

Proof. The proof is made by structural induction on CLS+ patterns.

- if $P \equiv SP$, $P \equiv \tilde{x}$ or $P \equiv x$ for any ρ the proof is trivial because encoding and expansion are the identity on P .
- If $P \equiv v \in TV \cup BV$ we have to prove that $\llbracket \sigma(v) \rrbracket \equiv \langle \llbracket v \rrbracket \rangle_{\rho\sigma_{[\cdot]}}$. Let $\llbracket \sigma(v) \rrbracket = S_1 \mid \dots \mid S_n$ then given a ρ such that $\rho(v) = n$, we have that $\langle \llbracket v \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \equiv \langle \{\epsilon\}_v \rangle_{\rho\sigma_{[\cdot]}} \equiv \sigma_{[\cdot]}(v_1) \mid \dots \mid \sigma_{[\cdot]}(v_n)$. The proof follows trivially from the definition of $\sigma_{[\cdot]}$.
- If $P \equiv P_1 \mid P_2$ for P_1 and P_2 patterns, we prove that $\llbracket (P_1 \mid P_2)\sigma \rrbracket \equiv \langle \llbracket P_1 \mid P_2 \rrbracket \rangle_{\rho\sigma_{[\cdot]}}$. As instantiation σ , encoding function $\llbracket _ \rrbracket$ and patterns expansion function distribute over $_ \mid _$ we have that $\llbracket (P_1 \mid P_2)\sigma \rrbracket \equiv \llbracket P_1\sigma \rrbracket \mid \llbracket P_2\sigma \rrbracket \equiv \llbracket P_1\sigma \rrbracket \mid \llbracket P_2\sigma \rrbracket$ and that $\langle \llbracket P_1 \mid P_2 \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \equiv \langle \llbracket P_1 \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \mid \langle \llbracket P_2 \rrbracket \rangle_{\rho\sigma_{[\cdot]}}$; assuming inductive hypothesis on P_1 and P_2 yields the proof.
- If $P \equiv (BP)^L \rfloor P$ we prove that $\llbracket ((BP)^L \rfloor P)\sigma \rrbracket \equiv \langle \llbracket (BP)^L \rfloor P \rrbracket \rangle_{\rho\sigma_{[\cdot]}}$. We have that $\llbracket ((BP)^L \rfloor P)\sigma \rrbracket \equiv \llbracket (BP\sigma)^L \rfloor P\sigma \rrbracket \equiv \bar{\lambda}_i \triangleright \llbracket BP\sigma \rrbracket \mid \lambda_i \triangleright \llbracket P\sigma \rrbracket$ and that $\langle \llbracket (BP)^L \rfloor P \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \equiv \langle \bar{\lambda}_i \triangleright \langle \llbracket BP \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \mid \lambda_i \triangleright \langle \llbracket P \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \rangle_{\rho\sigma_{[\cdot]}}$. The definition of $\langle _ \triangleright _ \rangle$ is similar to the one of $\llbracket _ \rrbracket$, hence it could be easily proved that the lemma holds also when $\llbracket P \rrbracket$ is replaced by $\langle P \rangle$. By the definition of $_ \triangleright _$ and $\langle _ \triangleright _ \rangle_{\rho}$ we can write $\langle \bar{\lambda}_i \triangleright \langle \llbracket BP \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \mid \lambda_i \triangleright \langle \llbracket P \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \rangle_{\rho\sigma_{[\cdot]}} \equiv \bar{\lambda}_i \triangleright \langle \langle \llbracket BP \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \rangle_{\rho\sigma_{[\cdot]}} \mid \lambda_i \triangleright \langle \langle \llbracket P \rrbracket \rangle_{\rho\sigma_{[\cdot]}} \rangle_{\rho\sigma_{[\cdot]}}$ and the proof follows from inductive hypothesis on BP and P .

□

Finally, we prove the correspondence between the semantics of CLS+ and SMSR.

Theorem 5.19. *For any CLS+ terms T and T'*

$$T \rightarrow T' \Leftrightarrow \exists \xi, \zeta. \lfloor T \rfloor \xrightarrow{\xi, \zeta} \lfloor T' \rfloor$$

Proof. The proof of *correctness* (\Rightarrow) is made by induction on the rules of the semantics of CLS+.

- We prove that $P_1\sigma \rightarrow P_2\sigma \Rightarrow \lfloor P_1\sigma \rfloor \xrightarrow{\xi, \zeta} \lfloor P_2\sigma \rfloor$ using rule (1) of SMSR. We assume $P_1\sigma \rightarrow P_2\sigma$; as (P_1, P_2) is a CLS+ rule, then $(\Delta \triangleright \llbracket P_1 \rrbracket, \Delta \triangleright \llbracket P_2 \rrbracket)$ is its encoding in SMSR. Let $\llbracket P_1 \rrbracket \equiv M_1$, $\llbracket P_2 \rrbracket \equiv M_2$, $\lfloor P_1\sigma \rfloor \equiv M$ and $\lfloor P_2\sigma \rfloor \equiv M'$. The SMSR instantiation function $\sigma_{\llbracket \cdot \rrbracket}$ that we need is one out of the infinite that can be built with respect to the σ of CLS+ and is such that $\sigma_{\llbracket \cdot \rrbracket}(\Delta) = \epsilon$ and it also has to satisfy the constraints imposed by SMSR rule (1). Notice that infiniteness of SMSR alphabet guarantees the existence of a $\sigma_{\llbracket \cdot \rrbracket}$ satisfying such constraints. As regards the function ρ we need, its existence is proved by Lemma 5.18.
- We prove that $T \mid T_1 \rightarrow T \mid T_2 \Rightarrow \lfloor T \mid T_1 \rfloor \xrightarrow{\xi, \zeta} \lfloor T \mid T_2 \rfloor$. Since $\lfloor _ \rfloor$ distributes over $_ \mid _$ rule (2) of SMSR can be applied where $M'' \equiv \lfloor T \rfloor$, $M \equiv \lfloor T_1 \rfloor$ and $M' \equiv \lfloor T_2 \rfloor$. By inductive hypothesis $T_1 \rightarrow T_2 \Rightarrow \lfloor T_1 \rfloor \xrightarrow{\xi, \zeta} \lfloor T_2 \rfloor$ constraints in (2) are satisfied because either T_1 does not contain term, brane variables or looping operators hence ξ is empty, or, by the restrictions on the term and brane variables and by the encoding of patterns containing looping operators, we have that all the prefixes in ξ are different from any other prefix in M'' . Finally, as regards constraints on ζ we have that, as in the previous case of the proof, there exist a $\sigma_{\llbracket \cdot \rrbracket}$ such that it generates fresh names.
- We prove that $(B)^L \rfloor T_1 \rightarrow (B)^L \rfloor T_2 \Rightarrow \lfloor (B)^L \rfloor T_1 \rfloor \xrightarrow{\xi, \zeta} \lfloor (B)^L \rfloor T_2 \rfloor$. Since $\lfloor _ \rfloor$ distributes over $(_)^L \rfloor _$ rule (2) of SMSR can be applied where $M'' \equiv \lfloor B \rfloor$, $M \equiv \lfloor T_1 \rfloor$ and $M' \equiv \lfloor T_2 \rfloor$. By inductive hypothesis $T_1 \rightarrow T_2 \Rightarrow \lfloor T_1 \rfloor \xrightarrow{\xi, \zeta} \lfloor T_2 \rfloor$ we can notice that constraints on ξ are satisfied. This because, by the definition of the encoding function, every string in M'' has a prefix $\bar{\lambda}_i$ that is different from any prefix λ_i in the encoding of T_1 . As before, constraints on ζ are satisfied by a proper choice of a $\sigma_{\llbracket \cdot \rrbracket}$.
- We prove that $(B_1)^L \rfloor T \rightarrow (B_2)^L \rfloor T \Rightarrow \lfloor (B_1)^L \rfloor T \rfloor \xrightarrow{\xi, \zeta} \lfloor (B_2)^L \rfloor T \rfloor$. The proof of $B_1 \rightarrow_{\mathcal{B}} B_2 \Rightarrow \lfloor B_1 \rfloor \xrightarrow{\xi, \zeta} \lfloor B_2 \rfloor$ is given by the first two cases of the proof of correctness.

In order to prove the *completeness* (\Leftarrow) we define the inverse of the encoding function $\lfloor _ \rfloor$. Let us denote the language of all the strings over the set $\{\lambda_i \mid i \in \mathbb{N}\} \cup \{\bar{\lambda}_i \mid i \in \mathbb{N}\} \cup \{\epsilon\}$ as $\mathcal{C}_{\mathcal{S}}$, the inversion of $\lfloor _ \rfloor$ is defined as follows

$$\begin{aligned} & \left[\epsilon \cdot S_1 \mid \dots \mid \epsilon \cdot S_{n_1} \mid \bar{\lambda}_j \cdot S'_1 \mid \dots \mid \bar{\lambda}_j \cdot S'_{n_2} \mid \lambda_j \cdot C_1 \cdot S''_1 \mid \dots \mid \lambda_j \cdot C_{n_3} \cdot S''_{n_3} \right]^{-1} = \\ & S_1 \mid \dots \mid S_{n_1} \mid (S'_1 \mid \dots \mid S'_{n_2})^L \mid (C_1 \cdot S''_1 \mid \dots \mid C_{n_3} \cdot S''_{n_3})^{-1} \end{aligned}$$

where $C_i \in \mathcal{C}_S$, $S_i, S'_i, S''_i \notin \mathcal{C}_S$, $j \in \mathbb{N}$ and $n_1, n_2, n_3 \geq 0$.

We assume also the inversion of the encoding functions $\llbracket _ \rrbracket$ and $\langle _ \rangle$. We can now prove completeness by induction on the rules of the semantics of SMSR.

- We prove that $\exists \xi, \zeta. M \xrightarrow{\xi, \zeta} M' \Rightarrow \llbracket M \rrbracket^{-1} \rightarrow \llbracket M' \rrbracket^{-1}$. We assume that (M_1, M_2) is a SMSR rule and we have that its corresponding CLS+ rule (P_1, P_2) is the one such that $\sigma_{\llbracket _ \rrbracket}(\Delta) \triangleright \llbracket P_1 \rrbracket \equiv M$ and $\sigma_{\llbracket _ \rrbracket}(\Delta) \triangleright \llbracket P_2 \rrbracket \equiv M'$. Notice that the value $\sigma_{\llbracket _ \rrbracket}(\Delta)$ represents the encoding of the CLS+ context in which the rule is applied; in other words this corresponds to a series of applications of the rules of the CLS+ semantics that chooses the point of application of the rule (P_1, P_2) inside the term representing the state of the computation. As regards the instantiation function for CLS+ we can notice that, since we have the one for SMSR, we can built an ad-hoc function σ as follows: $\forall v \in EV \cup SV. \sigma(v) = \sigma_{\llbracket _ \rrbracket}(v)$ and $\forall V \in TV \cup BV. \sigma_{\llbracket _ \rrbracket}(V_1) = S_1, \dots, \sigma_{\llbracket _ \rrbracket}(V_n) = S_n \Rightarrow \sigma(V) = \llbracket S_1 \mid \dots \mid S_n \rrbracket^{-1}$.
- We prove that $\exists \xi, \zeta. M'' \mid M \xrightarrow{\xi, \zeta} M'' \mid M' \Rightarrow \llbracket M'' \mid M \rrbracket^{-1} \rightarrow \llbracket M'' \mid M' \rrbracket^{-1}$. By inductive hypothesis $M \xrightarrow{\xi, \zeta} M' \Rightarrow \llbracket M \rrbracket^{-1} \rightarrow \llbracket M' \rrbracket^{-1}$; let $\llbracket M \rrbracket^{-1} \equiv T_1$ and $\llbracket M' \rrbracket^{-1} \equiv T_2$. The decoding distributes over \mid , and hence $\llbracket M'' \mid M \rrbracket^{-1} \equiv \llbracket M'' \rrbracket^{-1} \mid \llbracket M \rrbracket^{-1}$ and $\llbracket M'' \mid M' \rrbracket^{-1} \equiv \llbracket M'' \rrbracket^{-1} \mid \llbracket M' \rrbracket^{-1}$. We have the proof with $\llbracket M'' \rrbracket^{-1} \equiv T$.

□

Chapter 6

Conclusions

In this thesis we formally defined an intermediate language for the description and the simulation of biological systems. We propose to use the language for the encoding of higher level formalisms; here with the term high we refer to their ability of describing biological systems at different levels of abstraction.

We started presenting a very simple language, namely the MultiSet Rewriting formalism (MSR), for the simulation of mainly biochemical systems. This language is based on multisets of atomic objects which are modified by the application of rewriting rules; rewriting rules simply rewrite multisets. We examined the expressiveness of this formalism and we proved its non Turing completeness; this motivated us to define a more expressive language as higher level languages are Turing complete.

We proposed the String MultiSet Rewriting formalism (SMSR) as an intermediate language; SMSR is a natural extension of MSR. The choice of MSR as a basis for SMSR has two main motivations: the simple and clear syntax of MSR together with the possibility of easily developing a simulator for this formalism.

SMSR is based on multisets of strings that are modified, as in MSR, by the application of rewriting rules. As regards the implementation of SMSR, there exist many efficient algorithms for manipulating string that we could use to implement an efficient simulator based on SMSR.

The main features of SMSR are three: the possibility of defining rewrite rules with variables, the possibility of generating fresh data in the process of application of the rules and the maximal matching operator. In particular this operator can be very useful when higher level languages are encoded, accordingly to a well-known encoding technique, into SMSR.

We discussed such a technique with many examples, in particular we gave intuitions on the encoding of P-Systems, an established formalism for the description of biological systems based on membranes and contextual rules. We also encoded the Calculus of Looping Sequences in one of its variants, namely the CLS+ formalism, into SMSR by using such a technique. We formally defined the required encoding functions for terms, patterns and rules of CLS+; finally, we gave a proof of correctness and completeness of the defined encoding.

Furthermore, for both MSR and SMSR, we examined the possibility of adding a stochastic semantics to the formalisms; in particular we noticed that, although in higher level languages expressing a stochastic semantics may not be trivial, in the case of multisets

rewriting formalisms, it is easy.

As future work, as we mentioned before, we will develop a simulator based on SMSR; such a simulator will use plugins to support the dynamic encoding of higher level languages as CLS+, P-Systems, and so on. Furthermore, we will formally define the encoding of other formalism into SMSR in order to be able to use the related simulator to simulate the encoding of all these languages.

Bibliography

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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).
- [5] 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.
- [6] 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.
- [7] R. Barbuti, G. Caravagna, A. Maggiolo-Schettini, and P. Milazzo “An Intermediate Language for the Simulation of Biological Systems”. 1th Int. Workshop From Biology to Concurrency and Back (FBTC’07), ENTCS, to appear.
- [8] *Caenorhabditis elegans* WWW Server. web site. <http://elegans.swmed.edu/>.
- [9] G. Caravagna. “Progettazione e Realizzazione di un Simulatore di Sistemi Biologici”. BSc Thesis, University of Pisa, 2005.
- [10] L. Cardelli. “Brane Calculi. Interactions of Biological Membranes”. CMSB’04, LNCS 3082, pages 257–280, Springer, 2005.
- [11] L. Cardelli and A.D. Gordon. “Mobile Ambients”. Theoretical Computer Science, volume 240, number 1, pages 177–213, 2000.

- [12] I. Cervesato. “The Logical Meeting Point of Multiset Rewriting and Process Algebra: Progress Report”. Technical Memo CHACS-5540-153, Center for High Assurance Computer Systems, Naval Research Laboratory, Washington, DC, 2004.
- [13] I. Cervesato, N.A. Durgin, P. Lincoln, J.C. Mitchell and A. Scedrov “A Meta-Notation for Protocol Analysis”. 12th Computer Security Foundations Workshop, pp. 55-69, IEEE Computer Society Press, Mordano, Italy, 1999.
- [14] 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.
- [15] 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.
- [16] W.Damm and D. Harel. “LSCs: Breathing Life into Message Sequence Charts”. Formal Methods in System Design, volume 19, number 1, 2001.
- [17] Z. Dang and O.H. Ibarra. “On P Systems Operating in Sequential and Limited Parallel Modes”, Workshop on Descriptive Complexity of Formal Systems, pages 164–177, 2004.
- [18] V. Danos and C. Laneve. “Formal Molecular Biology”. Theoretical Computer Science, volume 325, number 1, pages 69–110, 2004.
- [19] V. Danos and S. Pradaliere. “Projective Brane Calculus”, Computational Methods in Systems Biology (CMSB’04), LNCS 3082, pages 134–148, Springer, 2005.
- [20] P. Degano and C. Priami. “Enhanced Operational Semantics: A Tool for Describing and Analyzing Concurrent Systems”. ACM Computing Surveys **33** (2001), 135–176.
- [21] 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.
- [22] R. Ewald, M. John, and A. Uhrmacher. “A Spatial Extension to the Pi Calculus” 1th Int. Workshop From Biology to Concurrency and Back (FBTC’07), ENTCS, to appear.
- [23] GBS web site: <http://www.di.unipi.it/~milazzo/biosims>.
- [24] D. Gillespie. “Exact Stochastic Simulation of Coupled Chemical Reactions”. Journal of Physical Chemistry, volume 81, pages 2340–2361, 1977.
- [25] D. Harel. “Statecharts: A Visual Formalism for Complex Systems”. Science of Computer Programming, volume 8, number 3, pages 231–274, 1987.
- [26] 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.

- [27] 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.
- [28] 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.
- [29] 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.
- [30] F. Martinelli, S. Bistarelli, I. Cervesato, G. Lenzini G. and R. Marangoni. “Representing Biological Systems Through Multiset Rewriting”. In Computer Aided Systems Theory (EUROCAST’03), LNCS 2809, pages 415–426, Springer, 2004.
- [31] 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.
- [32] P. Milazzo. “Qualitative and Quantitative Formal Modeling of Biological Systems”. PhD Thesis, University of Pisa, 2007.
- [33] R. Milner. “Communicating and Mobile Systems: the π –Calculus”. Cambridge University Press, 1999.
- [34] G. Pardini. “Topological Calculus of Looping Sequences”. MSc Thesis, University of Pisa, 2007
- [35] G. Păun. “Computing with Membranes”. Journal of Computer and System Sciences, volume 61, number 1, pages 108–143, 2000.
- [36] G. Păun. “Membrane Computing. An Introduction”. Springer, 2002.
- [37] G. Păun, G. Rozenberg. “A Guide to Membrane Computing”. Theoretical Computer Science, volume 287, number 1, pages 73–100, 2002.
- [38] C. Priami. “Stochastic π –Calculus”. The Computer Journal, volume 38, number 7, pages 578–589, 1995.
- [39] C. Priami and P. Quaglia “Beta Binders for Biological Interactions”. CMSB’04, LNCS 3082, pages 20–33, Springer, 2005.
- [40] 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.
- [41] 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.

- [42] A. Regev and E. Shapiro. “Cells as Computation”. *Nature*, volume 419, page 343, 2002.
- [43] 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.
- [44] G. Scatena. “Development of a Stochastic Simulator for Biological Systems Based on the CLS”. MSc Thesis, University of Pisa, 2007
- [45] The SPiM web page: <http://research.microsoft.com/~aphillip/spim/>.
- [46] Stochastic CLS Machine web page: <http://www.di.unipi.it/~milazzo/biosims/>.
- [47] The P Systems web page: <http://psystems.disco.unimib.it/>.
- [48] C. Versari. “A Core Calculus for a Comparative Analysis of Bio-inspired Calculi”. *European Symposium on Programming (ESOP 2007)*
- [49] C. Versari. “Encoding Catalytic P systems in $\pi@$ ”. *Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC 2006)*.