

**MEMBRANE SYSTEMS WITH  
LIMITED PARALLELISM**

by

Bianca Daniela Popa, B.S., M.S.

A Dissertation Presented in Partial Fulfillment  
of the Requirements for the Degree  
Doctor of Philosophy

COLLEGE OF ENGINEERING AND SCIENCE  
LOUISIANA TECH UNIVERSITY

November 2006

## ABSTRACT

Membrane computing is an emerging research field that belongs to the more general area of molecular computing, which deals with computational models inspired from bio-molecular processes. Membrane computing aims at defining models, called membrane systems or P systems, which abstract the functioning and structure of the cell. A membrane system consists of a hierarchical arrangement of membranes delimiting regions, which represent various compartments of a cell, and with each region containing bio-chemical elements of various types and having associated evolution rules, which represent bio-chemical processes taking place inside the cell.

This work is a continuation of the investigations aiming to bridge membrane computing (where in a compartmental cell-like structure the chemicals to evolve are placed in compartments defined by membranes) and brane calculi (where one considers again a compartmental cell-like structure with the chemicals/proteins placed on the membranes themselves). We use objects both in compartments and on membranes (the latter are called proteins), with the objects from membranes evolving under the control of the proteins. Several possibilities are considered (objects only moved across membranes or also changed during this operation, with the proteins only assisting the move/change or also changing themselves). Somewhat expected, computational universality is obtained for several combinations of such possibilities.

We also present a method for solving the **NP**-complete SAT problem using P systems with proteins on membranes. The SAT problem is solved in  $O(nm)$  time, where  $n$  is the number of boolean variables and  $m$  is the number of clauses for an instance written in conjunctive normal form. Thus, we can say that the solution for each given instance is obtained in linear time. We succeeded in solving SAT by a uniform construction of a deterministic P system which uses rules involving objects in regions, proteins on membranes, and membrane division.

Then, we investigate the computational power of P systems with proteins on membranes in some particular cases: when only one protein is placed on a membrane, when the systems have a minimal number of rules, when the computation evolves in accepting or computing mode, etc.

This dissertation introduces also another new variant of membrane systems that uses context-free rewriting rules for the evolution of objects placed inside compartments of a cell, and symport rules for communication between membranes. The strings circulate across membranes depending on their membership to regular languages given by means of regular expressions. We prove that these rewriting-symport P systems generate all recursively enumerable languages. We investigate the computational power of these newly introduced P systems for three particular forms of the regular expressions that are used by the symport rules. A characterization of ETOL languages is obtained in this context.

## TABLE OF CONTENTS

ABSTRACT . . . . .	ii
LIST OF TABLES . . . . .	vii
LIST OF FIGURES . . . . .	viii
ACKNOWLEDGEMENTS . . . . .	ix
1 INTRODUCTION . . . . .	1
1.1 An Informal View on Membrane Computing . . . . .	1
1.2 A Presentation of This Dissertation . . . . .	4
2 PREREQUISITES . . . . .	7
2.1 Elements of Languages, Automata, Complexity . . . . .	7
2.1.1 Languages . . . . .	7
2.1.2 Chomsky Grammars . . . . .	9
2.1.3 Lindenmayer Systems . . . . .	12
2.1.4 Automata and Register Machines . . . . .	14
2.1.5 Matrix Grammars . . . . .	19
2.1.6 Elements of Complexity . . . . .	24
2.2 Basic Classes of P Systems . . . . .	26
2.3 Some Computability and Efficiency Results . . . . .	37
3 APPLICATIONS OF MEMBRANE COMPUTING . . . . .	39

3.1 Motivation . . . . .	39
3.2 P Systems as Modeling Tools in Biology . . . . .	40
3.2.1 Mechanosensitive Channels . . . . .	41
3.2.2 Biological Dynamics . . . . .	43
3.2.3 Cell-Mediated Immunity . . . . .	50
3.2.4 P53 Signaling Pathways . . . . .	52
3.2.5 EGFR Signaling Cascade . . . . .	54
3.2.6 Quorum Sensing in Bacteria . . . . .	56
3.2.7 Respiration and Photosynthesis in Cyanobacteria . . . . .	58
3.2.8 Photosynthesis . . . . .	64
3.3 Applications in Computer Science . . . . .	67
3.3.1 Static Sorting P Systems . . . . .	68
3.3.2 Sub-LP Systems Used In Computer Graphics . . . . .	72
3.3.3 An Analysis of a Public-Key Protocol with Membranes . . . . .	75
3.3.4 Membrane Algorithms . . . . .	78
3.3.5 Computationally Hard Problems . . . . .	81
4 OPERATIONS FROM BRANE CALCULI . . . . .	85
4.1 Mate/Drip Operations in P Systems . . . . .	85
4.2 Computing Power . . . . .	88
5 P SYSTEMS WITH PROTEINS ON MEMBRANES . . . . .	96
5.1 The Model . . . . .	96
5.2 Computational Results for the Generating Mode . . . . .	102
5.2.1 Universality for One Type of Rules . . . . .	105

5.2.2 Using Two Types of Rules . . . . .	111
5.3 One Protein Case . . . . .	120
5.4 Accepting and Computing Systems . . . . .	124
5.5 A Small Universal P System . . . . .	126
5.6 Solving SAT in Polynomial Time . . . . .	131
5.7 Using a More Restrictive Type of Rules . . . . .	141
6 REWRITING P SYSTEMS WITH SYMPORT RULES . . . . .	149
6.1 The Model . . . . .	149
6.2 Computational Results . . . . .	152
7 CONCLUSIONS, OPEN PROBLEMS AND FURTHER RESEARCH . . .	165
BIBLIOGRAPHY . . . . .	167

## LIST OF TABLES

Table 4.1	Theorem 4.1 (simulating a matrix of type 2). . . . .	90
Table 4.2	Theorem 4.1 (simulating a matrix of type 3). . . . .	90
Table 4.3	Theorem 4.3 (simulating a matrix of type 2). . . . .	92
Table 4.4	Theorem 4.4 (simulating a matrix of type 2). . . . .	94
Table 4.5	Theorem 4.4 (simulating a matrix of type 3). . . . .	95
Table 4.6	Theorem 4.4 (simulating the end of computation). . . . .	95
Table 5.1	Restricted rules. . . . .	99
Table 5.2	Change protein rules. . . . .	100
Table 5.3	Theorem 5.2 (simulating a SUB instruction). . . . .	106
Table 5.4	Theorem 5.3 (simulating an ADD instruction). . . . .	109
Table 5.5	Theorem 5.3 (simulating a SUB instruction). . . . .	110
Table 5.6	Theorem 5.4 (simulating an ADD instruction). . . . .	112
Table 5.7	Theorem 5.4 (simulating a SUB instruction). . . . .	113
Table 5.8	Theorem 5.5 (simulating an ADD instruction). . . . .	115
Table 5.9	Theorem 5.5 (simulating a SUB instruction). . . . .	115
Table 5.10	Theorem 5.6 (simulating an ADD instruction). . . . .	118
Table 5.11	Theorem 5.6 (simulating a SUB instruction). . . . .	119
Table 5.12	Theorem 5.8 (simulating the input). . . . .	126
Table 5.13	The simulation of $l_5 : (\text{ADD}(5), l_6)$ and $l_6 : (\text{SUB}(7), l_7, l_8)$ . . . . .	131
Table 7.1	Universality results from [56]. . . . .	166

## LIST OF FIGURES

Figure 2.1	Inclusions for classes of problems. . . . .	25
Figure 5.1	Relationships between the types of rules. . . . .	103
Figure 5.2	The universal register machine $U_{22}$ . . . . .	129
Figure 5.3	The instructions of the universal register machine $U_{22}$ . . . . .	130

## ACKNOWLEDGEMENTS

First I would like to thank my advisor Dr. Andrei Păun for providing me the opportunity to work under his guidance, and for his constant interest, support, and advice throughout my PhD program.

I would also like to express my gratitude to my Louisiana Tech University professors for their patience and help, especially to Dr. Vir Phoha, Dr. Ben Choi, Dr. Weizhong Dai, Dr. Raja Nassar, Dr. Richard Greechie, Dr. Galen Turner, and Dr. B. Ramu Ramachandran.

This work was possible thanks to the financial support of research grants, in part by LA BoR RSC grant LEQSF (2004-07)-RD-A-23 and NSF Grants IMR-0414903 and CCF-0523572.

# CHAPTER 1

## INTRODUCTION

### 1.1 An Informal View on Membrane Computing

The main goal of this dissertation is to contribute to the bridging of *membrane computing* and *brane calculi* by introducing and studying new classes of membrane systems inspired from brane calculi.

Membrane computing is an area of natural computing initiated in 1998 in the technical report [78], and its main goal is to abstract computing ideas and to construct models based on the structure and functioning of living cells, as well as from the way cells are organized in tissues, organs, and organisms. Models in this area are called *membrane systems* or *P systems*, and three basic classes are considered: *cell-like* (inspired from the cell structure), *tissue-like* (inspired from the organization of cells in tissues), and *neural-like* (related to the way neurons are linked in neural nets).

Briefly, a membrane system is a distributed and parallel computing model processing multisets of objects in the compartments of a cell-like hierarchical arrangement of membranes (hence a structure of compartments which corresponds to a rooted tree), or in a tissue-like structure consisting of cells placed in the nodes of an arbitrary graph. Both the membranes and the objects of the membranes evolve according to

some rules. For instance, the multisets of objects may evolve by means of rewriting rules, which have the form of usual chemical equations (several objects react and get transformed into some product objects). A crucial aspect of this processing is the resulting communication of objects through membranes, between regions of the same cell, between cells, or between cells and their environment.

Since the initiation of membrane computing, many classes of P systems have been introduced (based on the form of rules and the way they are used), inspired from biological reality and motivated from mathematical or computer science points of view. Most of them are able to compute all Turing computable sets of natural numbers, showing that the cell is a powerful and efficient computing device. A comprehensive presentation of the field can be found in [79]; for a complete overview of membrane computing, see the P systems website at: <http://psystems.disco.unimib.it/>.

In February 2003, the Institute for Scientific Information (ISI) has mentioned [78] as a “fast breaking paper” in computer science (see <http://esi-topics.com>, February 2003); in October 2003, the domain itself was qualified by ISI as “emergent research front” in computer science, with the paper [75] mentioned as a “citation leader” in membrane computing.

In turn, several brane calculi were introduced in [23], where the emphasis is on operations involving membranes (e.g., mate, drip, pinocytosis, exocytosis, etc.) and controlled by proteins placed on membranes. The approach is based on process algebra techniques and has as the main goal to produce a biologically realistic model. The two directions of research are thus complementary from several points of view, hence combining their ingredients is a natural research idea (already explored, from

points of view related to but different from those we considered here, e.g., in [24], [19], etc.).

The topics dealt with in this dissertation can be placed in a broader perspective, first, in the framework of natural computing, and then in the framework of cell modeling.

In terms of natural computing, membrane computing is one of the main areas of bio-inspired computing areas, closely related to DNA computing, but also to older areas of natural computing, such as evolutionary computing and neural computing.

While theoretical developments in DNA computing can be traced back to 1987 [50], the birth of DNA computing is considered in the year 1994, when Leonard Adleman reported his successful experiment of computing in a test tube [1]. He used fragments of DNA to compute the solution to a complex graph theory problem, the Traveling Salesman Problem. Adleman's method utilizes sequences of DNA's molecular subunits to represent vertices of a graph. Thus, combinations of these sequences formed randomly by the massively parallel action of biochemical reactions in test tubes described random paths through the graph. Using the tools of biochemistry, Adleman was able to extract the correct answer to the problem out of the many random paths represented by the product DNA strands. A drawback of Adleman's DNA computer is that it requires human assistance. The goal of the DNA computing field is to create a device that can work independent of human involvement, and one possible idea – also related to the birth of membrane computing – is to place the computation in the natural environment where DNA evolves, in the cell.

In turn, there are two main approaches in modeling the living cell, the “tra-

ditional” one, mainly using differential equations (hence continuous mathematics), and several recent approaches, based on discrete mathematics combined with computational techniques. Membrane computing is perfectly suited from this point of view, because it uses tools coming from automata and language theory (hence discrete mathematics). Mainly, local processes were addressed by means of differential equations, with an intensive effort made in the last years towards obtaining models of the whole cell. In membrane computing, the whole cell is represented from the very beginning, but the models were not introduced with aims related to biological research but with computer science purposes. In addition to these two directions, we need to mention also the Petri nets (tool for describing and studying systems that are characterized as being concurrent, asynchronous, distributed, parallel, non-deterministic, and/or stochastic) and  $\pi$ -calculus (processes describing concurrent computations whose configuration can change during the computation).

The thesis presents both a series of applications of membrane computing (in particular, in modeling the cell), and several models and results of the natural computing type, hence covering both frameworks mentioned above.

## 1.2 A Presentation of This Dissertation

We start with a section of prerequisites, recalling notions and results from automata, formal language theory, and complexity theory used in the thesis. Then, we give the basic definition of a P system and the three main classes of membrane systems, together with the relevant computational and efficiency results known in this area.

In Chapter 3, we discuss the applications of membrane computing models in the two main directions, biology and computer science, and also about new areas of interest, such as linguistics and economics.

Chapter 4 gives the first results related to P systems using rules from brane calculi and evolution rules based only on proteins placed on membranes.

In Chapter 5, we consider a new class of P systems with objects placed in both compartments and on membranes, with the evolution of the former being controlled by the latter, which can also evolve, either by membrane operations or, directly, by object evolution rules. Several possibilities are considered (objects only moved across membranes or also changed during this operation, with the proteins only assisting the move/change or also changing themselves). Somewhat expected, computational universality is obtained for several combinations of such possibilities. We also investigate here some particular cases: P systems with only one protein on a membrane, P systems with minimal number of rules, P systems working in accepting and computing mode, etc. As an application of P systems with proteins on membranes, we give a solution to the Satisfiability problem in polynomial time, using exponential space.

In Chapter 6, we combine two central ideas used in membrane computing. Namely, we work with string objects processed by context-free rewriting rules (such systems are known not to be universal [79]), without target indications associated with the rules, and we move the strings across membranes by means of symport rules. The passage of strings across membranes is controlled by means of regular expressions which define languages of strings able to move in/out a region of a system. This is reminiscent of the EC (evolution-communication) P systems from [26] and [57], and

can also be related to the brane calculi – membrane computing bridging, in the sense that one combines the evolution of objects (described by strings this time) from the compartments of a membrane structure with the control imposed by the proteins placed on membranes.

The contents of the last four chapters are original and most of the results reported here were published in papers presented at specialized conferences or in international journals.

The dissertation ends with a short chapter indicating a series of research topics and open problems, and a comprehensive bibliography.

## CHAPTER 2

### PREREQUISITES

#### 2.1 Elements of Languages, Automata, Complexity

We recall here several notions and results from automata and formal language theory, as well as from complexity theory; for further details, we refer to some of the monographs in this area, such as [67], [86], [87].

We use standard set theory notations:  $\emptyset$  is the empty set, the fact that an element  $a$  belongs to a set  $M$  is denoted by  $a \in M$ , the inclusion of a set  $M_1$  in a set  $M_2$  is written  $M_1 \subseteq M_2$ , and the strict inclusion is indicated by  $M_1 \subset M_2$ ; the union, intersection, difference, and Cartesian product of two sets  $M_1$  and  $M_2$  are written as  $M_1 \cup M_2$ ,  $M_1 \cap M_2$ ,  $M_1 - M_2$ , and  $M_1 \times M_2$ , respectively. The set of natural numbers,  $\{0, 1, 2, \dots\}$ , is denoted by  $\mathbf{N}$ ; the set of  $n$ -dimensional vectors of natural numbers is denoted by  $\mathbf{N}^n$ , for  $n \geq 1$ . The cardinality (the number of elements) of a finite set  $M$  is denoted by  $card(M)$ .

##### 2.1.1 Languages

An *alphabet* is a finite non-empty set of abstract symbols. For an alphabet  $V$ , the free monoid (all finite strings of zero or more elements from  $V$ ) generated by  $V$  under the operation of concatenation is denoted by  $V^*$ , and the identity element of this monoid is the empty string, denoted by  $\lambda$ . The set of non-empty strings over  $V$ ,

that is  $V^* - \{\lambda\}$ , is denoted by  $V^+$ . Each subset of  $V^*$  is called a *language* over  $V$ . A language which does not contain the empty string (hence it is a subset of  $V^+$ ) is said to be  $\lambda$ -free. A set of languages is usually called a *family* of languages. If  $V = \{a\}$ , then we write  $a^*$  instead of  $V^*$ .

The *length* of a string  $x \in V^*$  (the number of occurrences in  $x$  of symbols from  $V$ ) is denoted by  $|x|$ . The number of occurrences of a symbol  $a \in V$  in  $x \in V^*$  is denoted by  $|x|_a$ . For a language  $L \subseteq V^*$ , the set  $length(L) = \{|x| \mid x \in L\}$  is called the *length set* of  $L$ . If  $FL$  is a family of languages, then we denote by  $NFL$  the family of length sets of languages in  $FL$ .

The *Parikh vector* associated with a string  $x \in V^*$  with respect to the alphabet  $V = \{a_1, a_2, \dots, a_n\}$  is  $\Psi_V(x) = (|x|_{a_1}, |x|_{a_2}, \dots, |x|_{a_n})$  (note that the ordering of the symbols from  $V$  is relevant);  $\Psi_V : V^* \rightarrow \mathbf{N}^n$  is also called the *Parikh mapping* associated with  $V$ . The *Parikh image* (or Parikh set) of a language  $L \subseteq V^*$  is defined by  $\Psi_V(L) = \{\Psi_V(x) \mid x \in L\}$ . For a family of languages  $FL$ , we denote by  $PsFL$  the family of Parikh images of the languages in  $FL$ .

A mapping  $h : V \rightarrow U^*$ , extended to  $h : V^* \rightarrow U^*$  by  $h(\lambda) = \{\lambda\}$ , with  $h(x_1x_2) = h(x_1)h(x_2)$ , for  $x_1, x_2 \in V^*$ , is called a *morphism*. If  $h(a) \neq \lambda$ , for each  $a \in V$ , then  $h$  is a  $\lambda$ -free morphism. A morphism  $h : V^* \rightarrow U^*$  is called a *coding* if  $h(a) \in U$ , for each  $a \in V$ , and a *weak coding* if  $h(a) \in U \cup \{\lambda\}$ , for each  $a \in V$ . Thus, a coding only renames the symbols, while a weak coding can also erase some of them.

The languages  $L$  are sets; hence, we can perform the usual set operations: union, intersection, difference, and complementation (with respect to  $V^*$ , if  $L \subseteq V^*$ ). There are also several operations which are specific to languages as sets of strings

(hence specific to strings).

The *concatenation* of two languages  $L_1$  and  $L_2$  is  $L_1L_2 = \{xy \mid x \in L_1, y \in L_2\}$ .

This operation can be iterated: take by convention  $L^0 = \{\lambda\}$ ; then, for all  $i \geq 0$ , we define  $L^{i+1} = LL^i$ . The union of all languages  $L^i$ , for  $i \geq 0$ , is denoted by  $L^*$  and is called the *Kleene closure* of  $L$ .

A language that can be obtained from symbols of an alphabet  $V$  and from  $\lambda$  by using finitely many times the operations of union, concatenation, and Kleene closure is called *regular*; also, the empty language is considered to be regular.

### 2.1.2 Chomsky Grammars

Generally speaking, a *grammar* is a finite device *generating* in a well-specified sense the strings of a language. The Chomsky grammars are particular cases of *rewriting systems*, where the operation used in processing the strings is the rewriting (the replacement of a substring of the processed string by another substring), and they have been introduced as a possible model of the syntax of natural languages (and developed mainly in relation with the syntax of programming languages).

**Definition 2.1** A *Chomsky grammar* (or a *type-0 grammar*) is a quadruple  $G = (N, T, S, R)$ , where  $N$  and  $T$  are disjoint alphabets,  $S \in N$ , and  $R$  is a finite set of *rewriting rules* (we also say *productions*) of the form  $u \rightarrow v$ , with  $u, v \in (N \cup T)^*$  and  $u$  contains at least one non-terminal symbol. The alphabet  $N$  is called the *non-terminal alphabet*,  $T$  is the *terminal alphabet*, and  $S$  is the *axiom*.

Let  $V = N \cup T$  and  $R$  having rules in the form  $r : u \rightarrow v$ , for  $u, v \in V^*$ . For a string  $w = w_1uw_2$ , we can rewrite  $u$  by means of  $v$  by using the rule  $r$ , and we get

the string  $z = w_1vw_2$ . This operation is called *direct derivation* and it is denoted by  $w \Rightarrow_r z$ , or simply  $w \Rightarrow z$  if  $r$  is understood. The reflexive and transitive closure of direct derivation is denoted by  $\Rightarrow^*$ . Thus, we have  $w \Rightarrow^* z$  if either  $w = z$ , or  $w \Rightarrow z_1 \Rightarrow z_2 \Rightarrow \dots \Rightarrow z_k = z$ , for some  $z_1, \dots, z_k \in V^*$ ,  $k \geq 1$ . Each string  $w \in V^*$  such that  $S \Rightarrow_G^* w$  is called a *sentential form*. The language  $L(G)$  generated by  $G$  is defined by

$$L(G) = \{x \in T^* \mid S \Rightarrow_G^* x\}.$$

Two grammars  $G_1$  and  $G_2$  are called *equivalent* if  $L(G_1) - \{\lambda\} = L(G_2) - \{\lambda\}$  (the two languages coincide modulo the empty string). Therefore, when comparing the languages generated or accepted by two devices, we ignore the empty string.

According to the form of their rules, the Chomsky grammars are classified as follows. A grammar  $G = (N, T, S, R)$  is called:

- *length-increasing* (or *type-1*), if for all  $u \rightarrow v \in R$  we have  $|u| \leq |v|$ ;
- *context-sensitive*, if each  $u \rightarrow v \in R$  has  $u = u_1Au_2, v = u_1xu_2$ , for  $u_1, u_2 \in (N \cup T)^*$ ,  $A \in N$ , and  $x \in (N \cup T)^+$ ;
- *context-free* (or *type-2*), if each production  $u \rightarrow v \in R$  has  $u \in N$ ;
- *linear*, if each rule  $u \rightarrow v \in R$  has  $u \in N$  and  $v \in T^* \cup T^*NT^*$ ;
- *right-linear*, if each rule  $u \rightarrow v \in R$  has  $u \in N$  and  $v \in T^* \cup T^*N$ ;
- *left-linear*, if each rule  $u \rightarrow v \in R$  has  $u \in N$  and  $v \in T^* \cup NT^*$ ;
- *regular* (or *type-3*), if each rule  $u \rightarrow v \in R$  has  $u \in N$  and  $v \in T \cup TN$ .

The family of languages generated by length-increasing grammars is equal to the family of languages generated by context-sensitive grammars; the families of languages generated by right-linear or by left-linear grammars coincide and they are equal to the family of languages generated by regular grammars, as well as to the family of regular languages.

We denote by  $RE$ ,  $CS$ ,  $CF$ ,  $LIN$ , and  $REG$  the families of languages generated by arbitrary (or *recursively enumerable*), context-sensitive (hence also by length-increasing), context-free, linear, and regular (hence also by right-linear and left-linear) grammars, respectively. By  $FIN$  we denote the family of finite languages.

The following strict inclusions hold:

$$FIN \subset REG \subset LIN \subset CF \subset CS \subset RE.$$

This is called *the Chomsky hierarchy*, the constant reference for investigations related to the power of membrane systems (and of any new types of computing devices). This important role of Chomsky hierarchy is due to the fact that the family  $RE$  of languages generated by type-0 Chomsky grammars is exactly the family of languages which are recognized by Turing machines, and according to the Turing-Church thesis, this is the maximal level of algorithmic computability. The Chomsky hierarchy is well structured; hence, we have a detailed classification of computing devices (there are other types of grammars considered in literature, which are related to this hierarchy).

Because  $RE$  is the family of languages recognized (computed) by Turing machines,  $NRE$ , the family of length sets of languages in  $RE$ , is the family of sets of

numbers recognized (computed) by Turing machines. If we consider the length sets of languages, the Chomsky hierarchy becomes

$$NFIN \subset NREG = NLIN = NCF \subset NCS \subset NRE,$$

because the length set of a context-free language can be obtained as the length set of a regular language. On the other hand, if we consider the Parikh sets of languages, the inclusions still hold:

$$PsFIN \subset PsREG = PsLIN = PsCF \subset PsCS \subset PsRE.$$

We continue by introducing a tool used in subsequent proofs, the Kuroda normal form for type-0 grammars.

**Definition 2.2** A type-0 grammar  $G = (N, T, S, R)$  is said to be in *Kuroda normal form* if the rules in  $R$  are of the forms  $A \rightarrow BC$ ,  $A \rightarrow a$ ,  $A \rightarrow \lambda$ , and  $AB \rightarrow CD$ , where  $A, B, C, D \in N$  and  $a \in T$ .

### 2.1.3 Lindenmayer Systems

*Lindenmayer systems* (or L systems), which were introduced in 1968 by Aristid Lindenmayer, are rewriting systems which model the development of multicellular organisms (for more details we refer to [62],[86]). The main difference with Chomsky grammars is the parallelism: in a derivation step one rewrites all symbols of the string.

**Definition 2.3** A *zero-interactions Lindenmayer system* (or 0L system) is a construct  $G = (V, w, R)$ , where  $V$  is an alphabet,  $w \in V^*$  is the axiom, and  $R$  is a finite

set of rules of the form  $a \rightarrow v$ , with  $a \in V, v \in V^*$ , such that for each  $a \in V$  there is at least one rule  $a \rightarrow v$  in  $R$  (we say that  $R$  is *complete*).

For  $w_1, w_2 \in V^*$ , we write  $w_1 \Longrightarrow w_2$  if  $w_1 = a_1 \dots a_n$ ,  $w_2 = v_1 \dots v_n$ , with  $a_i \rightarrow v_i \in R$ , for  $1 \leq i \leq n$ . The language generated by  $G$  is

$$L(G) = \{x \in V^* \mid w \Longrightarrow^* x\}.$$

If we have  $v \neq \lambda$ , for each rule  $a \rightarrow v \in R$ , then we say that  $G$  is *propagating* (non-erasing); if for each  $a \in V$  there is only one rule  $a \rightarrow v$  in  $R$ , then  $G$  is said to be *deterministic*. If we distinguish a subset  $T$  of  $V$  and we define the generated language as  $L(G) = \{x \in T^* \mid w \Longrightarrow^* x\}$ , then we say that  $G = (V, T, w, R)$  is *extended*. The family of languages generated by 0L systems is also denoted by 0L; we add the letters  $P, D, E$  in front of 0L if propagating, deterministic, or extended 0L systems are used, respectively.

**Definition 2.4** A *tabled 0L system*, abbreviated *T0L*, is a system  $G = (V, w, R_1, \dots, R_n)$ , such that each triple  $(V, w, R_i)$ , for  $1 \leq i \leq n$ , is a 0L system; each  $R_i$  is called a *table*, for  $1 \leq i \leq n$ .

The language generated by  $G$  is defined by

$$L(G) = \{x \in V^* \mid w \Longrightarrow_{R_{j_1}} w_1 \Longrightarrow_{R_{j_2}} \dots \Longrightarrow_{R_{j_m}} w_m = x, \\ m \geq 0, 1 \leq j_i \leq n, 1 \leq i \leq m\}.$$

(Each derivation step is performed by the rules of the same table.)

A *T0L system* is deterministic/propagating/extended when each of its tables is deterministic/propagating/extended.

The family of languages generated by  $T0L$  systems is also denoted by  $T0L$ ; the families  $ET0L$ ,  $EDT0L$ , etc., are obtained in the same way as  $E0L$ ,  $ED0L$ , etc., by adding the letters  $P$ ,  $D$ ,  $E$  in front of  $T0L$ .

The  $D0L$  family is incomparable with  $FIN$ ,  $REG$ ,  $LIN$ ,  $CF$ , whereas  $E0L$  strictly includes the family  $CF$ ;  $ET0L$  is the largest family of Lindenmayer languages (with 0-interactions); it is strictly included in  $CS$ , and it is closed under union, intersection with regular languages, arbitrary morphisms, concatenation, and Kleene closure.

For each language  $L \in ET0L$ ,  $L \subseteq T^*$ , there is a very useful *normal form* of  $ET0L$  systems with only two tables,  $G = (V, T, w, R_1, R_2)$ , such that  $L = L(G)$ .

**Definition 2.5** An  $ET0L$  system  $G = (V, T, w, R_1, R_2)$  is in the *two-table normal form* if it has only two tables, i.e.,  $(V, T, w, R_1)$  and  $(V, T, w, R_2)$  are  $E0L$  systems.

#### 2.1.4 Automata and Register Machines

The five basic families of languages in the Chomsky hierarchy,  $REG$ ,  $LIN$ ,  $CF$ ,  $CS$ ,  $RE$ , are also characterized by (recognizing) automata. These automata are: the finite automaton, the one-turn pushdown automaton, the pushdown automaton, the linearly bounded automaton, and the Turing machine, respectively. We present here only two of these devices, those which are more relevant for this dissertation: finite automata and Turing machines.

**Definition 2.6** A *non-deterministic finite automaton* is a construct  $A = (Q, V, s_0, F, \delta)$ , where  $Q$  and  $V$  are disjoint alphabets,  $s_0 \in Q$ ,  $F \subseteq Q$ , and  $\delta : Q \times V \rightarrow \mathcal{P}(Q)$ ;  $Q$  is the set of states,  $V$  is the alphabet of the automaton,

$s_0$  is the initial state,  $F$  is the set of final states, and  $\delta$  is the transition mapping.

If  $\text{card}(\delta(s, a)) \leq 1$ , for all  $s \in Q$ ,  $a \in V$ , then we say that the automaton is *deterministic* (in this case,  $\delta$  is a function from  $Q \times V$  to  $Q$ ). A relation  $\vdash$  is defined on the set  $Q \times V^*$  in the following way: for  $s, s' \in Q$ ,  $a \in V$ ,  $x \in V^*$ , we write  $(s, ax) \vdash (s', x)$  if  $s' \in \delta(s, a)$ ; by definition,  $(s, \lambda) \vdash (s, \lambda)$ . If  $\vdash^*$  is the reflexive and transitive closure of the relation  $\vdash$ , then the language of the strings recognized by automaton  $A$  is defined by

$$L(A) = \{x \in V^* \mid (s_0, x) \vdash^* (s, \lambda), s \in F\}.$$

**Definition 2.7** A *Turing machine* is a construct  $M = (Q, V, T, \flat, s_0, F, \delta)$ , where  $Q$  and  $V$  are disjoint alphabets (the set of states and the tape alphabet),  $T \subseteq V$  (the input alphabet),  $\flat \in V - T$  (the blank symbol),  $s_0 \in Q$  (the initial state),  $F \subseteq Q$  (the set of final states), and  $\delta$  is a partial mapping from  $Q \times V$  to the power set of  $Q \times V \times \{L, R\}$  (the move mapping; if  $(s', b, d) \in \delta(s, a)$ , for  $s, s' \in Q$ ,  $a, b \in V$ , and  $d \in \{L, R\}$ , then the machine reads the symbol  $a$  in state  $s$  and passes to state  $s'$ , replaces  $a$  with  $b$ , and moves the read-write head to the left when  $d = L$  and to the right when  $d = R$ ).

If  $\text{card}(\delta(s, a)) \leq 1$ , for all  $s \in Q$ ,  $a \in V$ , then  $M$  is said to be *deterministic*.

A configuration of a Turing machine as above is a string  $xsy$ , where  $x \in V^*$ ,  $y \in V^*(V - \{\flat\}) \cup \{\lambda\}$ , and  $s \in Q$ . In this way we identify the contents of the tape, the state, and the position of the read-write head: it scans the first symbol of  $y$ . Observe that the blank symbol may appear in  $x$ ,  $y$ , but not in the last position

of  $y$ ; both  $x$  and  $y$  may be empty. We denote by  $ID_M$  the set of all instantaneous descriptions of  $M$ .

On the set  $ID_M$ , one defines the *direct transition* relation  $\vdash_M$  as follows:

$$xsay \vdash_M xbs'y \text{ iff } (s', b, R) \in \delta(s, a),$$

$$xs \vdash_M xbs' \text{ iff } (s', b, R) \in \delta(s, b),$$

$$xcsay \vdash_M xs'cby \text{ iff } (s', b, L) \in \delta(s, a),$$

$$xcs \vdash_M xs'cb \text{ iff } (s', b, L) \in \delta(s, b),$$

where  $x, y \in V^*$ ,  $a, b, c \in V$ ,  $s, s' \in Q$ .

The language recognized by a Turing machine  $M$  is defined by

$$L(M) = \{w \in T^* \mid s_0w \vdash_M^* xsy \text{ for some } s \in F, x, y \in V^*\}.$$

(This is the language of all strings such that the machine reaches a final state when starting to work in the initial state by scanning the first symbol of the string.)

The family of recursively enumerable languages is characterized as follows:

$$RE = \{L \mid \text{there exists a Turing machine } M \text{ such that } L = L(M)\}.$$

The difference between a finite automaton and a Turing machine is visible only in their functioning: the Turing machine can move its head in both directions and it can rewrite the scanned symbol, possibly erasing it (replacing it with the blank symbol).

Consider an alphabet  $T$  and a Turing machine  $M = (K, V, T, b, s_0, F, \delta)$ . As we have seen above,  $M$  starts working with a string  $w$  written on its tape and reaches or does not reach a final state (and then halts), depending on whether or not  $w \in L(M)$ .

A Turing machine can be codified as a string of symbols over a suitable alphabet. Let us denote such a string by  $code(M)$ . Imagine a Turing machine  $M_u$  which starts its work from a string which contains both  $w \in T^*$  and  $code(M)$ , for a given Turing machine  $M$ , and stops in a final state if and only if  $w \in L(M)$ . Such a machine  $M_u$  is called *universal*. It can simulate any given Turing machine, providing that a code of the particular machine is written on the tape of the universal one, together with a string as input for the particular machine.

Note the important distinction between *computational completeness* and *universality*. Given a class  $\mathcal{C}$  of computability models, we say that  $\mathcal{C}$  is *computationally complete* if the devices in  $\mathcal{C}$  can characterize the power of Turing machines (or of any other type of equivalent computing devices). This means that given a Turing machine  $M$ , we can find an element  $C$  in  $\mathcal{C}$  such that  $C$  is equivalent to  $M$ . Thus, completeness refers to the capacity of covering the level of computability of Turing machines (in grammatical terms, this means to generate all recursively enumerable languages). *Universality* is an internal property of  $\mathcal{C}$  and it means the existence of a fixed element of  $\mathcal{C}$  which is able to simulate any given element of  $\mathcal{C}$  in the way described above for Turing machines.

Because in the membrane computing area computational completeness always implies universality, we will say “computational universality” (or simply “universality”) when having a result which is stated as a computational completeness result.

In the proofs from the next sections, we will use register machines as one of the devices characterizing *NRE*, hence the Turing computability.

Informally speaking, a register machine consists of a specified number of registers (counters) which can hold any natural number, and which are handled according to a program consisting of labeled instructions; the registers can be increased or decreased by 1 – the decreasing being possible only if a register holds a number greater than or equal to 1 (we say that it is non-empty) – and checked whether they are non-empty.

**Definition 2.8** A (non-deterministic) *register machine* is a device  $M = (m, B, l_0, l_h, R)$ , where  $m \geq 1$  is the number of counters,  $B$  is the (finite) set of instruction labels,  $l_0$  is the initial label,  $l_h$  is the halting label, and  $R$  is the finite set of instructions labeled (hence uniquely identified) by elements from  $B$  ( $R$  is also called the *program* of the machine). The labeled instructions are of the following forms:

- $l_1 : (\text{ADD}(r), l_2, l_3)$ , for  $1 \leq r \leq m$  (add 1 to register  $r$  and go non-deterministically to one of the instructions with labels  $l_2, l_3$ );
- $l_1 : (\text{SUB}(r), l_2, l_3)$ , for  $1 \leq r \leq m$  (if register  $r$  is not empty, then subtract 1 from it and go to the instruction with label  $l_2$ ; otherwise, go to the instruction with label  $l_3$ );
- $l_h : \text{HALT}$  (the halting instruction, which can only have the label  $l_h$ ).

We say that a register machine has no ADD instructions looping to the same label (or *without direct loops*) if there are no instructions of the form  $l_1 : (\text{ADD}(r), l_1, l_2)$  or  $l_1 : (\text{ADD}(r), l_2, l_1)$  in  $R$ . For instance, an instruction of the form  $l_1 : (\text{ADD}(r), l_1, l_2)$  can be replaced by the following instructions, where  $l_3, l_4, l_5, l_6$  are new labels:

$l_1 : (\text{ADD}(r), l_3, l_4)$ ,  $l_3 : (\text{ADD}(r), l_5, l_5)$ ,  $l_4 : (\text{ADD}(r), l_6, l_6)$ ,  $l_5 : (\text{ADD}(r), l_1, l_1)$ ,  $l_6 : (\text{ADD}(r), l_2, l_2)$ . The generated set of numbers is not changed.

A register machine generates a natural number in the following manner: it starts computing with all  $m$  registers being empty, with the instruction labeled by  $l_0$ ; if the computation reaches the instruction  $l_h : \text{HALT}$  (we say that it halts), then the value of register 1 is the number generated by the computation. The set of numbers computed by  $M$  in this way is denoted by  $N(M)$ . It is known (see [67]) that non-deterministic register machines with three registers generate exactly the family  $NRE$ , of Turing computable sets of numbers. Moreover, without loss of generality, we may assume that in the halting configuration, all registers, except the first one where the result of the computation is stored, are empty.

If the addition instruction has the form  $l_1 : (\text{ADD}(r), l_2)$ , meaning that the value of register  $r$  is increased by 1 and the computation goes to instruction  $l_2$ , then the register machine is called *deterministic*.

### 2.1.5 Matrix Grammars

In membrane computing area, context-free grammars with regulated rewriting, such as matrix grammars, are very useful devices [39].

**Definition 2.9** A context-free *matrix grammar* (without appearance checking) is a construct  $G = (N, T, S, M)$ , where  $N$  and  $T$  are disjoint alphabets (of non-terminals and terminals, respectively),  $S \in N$  (axiom), and  $M$  is a finite set of *matrices*, which are sequences of the form  $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$ , for  $n \geq 1$ , of context-free rules over  $N \cup T$ .

For a string  $x$ , a matrix  $m : (r_1, \dots, r_n)$  is executed by applying the productions  $r_1, \dots, r_n$  one after the other, following the order in which they appear in the matrix.

Formally, we write  $w \Longrightarrow_m z$  if there is a matrix  $m : (A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$  in  $M$  and the strings  $w_1, w_2, \dots, w_{n+1} \in (N \cup T)^*$  such that  $w_1 = w$ ,  $w_{n+1} = z$ , and for all  $1 \leq i \leq n$  we have  $w_i = w'_i A_i w''_i$ ,  $w_{i+1} = w'_i x_i w''_i$ . If the matrix  $m$  is understood, then we write  $\Longrightarrow$  instead of  $\Longrightarrow_m$ . As usual, the reflexive and transitive closure of this relation is denoted by  $\Longrightarrow^*$ . Then, the language generated by  $G$  is

$$L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}.$$

The family of languages generated by context-free matrix grammars is denoted by  $MAT$ .

The following results on matrix languages are known:

1.  $CF \subset MAT \subset RE$ , and  $NCF = NMAT \subset NRE$ ,  $PsCF \subset PsMAT \subset PsRE$ ;
2. The family  $MAT$  is closed under union, concatenation, intersection with regular languages, morphisms, right and left derivatives, mirror image, and permutation;
3. Each language  $L \in MAT$ ,  $L \subseteq a^*$ , is regular.

In a matrix grammar as above, without appearance checking, when using a matrix, all rules are used in the order imposed by the matrix. A powerful extension is provided by the possibility to skip certain rules, and this leads to the following definition.

**Definition 2.10** A construct  $G = (N, T, S, M, F)$  is a matrix grammar *with appearance checking* if  $(N, T, S, M)$  is a matrix grammar as in Definition 2.9 and  $F$  is a set of occurrences of rules in the matrices of  $M$ .

For  $w, z \in (N \cup T)^*$ , we write  $w \implies z$  if there is a matrix  $(A_1 \rightarrow x_1, \dots, A_n \rightarrow x_n)$  in  $M$  and the strings  $w_i \in (N \cup T)^*$ , for  $1 \leq i \leq n + 1$ , such that  $w_1 = w$ ,  $w_{n+1} = z$ , and for all  $1 \leq i \leq n$ , either (1)  $w_i = w'_i A_i w''_i$ ,  $w_{i+1} = w'_i x_i w''_i$ , for some  $w'_i, w''_i \in (N \cup T)^*$ , or (2)  $w_i = w_{i+1}$ , if  $A_i$  does not appear in  $w_i$ , and the rule  $A_i \rightarrow x_i$  appears in  $F$ .

Therefore, the difference between a matrix grammar with appearance checking and one without appearance checking is the fact that in the former case we have at our disposal the set  $F$ , consisting of *occurrences* of rules in the matrices of  $M$  (that is, if the same rule, say  $A \rightarrow x$ , appears several times in the matrices, only some of these occurrences can be present in  $F$ ; we may interpret them as having some “flags” which distinguish them from the other rules); the rules of a matrix are applied in order, as usual, possibly skipping the rules in  $F$  if they cannot be applied. Thus, if a rule not in  $F$  is met, then it has to be used. If a rule from  $F$  is met, then we have two cases: if it can be applied, then it must be applied; if it cannot be applied, then the rule may be skipped. That is why the rules from  $F$  are said to be applied in the appearance checking mode. The information given by such an application of a rule from  $F$  is rather useful because in the case of skipping the rule, we get “negative information”: a certain symbol is not present in the string.

The language generated by a matrix grammar with appearance checking  $G$  is

defined by

$$L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}.$$

The family of languages of this form is denoted by  $MAT_{ac}$ .

The following results are known:

$$CF \subset MAT \subset MAT_{ac} = RE$$

(which implies  $NMAT_{ac} = NRE$  and  $PsMAT_{ac} = PsRE$ ).

Note that in the previous definitions of matrix grammars we allow erasing rules; in general, there is a difference between using or not erasing rules, but we do not consider this distinction here.

**Definition 2.11** A matrix grammar (with appearance checking)  $G = (N, T, S, M, F)$  is said to be in the *binary normal form* if  $N = N_1 \cup N_2 \cup \{S, \#\}$ , with these three sets mutually disjoint, and the matrices in  $M$  are in one of the following forms:

1.  $(S \rightarrow XA)$ , where  $X \in N_1, A \in N_2$ ;
2.  $(X \rightarrow Y, A \rightarrow x)$ , where  $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$ ;
3.  $(X \rightarrow Y, A \rightarrow \#)$ , where  $X, Y \in N_1, A \in N_2$ ;
4.  $(X \rightarrow \lambda, A \rightarrow x)$ , where  $X \in N_1, A \in N_2, x \in T^*, |x| \leq 2$ .

Moreover, there is only one matrix of type 1 and  $F$  consists exactly of all rules  $A \rightarrow \#$  appearing in matrices of type 3;  $\#$  is a trap symbol – once introduced, it is never removed. A matrix of type 4 is used only once – in the last step of a derivation.

In the proofs from the subsequent chapters, a matrix grammar with appearance checking in the binary normal form is always given as  $G = (N, T, S, M, F)$ , with  $N = N_1 \cup N_2 \cup \{S, \#\}$ , and with  $n + 1$  matrices in  $M$ , injectively labeled with  $m_0, m_1, \dots, m_n$ ; the matrix  $m_0 : (S \rightarrow X_{init}A_{init})$  is the initial one, with  $X_{init}$  a given symbol from  $N_1$  and  $A_{init}$  a given symbol from  $N_2$ ; the next  $k$  matrices are without appearance checking rules,  $m_i : (X \rightarrow \alpha, A \rightarrow x)$ , for  $1 \leq i \leq k$ , where  $X \in N_1, \alpha \in N_1 \cup \{\lambda\}$ , and  $A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$  (if  $\alpha = \lambda$ , then  $x \in T^*$ ); the last  $n - k$  matrices have rules to be applied in the appearance checking mode,  $m_i : (X \rightarrow Y, A \rightarrow \#)$ , for  $k + 1 \leq i \leq n$ , with  $X, Y \in N_1, A \in N_2$ .

The non-terminal matrices  $m_i$ , for  $1 \leq i \leq k$ , will be called *matrices of type 2*, the matrices  $m_i$ , for  $k + 1 \leq i \leq n$ , will be called *matrices of type 3*, and the terminal matrices  $m_i : (X \rightarrow \lambda, A \rightarrow x)$ , for  $1 \leq i \leq k$ , will be called *matrices of type 4*.

**Definition 2.12** A matrix grammar (with appearance checking)  $G = (N, T, S, M, F)$  is in the *Z-binary normal form* if  $N = N_1 \cup N_2 \cup \{S, Z, \#\}$ , with these three sets mutually disjoint, and the matrices in  $M$  are in one of the following forms:

1.  $(S \rightarrow XA)$ , where  $X \in N_1, A \in N_2$ ;
2.  $(X \rightarrow Y, A \rightarrow x)$ , where  $X, Y \in N_1, A \in N_2, x \in (N_2 \cup T)^*, |x| \leq 2$ ;
3.  $(X \rightarrow Y, A \rightarrow \#)$ , where  $X \in N_1, Y \in N_1 \cup \{Z\}, A \in N_2$ ;
4.  $(Z \rightarrow \lambda)$ .

Moreover, there is only one matrix of type 1,  $F$  consists exactly of all rules  $A \rightarrow \#$  appearing in matrices of type 3, and, if a sentential form generated by  $G$

contains the symbol  $Z$ , then it is of the form  $Zw$ , for some  $w \in (T \cup \{\#\})^*$  (that is, the appearance of  $Z$  makes sure that, except for  $Z$ , all symbols are terminal). As above,  $\#$  is a trap symbol, and the (unique) matrix of type 4 is used only once – in the last step of a derivation.

Usually, in the case of grammars in the  $Z$ -binary normal form, we have  $N = N_1 \cup N_2 \cup \{S, Z, \#\}$ , the matrices  $m_i$ , for  $k + 1 \leq i \leq n$ , can also be of the form  $m_i : (X \rightarrow Z, A \rightarrow \#)$ , and the only terminal matrix is  $m_{n+1} : (Z \rightarrow \lambda)$ .

### 2.1.6 Elements of Complexity

In this section, we introduce some notions from the theory of computational complexity ([45]), and we start with some basic definitions.

A problem  $X$  is *tractable* (or  $X \in \mathbf{P}$ ) if there exists an algorithm which solves  $X$  in polynomial time (its worst case time efficiency has an asymptotic upper bound that is polynomial of the problem input size). The class  $\mathbf{NP}$  contains the problems which can be solved by a non-deterministic algorithm in polynomial time.

Obviously, deterministic classes of problems are included in the non-deterministic classes. The problems from the class  $\mathbf{P}$  are considered computationally tractable, that is, one can afford to solve them in a feasible time, while the problems from  $\mathbf{NP}$  (which are not known to be in  $\mathbf{P}$ ) are considered intractable. All known deterministic algorithms for problems in  $\mathbf{NP} - \mathbf{P}$  are of an exponential time complexity, and the exponentials grow “too fast” in order to wait an exponential time for a solution. Thus, it is important to know whether a given problem lies in the class  $\mathbf{P}$  or not. For that, the problems can be *reduced* from one problem to another one. We say that problem  $B$  reduces to problem  $A$  if there is a transformation  $R$  which, for any

input  $x$  of  $B$ , produces an input  $R(x)$  to  $A$  such that the answer of  $A$  to the input  $R(x)$  is the answer of  $B$  to the input  $x$ . Thus, if  $B$  reduces to  $A$ , then in order to solve  $B$ , it is enough to compute  $R(x)$  and to solve  $A$  for the input  $R(x)$ . Of course, a reduction is acceptable if it is less complex than the problem  $B$  itself. With respect to classes  $\mathbf{P}$  and  $\mathbf{NP}$ , a reduction  $R$  is acceptable if it can be done in polynomial time. In such a case, when  $B$  reduces to  $A$ , we may say that  $A$  is at least as complex as  $B$ ; hence, it is important to know whether a problem  $A$  has the property that all problems from a given class can be reduced to  $A$  (we may say that  $A$  is among the hard problems of that class). For most natural complexity classes,  $\mathbf{P}$  and  $\mathbf{NP}$  included, there are problems such that all problems from that class can be reduced to them. Such problems are called *complete* for the respective classes. Of particular interest are the problems which are  $\mathbf{NP}$ -complete ( $\mathbf{NPC}$  class from Figure 2.1).

The inclusion  $\mathbf{P} \subseteq \mathbf{NP}$  is clear; whether or not this is an equality is probably the best known and the most important open problem in computer science today, the so-called  $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$  problem. As for now, the situation looks as in the following figure.

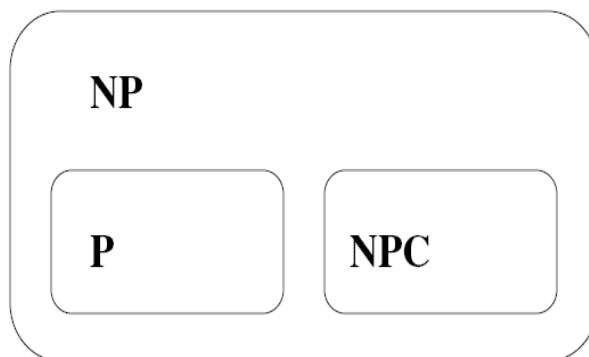


Figure 2.1 Inclusions for classes of problems.

## 2.2 Basic Classes of P Systems

The essential ingredient of a P system is its *membrane structure*, which is a hierarchical arrangement of membranes, as in a cell (hence described by a tree), or a net of membranes (placed in the nodes of a graph), as in a tissue or a neural net. The intuition behind the notion of a membrane is a three-dimensional vesicle from biology, but the concept itself is generalized to interpret a membrane as a separator of two regions, a finite “inside” and an infinite “outside,” providing the possibility of a selective communication between the two regions.

Following the real-life cell structure, where chemicals appear in different concentrations, and where the number of occurrences matters, membrane computing uses multisets as data structures (a multiset is a set with multiplicities associated with its elements). Each region contains a multiset of objects represented by symbols from a given alphabet. Usually, multisets are represented by strings where the order of symbols does not matter. In addition, each region has a set of rules applying to its objects or to the membrane itself (changing the structure of the system). The objects are considered to be identical; thus, the rules and the objects to be applied are non-deterministically chosen.

Rules are applied in a non-deterministic and maximally parallel way (by maximal we mean that at each step as many rules as possible are used; hence, no further rule can be applied to the objects present in the membranes). The rules to be used and the objects are chosen in a non-deterministic manner, and all compartments of the system structure evolve at the same time, synchronously. This way, transitions among system configurations are obtained. A sequence of configurations starting from

the initial one is called a computation, and a result may be associated only with a halting computation (ending in a halting configuration where no rule can be further applied). Because P systems are characterized by non-determinism, a successful computation provides a result that can be a set of numbers, a set of vectors (Parikh set) of numbers, a set of strings (a language), etc., depending on the way the result of a computation is defined.

Besides the maximally parallel way of applying the rules, several others were considered, such as sequential way (at each step only one rule is used in the whole system or in each region), bounded parallel way (at each step at least  $k$  or exactly  $k$  rules are used in the whole system or in each region), or minimally parallel way (at each step at least one rule is used in each region where rules can be applied).

For a given system  $\Pi$ , we denote by  $N(\Pi)$  the set of numbers computed by  $\Pi$ . When we consider the vector of multiplicities of objects from the output region, we write  $Ps(\Pi)$ . In turn, in the case where we take as (external) output the strings of objects leaving the system, we denote the language of these strings by  $L(\Pi)$ .

We specify here some notations which are already standard in membrane computing. The family of sets  $N(\Pi)$  of numbers generated by P systems of a specified type, working with symbol objects, having at most  $m$  membranes, and using ingredients from a given *list-of-features* is denoted by  $NOP_m(\textit{list-of-features})$ . If we compute sets of vectors, we write  $PsOP_m(\textit{list-of-features})$ . When string objects are used,  $N$  is replaced by  $L$  (from “languages”) and  $O$  by  $S$  (from “strings”), thus obtaining families  $LSP_m(\textit{list-of-features})$ .

A P system can be defined as an accepting or a generating device. By accept-

ing, we mean that the P system is able to determine whether a given number (placed in the input membrane in the form of the multiplicity of a given object) is part of a specific set, while a generating P system is one that generates a set of numbers. The two main topics investigated in this framework are the computing power of P systems (working in the generative or accepting modes) and their usefulness in solving hard decision problems (based on the maximally parallel way of applying the rules). In the case of computing functions or solving problems, we need deterministic P systems in order to get only one result, not a set.

In what follows, we describe the three basic types of P systems: cell-like, tissue-like, and neural-like.

The main component of a *cell-like P system* is its membrane structure, a hierarchically arranged set of membranes, which is contained in a distinct external membrane (called the skin membrane), and which can be represented by a rooted tree or a string of labeled matching brackets. Each membrane determines a compartment (also called region) delimited from the region containing it and from the membranes placed directly inside, if any exists.

Many types of rules have been considered in literature such as evolution rules (for instance multiset rewriting with catalysts in [42]), communication rules (e.g., symport/antiport in [75]), rules involving the membranes too (division, dissolution, etc.), or combinations of the previous types. The use of rules can be controlled by promoters, inhibitors, catalysts, priorities, permeability of membranes, etc.

In a *tissue-like P system*, the structure consists of several one-membrane cells placed in the nodes of an arbitrary graph. Certain cells can communicate by channels

(represented by edges of the graph) that are provided between them, but all cells can communicate through the environment. The communication among cells (usually by symport/antiport rules, [43]) can be done directly (in one step) or indirectly (through the environment: one cell sends out objects and other cells can take these objects in one of the next steps). The channels (also called synapses) can be given in advance or they can be dynamically established during the evolution of the system. To each synapse can be associated a state, and by applying a rule to a synapse, the state can be changed. The use of rules is sequential at the level of each synapse, but it is maximally parallel at the level of the system (all the synapses which can use a rule must do so). An even more general type of tissue-like P system, called population P system, was introduced in [15], where also the cells can evolve as the membranes in a cell-like P system (by division, dissolution, creation, etc.), and, in addition, the synapses are inherited.

The *neural-like P systems* are similar to tissue-like P systems in the way that the cells are placed in the nodes of a graph and they contain multisets of objects, but they communicate by signals (spikes) along synapses. In this case, the cells have associated states which can be changed according to the signals that are received during computation. There are three possible ways of applying the rules: minimal (a rule is chosen and applied once to the pair state–multiset of objects), parallel (a rule is chosen and is used in maximally parallel manner related to the multiset of objects), and maximal (all rules are applied in a maximally parallel manner related to the pair state–multiset of objects). For formal definitions and details, see [31], [79].

To define a P system, we have to specify the alphabet of objects (a finite non-empty set of abstract symbols identifying the objects), the membrane structure (a string of labeled matching brackets), the multisets of objects present in each region of the system, the sets of evolution rules associated with each region, and the indication about the way the output is defined. Next, we give the basic definitions for some common types of P systems, following [79].

**Definition 2.13** A *transition P system* (or a P system with multiset-rewriting rules) of degree  $m$  (with  $m \geq 1$ ) is a construct of the form

$$\Pi = (O, C, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m, i_o),$$

where:

1.  $O$  is the (finite and non-empty) alphabet of objects;
2.  $C \subset O$  is the set of catalysts;
3.  $\mu$  is a membrane structure, consisting of  $m$  membranes, labeled  $1, 2, \dots, m$ ;
4.  $w_1, w_2, \dots, w_m$  are strings over  $O$  representing the multisets of objects present in regions  $1, 2, \dots, m$  of the membrane structure;
5.  $R_1, R_2, \dots, R_m$  are finite sets of evolution rules associated with regions  $1, 2, \dots, m$  of the membrane structure;
6.  $i_o$  is either one of the labels  $1, 2, \dots, m$ , and the respective region is the output region of the system, or it is 0, and the result of a computation is collected in the

environment of the system ( usually,  $i_o$  is the label of an elementary membrane of  $\mu$ ).

The rules are of the form  $u \rightarrow v$  or  $u \rightarrow v\delta$ , with  $u \in O^+$  and  $v \in (O \times Tar)^*$ , where  $Tar = \{here, in, out\}$  and  $\delta$  represents the membrane dissolution. The rules can be cooperative (with  $u$  arbitrary), non-cooperative (with  $u \in O - C$ ), or catalytic (of the form  $ca \rightarrow cv$  or  $ca \rightarrow cv\delta$ , with  $a \in O - C, c \in C$ , and  $v \in ((O - C) \times Tar)^*$ ). Note that in the standard case, the catalysts never evolve and never change the region, but they only help the other objects to evolve.

In general, the membrane structure and the multisets of objects from its compartments identify a configuration of a P system. The initial configuration is given by specifying the membrane structure and the multisets of objects available in its compartments at the beginning of a computation, that is, by  $(\mu, w_1, \dots, w_m)$ . During the evolution of the system, by applying the rules, both the multisets of objects and the membrane structure can change.

A possible extension of this definition is to consider a *terminal* set of objects,  $T \subseteq O$ , and to count only the copies of objects from  $T$ , discarding the objects from  $O - T$  present in the output region.

The multiset rewriting rules correspond to reactions taking place in the cell inside the compartments. However, an important part of the cell activity is related to the passage of substances through membranes. The process by which two molecules pass together across a membrane (through a specific protein channel) is called *symport*; when the two molecules pass simultaneously through a protein channel, but

in opposite directions, the process is called *antiport*. In membrane computing, these operations are represented in the following way:  $(ab, in)$  or  $(ab, out)$  are symport rules, stating that objects  $a$  and  $b$  pass together through a membrane, entering in the former case and exiting in the latter case; similarly,  $(a, out; b, in)$  is an antiport rule, stating that  $a$  exits and, at the same time,  $b$  enters the membrane. Separately, neither  $a$  nor  $b$  can cross a membrane unless we have a rule of the form  $(a, in)$  or  $(a, out)$ , called *uniport* rule (or minimal symport). We can generalize these types of rules, by considering symport rules of the form  $(x, in)$  and  $(x, out)$ , and antiport rules of the form  $(z, out; w, in)$ , where  $x, z, w \in V^*$  are multisets of arbitrary size, over an alphabet  $V$ . The weight of the symport rule is equal to  $|x|$ , and the weight of the antiport rule is computed as  $\max(|z|, |w|)$ .

Now, such rules can be used in a P system instead of the target indications *here*, *in*, and *out*. If we consider multiset rewriting rules without target indications, as well as symport/antiport rules for communication of the objects between compartments, we obtain *evolution-communication* P systems, which were considered in [26]. If we use symport/antiport rules alone, we can compute using only communication, as in [18], [74].

**Definition 2.14** A *P system with symport/antiport rules* is a construct of the form

$$\Pi = (O, \mu, w_1, \dots, w_m, E, R_1, \dots, R_m, i_o),$$

where:

1.  $O$  is the alphabet of objects;

2.  $\mu$  is the membrane structure (of degree  $m \geq 1$ , with the membranes labeled  $1, 2, \dots, m$  in a one-to-one manner);
3.  $w_1, \dots, w_m$  are strings over  $O$  representing the multisets of objects present in the  $m$  compartments of  $\mu$  in the initial configuration of the system;
4.  $E \subseteq O$  is the (finite) set of objects supposed to appear in the environment in arbitrarily many copies;
5.  $R_1, \dots, R_m$  are the (finite) sets of symport/antiport rules associated with the  $m$  membranes of  $\mu$ ;
6.  $i_o$  is the label of an elementary membrane of  $\mu$ , which indicates the output region of the system.

The rules are used in the non-deterministic maximally parallel manner. The number (or the vector of multiplicities) of objects present in region  $i_o$  in the halting configuration is said to be computed by the system by means of that computation; the set of all numbers (or vectors of numbers) computed in this way by  $\Pi$  is denoted by  $N(\Pi)$  (by  $Ps(\Pi)$ , respectively).

We note here a new component of the system, the set  $E$  of objects which are present in the environment in arbitrarily many copies. Because objects are moved only across membranes and the computation starts with finite multisets of objects present in the system, we cannot increase the number of objects necessary for the computation (no object is created, destroyed, or changed – the conservation law can be applied) if we do not provide a supply of objects in the environment. Because

the environment is supposedly inexhaustible, the objects from  $E$  are inexhaustible; regardless of how many of them are brought into the system, arbitrarily many remain outside.

We pass now to presenting a class of P systems, which, together with the basic transition systems and the symport/antiport systems, is one of the three central types of cell-like P systems considered in membrane computing. This last type of P systems gives a very active role to the membranes (by division rules, dissolution rules, etc.) during the computation, [44].

**Definition 2.15** The *P systems with active membranes* are constructs of the form

$$\Pi = (O, H, \mu, w_1, \dots, w_m, R),$$

where:

1.  $m$  is the initial degree of the system ( $m \geq 1$ );
2.  $O$  is the alphabet of objects;
3.  $H$  is a finite set of labels for membranes;
4.  $\mu$  is the membrane structure, consisting of  $m$  membranes initially having neutral polarizations and labels from  $H$  (not necessarily in a one-to-one manner);
5.  $w_1, \dots, w_m$  are strings over  $O$ , describing the multisets of objects placed in the  $m$  regions of  $\mu$ ;
6.  $R$  is a finite set of developmental rules, of the following forms:

$$(a) [{}_h a \rightarrow v]_h^e, \text{ for } h \in H, e \in \{+, -, 0\}, a \in O, v \in O^*$$

(object evolution rules: associated with membranes and depending on the label and the charge of the membranes, but not directly involving the membranes);

$$(b) a[{}_h ]_h^{e_1} \rightarrow [{}_h b]_h^{e_2}, \text{ for } h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$$

(*in* communication rules: an object is introduced in the membrane, and possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

$$(c) [{}_h a]_h^{e_1} \rightarrow [{}_h ]_h^{e_2} b, \text{ for } h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$$

(*out* communication rules: an object is sent out of the membrane, and possibly modified during this process; also the polarization of the membrane can be modified, but not its label);

$$(d) [{}_h a]_h^e \rightarrow b, \text{ for } h \in H, e \in \{+, -, 0\}, a, b \in O$$

(dissolving rules: in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

$$(e) [{}_h a]_h^{e_1} \rightarrow [{}_h b]_h^{e_2} [{}_h c]_h^{e_3}, \text{ for } h \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$$

(division rules for elementary membranes: in reaction with an object, the membrane is divided into two membranes with the same label, and possibly of different polarizations; the object specified in the rule is replaced in the two new membranes possibly by new objects; the remaining objects are duplicated and may evolve in the same step by rules of type (a)).

Inside each membrane, the rules of type (a) are applied in the non-deterministic

maximally parallel manner, with each copy of an object used by only one rule of any type from (a) to (e). Each membrane can be involved in only one rule of types (b)–(e). The types of rules are chosen in a bottom-up way: first we use the rules of type (a), and then the rules of other types; in this way, in the case of dividing membranes, the result of using the rules of type (a) first is duplicated in the newly obtained membranes.

The set  $H$  of labels has been specified because it is possible to allow the change of membrane labels. For instance, a division rule can be of the more general form

$$(e') [_{h_1} a ]_{h_1}^{e_1} \rightarrow [_{h_2} b ]_{h_2}^{e_2} [_{h_3} c ]_{h_3}^{e_3}, \text{ for } h_1, h_2, h_3 \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O.$$

The change of labels can also be considered for rules of types (b) and (c). Moreover, we can consider the possibility of dividing membranes in more than two copies ( $d$ -division), or even of dividing non-elementary membranes (in such a case, all inner membranes are duplicated in the new copies of the membrane).

It is important to note that in the case of P systems with active membranes, the membrane structure evolves during the computation, not only by decreasing the number of membranes, due to dissolution operations, but also by increasing the number of membranes by division. This increase can be exponential in a linear number of steps: using a division rule successively  $n$  steps, due to the maximal parallelism, we get  $2^n$  copies of the same membrane. This is one of the most investigated ways of obtaining an exponential working space in order to trade time for space and solve computationally hard problems (**NP**-complete problems) in feasible time.

### 2.3 Some Computability and Efficiency Results

The initial goal of membrane computing was to define computability models inspired from the cell biology, and to examine their computing power in comparison with the standard models in computability theory, Turing machines and their restricted variants. As it turns out, most of the classes of P systems considered are equal in power to Turing machines. All classes of systems considered above, whether cell-like, tissue-like, or neural-like, with symbol objects or string objects, working in the generative or the accepting modes, with certain combinations of features, are known to be universal. The cell turns out to be a very powerful “computer,” both when standing alone and in tissues.

We mention here some recent results, which are relevant for our work (for proofs, see the mentioned papers):

- $NOP_1(cat_2) = NRE$  (transition P systems with one membrane and two catalysts are universal [42]);
- $NOP_3(sym_1, anti_1) = NOP_3(sym_2, anti_0) = NRE$  (symport/antiport P systems of degree 3 using symport and antiport rules of weight 1, or only symport rules of weight 2, are universal [6]);
- $NOP_4(obj_3, sym_*, anti_*) = NRE$  (P systems with four membranes and symport and antiport rules of arbitrary weight are universal even when using only three objects [80]).
- $NOP_3((a), (b), (c)) = NRE$  (P systems with active membranes of degree 3 and rules of types  $(a), (b), (c)$  are universal [64]);

- $NOP_9(endo, exo) = NRE$  (P systems of degree 9 and mobile membranes using brane calculi operations, *endocytosis* and *exocytosis*, are universal [58]);
- $NSP_3(repl_2) = NRE$  (P systems of degree 3 with strings and using replicated rewriting rules, with at most 2 copies of each string produced by replication, are universal [60]).

## CHAPTER 3

### APPLICATIONS OF MEMBRANE COMPUTING

#### 3.1 Motivation

The results from the last years show that membrane computing offers a variety of tools, techniques, and models, which can be applied to biology, linguistics, theoretical and practical computer science problems (for more details see [34]). In addition to previous mentioned areas of application, there are also new approaches, such as economics: in accounting (see [54]), in human resource management (see [13]), etc. Here we also refer to several applications of P systems in linguistics, as a representation language for various concepts related to language evolution, dialogue, syntax and semantics [14], and making use of the parallelism in solving parsing problems in an efficient way [49]), etc.

Before looking at these applications, let us stress the attractiveness of membrane computing as a modeling framework by mentioning several essential features relevant to membrane computing that are of interest for many applications: distribution, discrete mathematics, the property of being algorithmic, scalability/extensibility, transparency (multiset rewriting rules are nothing other than reaction equations from chemistry and biochemistry), massive parallelism, non-determinism, communication, etc.

All the applications have in common some aspects: the concepts are presented in the mathematical formalism of membrane computing and, in addition, they have a graphical representation of cell-like structure, tissue-like structure, and so on; membrane computing also provides a way to visualize the system's evolution (the way rules are applied).

### 3.2 P Systems as Modeling Tools in Biology

There are different domains of computer science which are inspired from the biology behind the living organisms. We are talking about evolutionary computing (genetic algorithms), neural networks, DNA computing, and membrane computing. All of them are known as molecular computing.

Most applications of membrane computing are related to biology. This is quite natural in view of the fact that the theory of membrane systems as a model of computation started from biology as an abstraction of the structure and functioning of biological membranes. The final goal is to develop *in silico* simulators in order to replace the experiments *in vitro* or *in vivo*.

Also, other classes of rewriting systems related to biological phenomena are presented in the literature (see [50], [62], [87]), but the P systems have a membrane structure similar to the cell and they may be seen as computational universal devices and also as simulators of evolution processes by observing the transitions between configurations.

In what follows, we briefly recall some biological applications of membrane computing, mainly following [34], but also other recent papers from membrane computing literature.

### 3.2.1 Mechanosensitive Channels

There are different approaches for describing the evolution of molecular functioning of living cells. As we already mentioned, there are simulator softwares based on the numerical integration of sets of differential equations (using continuous mathematics), such as E-CELL project (which models and reconstructs biological phenomena to allow precise whole cell simulation – see [90]), or Virtual Cell (a computational environment that constructs cell biological models and generates simulations of reactions – see [63]).

A totally different approach is to use discrete mathematics that is more appropriate for the simulation of membrane behavior or the activity of channels in bacteria. One such application is described in [10] for mechanosensitive channels of large conductance (MscL). Mechanosensitive channels, located in the cell membrane, are protein-based channels gated by mechanical forces and they allow the rapid exit of different chemicals decreasing the osmotic pressure inside the cell. Based on the biological notions concerning MscL in bacteria and the way they are gated, two P system models are defined corresponding to *in vitro* patch clamping experiments and *in vivo* hypotonic shocks. These systems consist of some basic components: an environment, a region, a membrane, and rules, which naturally correspond to essential aspects of MscL activity.

In both models, objects are never modified by evolution rules; instead, they are

only exchanged between the internal region and the environment (so they are more communication rules with respect to objects). In addition, by using different target indications, the direct passage of water through the cellular membrane, by osmosis, is distinguished from the passage of chemicals which needs the opening of mechanically gated channels.

In the *in vitro* simulation P system model, 43 rules are used to obtain a cycle simulation, a transition of the channels from closed state to open state and then back to closed. By applying the rules, the label of the membrane is also changed according to the status of the channel (the label is seen as a tension parameter which can take eight values corresponding to real membrane tension and channel status). Probabilities are associated with evolution rules in order to achieve a closer resemblance to biological reality of the priorities between reactions.

The second P system model describes the *in vivo* activity of MscL based on laboratory experiments data. The number of membrane tension values is reduced to 5 and, in this case, probabilities are not associated with rules. With respect to the number of water molecules, for the chemicals from a multiset, a concentration function is defined, and then extended for the whole multiset that is present in the environment or in the region. The transitions among tension values are due to addition of water in the environment and they are coded by environmental rules. The communication of chemicals through channels depends on their concentration and the rewriting rules have the form of concentration-based evolution rules or they may be combined with environmental rules. Using only 10 rules, the P system can predict the possible behavior of MscL for different amount of water molecules placed in the environment.

A software environment has been used to produce simulations of the *in vitro* P system model and to obtain predictions about observable quantities that could be used to test its soundness and validity. The MscL activity was simulated using *EdnaCo*, a virtual test tube spatially arranged as a 3D coordinate system, in which biomolecules may move, interact, etc. The parallel way of applying the rules in a P system is implemented in *EdnaCo* by dividing the virtual test tube into segments that are run on different processors. The main advantage of using *EdnaCo* is that a simulation requiring multiple MscL (in a bacterial cell there are 50–100 channels, not only one as in the *in vitro* model) interacting under the same changing environmental conditions can be run in short time because of the massive parallelism.

The models presented could be enlarged to also include other processes occurring during osmotic shocks, but there is presented only the formal simulation of the activity of MscL in prokaryotes, and the investigation is based on biological data and results obtained from *Escherichia coli*. However, the models are general enough to cover distinct conditions for the functioning of MscL in other prokaryotes too.

### 3.2.2 Biological Dynamics

There are some aspects which are crucial in almost any study of biomolecular processes, and the traditional P systems do not take them into major account. In view of the studies from [17], [21], and [40], these aspects are: dynamical behavior of bio-molecular processes, environmental energy and resources, and asynchronous system control. The focus of the P system on final configuration and maximal parallelism (which leads to consumption of all available resources of the system during one transition) is changed to dynamics of evolution.

New classes of P systems are defined, with a more active role to the membrane boundary and to the environment. These are: *P systems with boundary rules* (PB systems), *PB systems with environment* (PBE systems), and *PBE systems with resources*. The boundary rules (for a formal definition, see [17]) are a special case of communication rules which recall of antiport rules. For PB systems, the notation  $x[_i y$  means that a membrane labeled  $i$  can see outside, close to its boundary, a multiset  $x$ , and inside the membrane is the multiset  $y$ . Biological processes such as periodicity, stability, adaptability, growth, or degeneration, and the relationships between them are implemented in the new types of P systems.

To obtain graphical representation of a P system which describes an oscillatory biochemical system, a simulator called *Psim* is used. *Psim* is a Java implemented simulator with a user-friendly graphical interface which allows the definition of membrane structure by XML files (for details see [20]). The oscillatory process refers to the periodic behavior of biological systems and it is modeled using concentrations associated to biological or chemical elements. The algorithm that is implemented, called metabolic, is inspired by the chemical reading of the rewriting rules. The objects are seen as reactants or products and the rules as chemical reactions with a reactivity coefficient associated. The algorithm is tested with *Psim* to simulate the oscillations of Brusselator model, which is a simplified model of the Belousov–Zhabotinskii reaction (that occurs between reactants such as sulphuric acid, malonic acid, ferroin, and bromate sodium, combined together in presence of a cerium catalyst). The results had been verified by the numerical integration Runge-Kutta method on the associated differential equations.

A second dynamic system is investigated and tested by *Psim*. The system is of *predator-prey* type. To be more concrete, a population of cells (or people) is divided in three groups: healthy, ill, and immune, and its evolution is observed in time. The simulation of this system with *Psim* has shown results which agree with those obtained by solving the Lotka-Volterra equation system. The results have highlighted the existence of a threshold of activation for the epidemic: if the initial healthy population is below a certain amount, then the epidemic does not start and, hence, ill people decrease in number until they vanish; on the other hand, if the initial healthy population is beyond that threshold, then the epidemic activates and the number of ill people grows up until reaching its maximum.

The immune system represents a case of complex adaptive system where the notion of cell membrane is essential. P systems provide a good mean to describe the main processes happening in the immune system. The surface of immune cells (and of other cells too) is covered with receptors that are complex three-dimensional electrically charged structures. The receptor of a pathogen is called *epitope*, and the bond strength between a receptor and an epitope is called *affinity*. In order to formally express the presence of a receptor, instead of using  $[_i$  to indicate a membrane, it is used the notation  $\{ [_i$ , with an interstice between braces and square brackets, intended as a region belonging to the external surface where receptors are detectable from the outside and, in their turn, can detect objects from the environment. This feature can be considered as an extension of the communication mechanism of PB systems and of symport/antiport P systems.

Some pathogens constantly attack the body and can be harmful if they are

left unchecked. Since different antigens have to be destroyed in different ways, the problem faced by the immune system is to recognize them and to choose the right tools for destroying that particular kind of pathogens. The relevant biological properties that characterize the immune system can be represented by P systems with labels for antigens, objects for epitopes, and electrical charges associated to membranes. To protect organisms from foreign pathogens (such as viruses, bacteria, parasites, etc.), the immune system must be capable of distinguishing harmful foreign material from normal constituents of the organism. The recognition of an antigen as foreign in the immune system can be seen as a problem of pattern recognition implemented by binding. For example, lymphocytes recognize pathogens by forming molecular bonds between pathogen fragments and receptors on the surface of the lymphocyte. The more complementary the molecular shape and electrostatic surface charge between pathogen and receptor, the stronger the bond (and the higher the affinity) is.

The architecture of the immune system is made of different levels. The most elementary one is the skin, which is the physical barrier to infection. Another level is constituted by the physiological conditions (such as pH or temperature) that provide uncomfortable living conditions for foreign organisms. When pathogens enter the body, they encounter the innate immune system and then the adaptive immune system. Both systems consist of a multiplicity of cells and molecules that interact in a complex manner to detect and eliminate pathogens. The first system is not changed from the birth of the organism and it provides a rapid defense to keep an early infection in check, giving the adaptive immune system enough time to prepare a more specific response. The adaptive system learns how to recognize specific types of

pathogens, and it retains memory of this for speeding up future responses. Both detection and elimination depend upon chemical bonds. The surfaces of immune system cells are covered with various receptors. Some of these receptors bind to pathogens and some others bind to other immune system cells or molecules.

The membrane system defined to describe the principal steps of innate immune system contains two membranes representing the first two levels of defense before going to the adaptive part of the immune system (that is located in the environment for this membrane structure). The attack of antigens is seen from the inside to the outside of the system, because the most internal region represents the external world, and the most external region represents the last, and more specific body defense. This choice is motivated by the increasing complexity of the processes that must be described.

*Complement* molecules are the primary chemical response of the immune system in the early stages of infection, and they are involved in two distinct phenomena, respectively, *lysis* and *opsonization*. Lysis is the process by which the complement ruptures the bacterial membrane; this action results in the destruction of the bacterium. Opsonization refers to the coating of bacteria with the complement, causing the bacteria to be detected by macrophages (phagocytes which engulf cellular debris and pathogens). Macrophages have receptors both for certain kinds of bacteria and for complement.

*Cytokines* are molecules that act as a variety of important signals, and their release activates the next phase of the host defense, called early induced response. They are produced not only by macrophages and other immune system cells, but also

by some self cells (which are not part of the immune system) when they are damaged by pathogens. Their major effect is to induce an inflammatory response, associated to some physiological changes (fever) which reduce the activity of pathogens and reinforce the immune response by triggering the production of *acute phase proteins* (APP), substances which bind to bacteria, thus activating macrophages or complements. When infected by viruses, certain cells produce *interferons*, a family of cytokines, which inhibit the viral replication. Moreover, they activate certain immune system cells called *Natural Killer* (NK) that kill infected cells. NK cells bind to normal host cells, but they are normally not active because healthy cells express molecules that act as inhibitory signals. When some virally infected cells cannot express these signals they are killed by activated NK cells, that release special chemicals that trigger the *apoptosis* (death) on an infected cell.

P systems are topological spaces without a metric on objects, but a metric can be important when dealing with cellular reactions. The membrane system that is defined has rules which are not used like in the P systems, but in the way of *Psim*; therefore, the process dynamics and effects are regulated at every step by the actual amounts of reactants. Every dissolving membrane delivers its content to the immediately outer membrane, and every dividing membrane replicates its content inside the new membrane. The objects represent biological properties of the antigen, epitopes in all possible forms (before and after lysis), self cell receptors, complements, cytokines, APP as symbol for activation of macrophages, and apoptosis. The rules simulate the processes of entrance of antigens, replication, infection, lysis, opsonization, debris remotion, opsonized antigen remotion, antigen-macrophage complex, functionality

restoration, cytokine production by macrophages, cytokines production by infected cells, APP production by cytokines, complement increasing by APP, macrophages increasing by APP, self cells immunization by cytokines, virus replication/inhibition, NK cell activation by interferons, interferon production by activated NK cells, apoptosis, and self cell death.

In an organism the first response against an inflammatory process consists in the activation of recruitment of leukocytes. Activation relies on the complex functional interplay between the surface molecules. These molecules are differently expressed by leukocytes circulating in the blood, and by endothelial cells covering the blood vessel. A leukocyte cell has some receptors on its surface that bind with counterreceptors located on the surface of the endothelial cells. These bonds slow down the initial speed of leukocyte. Moreover, some molecules, called *chemokines*, are produced by the epithelium and by the bacteria that have activated the inflammation process. Chemokines can bind with the leukocyte receptors, producing signals inside of it. Such signals generate on the leukocyte surface new and different receptors that, while interacting with the endothelial receptors, strongly slow down the cell speed until it stops. There are four stages in the movement of the leukocyte: the initial state with fast circulating leukocytes into the blood, the rolling state, the activation state, and the final adhesion state. A membrane system representing this immunological phenomenon is described in [40], and then a simplified model with two membranes is analyzed by *Psim*.

### 3.2.3 Cell-Mediated Immunity

From the simulation of the innate immune system using *Psim* presented in the previous section, we move to the adaptive immune system modeled by client-server P systems and described in [32]. As we already mentioned, the most important function of the immune system is to distinguish between the harmless self and the potentially dangerous non-self. Adaptive immunity (or acquired immunity) is developed and modified throughout life by learning the specific antigens before removing them from the organism. The most important components of adaptive immune system are the major types of lymphocytes: T cells (80%) and B cells (15%). There are two adaptive mechanisms: humoral immunity and cell-mediated immunity. Humoral immunity is mediated by serum antibodies, which are proteins secreted by the B cells, while cell-mediated immunity consists of the T cells. Each T cell has many identical antigen receptors which interact with antigens.

T cells play a central role in the cell-mediated immunity. When a T cell recognizes a foreign antigen, it initiates several signaling pathways and the cell activates. The T cell activation is caused by an appropriate interaction between the T cells armed with T cell antigen receptors and the antigen cells. The antigen recognition initiates signal transduction, which can be broken down into a series of discrete steps that are related to various molecular events within the signaling pathways.

In order to model the T cell signaling network, a distributed version of P systems is used. The P system application of rules in a maximally parallel manner expresses the natural competition for scarce resources in the immune system. Communication and coordination is essential, and thus symport/antiport rules are consid-

ered for communication among membranes. Since the immune system environment is highly distributed, a new type of P systems, called *client-server P system* (CSPS), as defined in [33], is used to model the T cell signaling pathways and T cell activation. A client-server P system is a P system composed of elementary membranes (except the skin), with state objects modeling the states of the clients, and rule objects modeling the communication (by symport rules) between clients. Client-server P systems were theoretically investigated in [33], where it is proven that CSPSs of degree at most 4 and using symport rules of weight at most 4 are computationally universal.

Starting from CSPS, a *client-server P simulator* (CSPsim) is defined as a set of communicating automata together with appropriate internal transitions for each component, and communication steps between components. Each membrane of a CSPS corresponds to an automaton in CSPsim. The CSPsim adds to the abstract model both qualitative and quantitative features of the T cell signaling network, while a client-server P simulator with two clients has the same computational power as a Turing machine. These abstract simulators represent an intermediate step from a formal theory suitable for theoretical results to a software implementation of a molecular network.

The steps needed to obtain useful results from the software are as follows:

$$\text{T cell real system} \xrightarrow{\text{modeling}} \text{CSPS} \xrightarrow{\text{simulation}} \text{CSPsim} \xrightarrow{\text{implementation}} \text{MOINET}$$

For the implementation of T cell molecular networks model, it is used a new software environment called MOINET (MOlecular NETworks) [33]. The software experiments provide data which is then statistically processed and interpreted. One of the main goals of statistics is inferring conclusions based solely on a finite number

of observations about events likely to happen an indefinite (infinite) number of times. Nonetheless, the strength of conclusions yielded depends on the sample size upon which the analysis is based. In fact, in many cases a big difference appears between the minimum size required by statistics in order to make methods applicable, and the size that biological “wet” experiments can provide. This is why computer simulators for biological processes are needed: the use of such tools overcomes the problems of budgeting, since the cost per software experiment is low in comparison with the biological lab experiments. Therefore, data sets of a desired size can be obtained allowing for correct statistical inferences and hypothesis testing.

The results contribute to explaining how various factors determine differences in the formation and composition of the T cell antigen receptors signaling complexes, and how they drive different biological consequences of T cell signaling networks. The T cell behavior is determined by the signaling network that could engage various cell responses due to potentially different signal types, quantities, and durations.

### **3.2.4 P53 Signaling Pathways**

ARMS (Abstract Rewriting system on MultiSets) is a simple model for chemical reactions and consists only of an alphabet of symbols (molecules) and a set of rewriting rules (reactions); hence, this is a one membrane P system with multiset rewriting rules. The rewriting rules are applied in parallel and, when there are more than one rule applicable to the same object, then one rule is randomly selected. An ARMS cannot be considered a P system because of the lack of a membrane structure. In [88], there is defined a type of ARMS, called ACS (Artificial Cell System), which has a membrane structure with multisets of objects associated to each membrane.

The rules are still applied globally, meaning that there are no different rules for each membrane. The division and dissolution reactions are not implemented by rules as in standard P systems, but by using threshold values. A membrane disappears if the volume of membrane compounds decreases below a given dissolving threshold. When the volume of a membrane's compounds increases above a given dividing threshold, a new membrane is created inside of it and the multiset of objects is divided into two multisets of random sizes between these membranes.

Various types of ARMS had been proposed and used, and we further refer to the ARMS with membrane structure presented in [89], which is equivalent to a P system with multiset rewriting rules and target indications. The system models the p53 signaling pathways and consists of two membranes (one for the nucleus and another one to enclose the cytoplasm) and 9 rules.

The p53 signaling network has been studied intensively because it plays a major role in cell survival, and it protects against genetic instability, which leads to tumor formation. The p53 protein is an important factor in regulating the response of cells to stresses and damage, mainly through the activation of genes involved in cell cycle control, DNA repair, senescence (aging), and apoptosis. In normal cells, p53 is an ephemeral and scarce protein because of its rapid degradation, and it exists in an inactive form. Once a cell has a DNA damage, p53 transforms itself from latent to active conformation and moves from the cytoplasm to the nucleus of the cell. Protein p53 has two levels of activation, depending on the level of DNA damage. The weakly activated p53 prevents damaged cells from proceeding in the cell division cycle and promotes DNA repair, while the highly activated p53 induces apoptosis and

eliminates mutated or irrevocably DNA damaged cells [89].

The evolution of the p53 signaling network modeled by the ARMS with two membranes is then simulated on computer and the results agree with the biological facts: when the DNA damage increases, the p53 protein is activated and translocated from cytoplasm to nucleus; after the damage is repaired, the protein returns to the normal state.

### 3.2.5 EGFR Signaling Cascade

In this section we describe a model for epidermal growth factor receptor (EGFR) signaling cascade simulated by a continuous variant of P systems [82]. EGFR is an important biological target for the development of novel anticancer therapies and it had been investigated using mathematical formalization of differential equations, which is focused on the description of the change in concentration of the chemical compounds. The proposed continuous P system is a different approach of simulating the intracellular signaling networks by using a topological and modular view.

For the usual discrete P systems, the rules are applied in a maximally parallel way an integer number of times. A continuous P system can evolve in every instant by applying a maximal set of rules a positive real number of times determined by a given function  $K$ . This way of applying the rules is inspired from the *in vivo* chemical reactions evolution which is continuous and based on the concentration of the reactants. The objects also have non-negative real number multiplicities and take part of the continuous multisets. The rules are of the form  $u[ v ]_i \rightarrow u'[ v' ]_i$ , where the multisets  $u$  and  $v$  represent the reactants, the multisets  $u'$  and  $v'$  represent the products, and  $i$  is the membrane relevant for the reaction. A configuration  $E(t)$  of a

continuous P system at an instant  $t$  is given by a matrix of non-negative real values  $a_{ij}$  representing the multiplicity of the  $j$ th object in the membrane  $i$  of the system.  $K(r, E(t))$  is the rate of application function and associates with each rule  $r$  and each configuration  $E(t)$  a non-negative real value representing the rate of application of rule  $r$ . The standard P systems perform computations and produce results, whereas in the case of continuous P systems the evolution at various moments of time is significant.

EGFR signaling cascade is a complex process and it is modeled by more than 60 proteins and complexes of proteins and 160 chemical reactions. The continuous P system consists of an alphabet of objects representing the proteins and the complexes of proteins, and a membrane structure with three regions (for the environment, the cell surface, and the cytoplasm). The initial multisets present in the regions have values of concentrations of substances from real experiments. For rules, the rate of application function is computed based on the *Law of Mass Action*, which means that the rate of a reaction is proportional to the product of the concentrations of the reactants.

In order to simulate evolutions of continuous P systems on computers, they need to be approximated by discrete P systems. The effect of a rule  $r$  during an interval of time  $[t_l, t_{l+1}]$  of length  $p = t_{l+1} - t_l$ , for  $p$  small enough, is approximated by  $pK(r, E(t_l))$ . By this approximation, a usual P system is obtained, which performs a finite number of steps and at each step the rules are applied  $pK(r, E(t_l))$  times, in a bounded parallel manner (as defined in Section 2.2).

The model of EGFR signaling cascade is implemented using CLIPS (C Lan-

guage Integrated Production System), which is an expert system tool that allows the construction of needed rules and objects. The evolutions of the continuous P system is approximated by computations of a discrete P system working in a bounded parallel manner of parameter  $p = 10^{-3}$ . The graphical results show the effect of concentrations of the most relevant proteins in the signaling cascade over time and suggest that the cascade is robust to variations in the EGF (epidermal growth factor) concentration, which agrees with experimental data. This model may be used for predictions and new hypothesis about the behavior of EGFR signaling cascade in the cell cycle and tumor genesis.

### 3.2.6 Quorum Sensing in Bacteria

In [16], a new variant of P systems with generalized boundary rules is used to model the quorum sensing system of the marine bacterium *Vibrio fischeri*. The model provides a description for the reactions involved in the quorum sensing regulatory network in an arbitrarily large colony of bacteria seen as a whole complex system. Usually, bacteria are considered as independent organism, but certain bacteria, such as *Vibrio fischeri* bacterium, exhibit coordinated behavior which allows an entire population of bacteria to regulate the expression of specific genes depending on the size of the population. This cell density dependent gene regulation system is known in literature as *quorum sensing*. In this way, bacteria can effectively communicate to each other by responding to changes in the concentration of the signal molecules present inside of them and in the environment.

The P system model used is a cell-like P system consisting of a number of compartments, which represent the bacteria placed inside of the skin membrane that

represents the environment. The elementary membranes communicate only with the environment region and not directly to each other. Only 5 objects are necessary to encode the proteins and the complexes of proteins involved in the quorum sensing process. The rules have the form:  $j : u[ v ]_i \xrightarrow{k_j} u'[ v' ]_i$ , with the difference from the previous mentioned model (from [82]) being in the rate of application function that is replaced by a kinetic coefficient  $k_j \in \mathbf{R}^+$  associated with each rule  $j$  (as a measure of rule's reactivity). Using these kinetic coefficients and a mass action law, at each step only one rule is applied, meaning that the evolution of the system is not based on parallelism. The set of rules consists of 2 rules related to the environment membrane and another 12 which correspond to all inner regions representing the bacteria (all having the same label).

The P system defined to model the quorum sensing depends on the considered number of bacteria in the colony and on the choice of the real values for the kinetic coefficients associated with each rule. The system is implemented using *Scilab*, a scientific software package for numerical computations with a user-friendly interface. The behavior of the system is changing for populations of different sizes and, for this reason, it is important to see how a bacterium can sense the number of bacteria in the colony and respond (by producing light) only when the number of individuals is big enough. First, the model is simulated for a population of 300 bacteria and the graphical results show the evolution over time of the bacteria which respond and the number of signal molecules sent to the environment. The signal molecules accumulate in the environment until saturation (a specific threshold of concentration) and then bacteria are able to detect that the size of the population is big enough. Moreover,

the evolution of a population of only 10 bacteria shows no coordination of the colony behavior.

The simulations indicate that *Vibrio fischeri* bacterium has a quorum sensing system where a single bacterium guesses that the size of the population is big enough and starts producing light. This bacterium massively produces signal molecules and, if the signals do not accumulate in the environment (meaning that the guess was wrong), then it switches off. On the other hand, if the signals do accumulate in the environment (meaning that the number of bacteria in the colony is big), then a recruitment process takes place and makes the whole population of bacteria to glow. These results agree with the *in vitro* experiments.

### 3.2.7 Respiration and Photosynthesis in Cyanobacteria

Starting from the evolution-communication P system proposed in [26], a new probabilistic model was defined in [11], and then the mathematical model results were compared with the biology reality by simulations (for details about simulator see [11], [27]). The software was used to simulate simple biological phenomena related to respiration in *Escherichia coli* and the interaction between respiration and photosynthesis in cyanobacteria.

In an *evolution-communication P system* (EC P system), there are two types of rules: rewriting rules for evolution and symport/antiport rules for communication that work simultaneously and independently. Each copy of a symbol (object) evolves according to given evolution rules that represent chemical reactions associated with the regions and described by rewriting rules. In the same time, symport/antiport rules are applied, simulating biochemical transport mechanisms present in the cell.

The transport of molecules and chemicals (objects) across membranes can be passive or active. The transport is *passive* when molecules pass across the membrane from the compartment with a higher concentration to that with a lower concentration, and no energy is used for the transport. An example of passive transport is the entry of oxygen molecules by diffusion into the cell of *Escherichia coli* bacteria or the exit of carbon dioxide outside the bacterium. These two passive processes are both important for the aerobic respiration in *Escherichia coli*. The transport is *active* when molecules pass across the membrane from a compartment with a lower concentration to one with a higher concentration. In this case, it is necessary to consume some energy to accomplish the transport.

The symport of different substances is needed for bacterial growth. In *Escherichia coli*, the protons are moved in symport with either lactose, arabinose, or galactose. An example of antiport mechanism is the proton-sodium antiport found in many bacteria, where the main function is the maintaining of a constant concentration of either protons or sodium ions inside the cell [27].

The application of the rules of either type, evolution and symport/antiport rules, of an EC P system is made in a non-deterministic and maximally parallel way and the process is synchronized. Any computation, halting or non-halting, of a system is considered, corresponding to the biochemical transformations and transport processes in a living cell.

A software simulator is built to implement EC P systems with probabilities associated to rules implying a weak priority that solves the competition of rules for each copy of an object. The simulator takes, as input, the rules of the EC P system

to simulate, the structure of the system, and the occurrences of the objects present at the beginning of the computation in the regions of the P system. In addition, it needs two types of probabilities for each rule: *the probability to be available* and *the probability to win a conflict*. The probabilities of rules to be available (given in the initial configuration) are independent of each other and also independent of the presence of objects. The probability of a rule to win a conflict over a symbol with other rules is computed at each step and is dependent on the coefficients associated with each rule (the initial coefficients are given). The phases followed by the simulator are: list all available rules, search for conflicts, solve the conflicts, execute the rules, and then repeat the process for a new step of computation.

A first interesting application on the simulator is an one-membrane EC P system with only one symbol  $a$ , two evolution rules:  $r_1 : a \rightarrow aa$  (representing associations) and  $r_2 : aa \rightarrow a$  (representing dissociations), and equal probabilities. The model may be seen as describing the processes from the “origin of life on Earth”, [27]. Even the system could seem very stable, the results over 20000 steps show that the number of copies of symbol  $a$  is changing with large variations, meaning that the behavior of the system is unstable.

A biological component of the real cell is the *enzyme* which allow the chemical reactions to occur. In P system area, they are sometimes called *catalysts* and they had been used in many classes of P systems. Another concept present in biology is the *activity rate* of an enzyme, which is defined as the speed of the enzyme or the number of reactions that it can catalyze in a fixed unit of time. Every type of enzyme has its own activity rate, and this can change in time according to biological parameters

dependent on the particular chemical reaction considered. Usually, each object in a classic P system has the same activity rate or velocity (at each step, each copy of an object is used, if possible). For the catalysts used in an EC P system different activity rates can be considered as the number of rules where the catalyst can be used in one step.

Respiration is the biological process that allows cells to obtain energy from a flux of electrons moving from electron donors to a final electron acceptor, which in most cases is the molecular oxygen. In *Escherichia coli*, as well as in other bacteria, the cell ability to consume molecular oxygen during the respiration is determined by the presence of two different enzymes that catalyze the final step of respiration: the reduction of molecular oxygen with protons and electrons. These two enzymes are: *cytochrome bd* and *cytochrome bo*. The activity rates of these two enzymes are different, according to the percentage of saturation of molecular oxygen.

The simulator is used for a simple EC P system with one membrane that models the consumption of oxygen in the respiration process of *Escherichia coli* in three different cases: when only *cytochrome bd* enzyme is present, when only *cytochrome bo* is present, and when both are present. The system has as parameters the number of copies of oxygen objects (nanomols of oxygen) and the number of copies of enzymes (unit enzymes). The results showing the oxygen consumption, represented as diagrams, are compared for the three cases and for different quantities of enzymes. The difference between the cases when the enzymes are used separately and when they are used together is not large; when the two enzymes are used together, the speed of the consumption of oxygen is approximately half of that when only the enzyme *cy-*

*tochrome bo* is used, and then doubled when only the enzyme *cytochrome bd* is used. In practice, a stronger difference can be noticed when the quantity of oxygen available is very low and the affinity of the two enzymes becomes a fundamental parameter.

In *Escherichia coli*, the consumption of molecular oxygen is related to the *translocation* (pumping out) of protons outside the cell. Specifically, *cytochrome bo* enzyme translocates two protons for every electron transported to molecular oxygen, while *cytochrome bd* enzyme translocates only one proton for every electron. Given the situation where only consumption of molecular oxygen and translocation of protons are taken into consideration and where only one type of enzyme is present, the results show the accumulation of protons in the environment after a number of steps (each step is equivalent to 60 minutes).

These results may be very helpful in academic studies and bioindustrial activities because enzyme activity or enzyme quantity cannot be measured in intact cells, while oxygen measurements are laborious and time consuming.

Respiration process can be found also in cyanobacteria, which is the largest group of aquatic photosynthetic prokaryotes. Cyanobacteria have the ability to perform both processes of photosynthesis (within the thylakoid membranes) and of respiration (within the plasma membranes and thylakoid membranes). The process of photosynthesis consists of using electrons from water to reduce carbon dioxide, producing other chemicals. The first reaction in photosynthesis is the splitting of water, at the expense of light energy, to produce molecular oxygen, protons, and electrons. In respiration, the opposite process occurs: water is produced by the consumption of oxygen during its combination with protons and electrons, as in *Escherichia coli*. In

cyanobacteria, there is a strong interaction between respiration and photosynthesis: the oxygen produced by photosynthesis in the inner membrane (thylakoid membrane) is used in the cell membrane for respiration and, in the same time, the carbon dioxide produced by respiration in the cell membrane is used in the inner membrane for photosynthesis.

The EC P system considered to model the processes of respiration and photosynthesis in cyanobacteria consists of two membranes, one for the cell membrane and one for the thylakoid membrane. The system is closed, meaning that there is no exchange of chemicals with the environment (oxygen and carbon dioxide are not introduced in the system). Water and light are supposed to be present in inexhaustible quantities. The results show that after some time (one step of computation is equivalent to 10 minutes) the oxygen stops to accumulate and there is the same amount of oxygen in the cell membrane as carbon dioxide in the thylakoid membrane. The simulation of the EC P system was considered in two situations: with oxygen present in the initial configuration and without, the difference between the results was only in the number of steps needed for the system to get at the same state.

In cyanobacteria, it is possible to inhibit the production of oxygen during photosynthesis by adding a synthetic chemical inhibitor called *diuron*. Using diuron in low concentration, it is possible to decrease the production of oxygen by 50%, while the consumption of oxygen is not modified. This known biological fact is obtained by simulation of an EC P system in two cases: with oxygen present in the initial configuration and without. The addition of diuron to the system changes the results only in the increasing number of steps (time) needed. To verify these simulations

results, the real values were considered for cyanobacterium *Synechocystis PCC 6803*.

The process of proton translocation is considered in the case of another cyanobacteria, *Anacystis nidulans*. The proton translocation consists in the pumping of protons, outside the cell, when oxygen is consumed; therefore, the decrease of oxygen concentration inside the cell corresponds to an increase of proton concentration outside the cell. To model only the reaction involving oxygen and protons in the process of pumping of protons in *Anacystis nidulans*, it is considered an EC P system with two membranes, the cyanobacterium cell and the environment. The consumption of oxygen takes place inside the cell membrane, and by using the symport rule present here, the protons are sent out and accumulated in the environment. One step of simulation corresponds to 2 minutes, and the results indicate that after a few steps the amount of oxygen in the cell is the same as the amount of protons accumulated outside. The number of steps always depends on the initial configuration.

Using the basic definition of an EC P system plus additional features, and an improved simulator (in order to consider, for example, the fact that the affinity of enzymes is involved in the oxygen concentration), other biological processes could be modeled.

### **3.2.8 Photosynthesis**

All cells are surrounded by a plasma membrane, which is made of a bilayer of phospholipids. Within this membrane is the cytoplasm, which is composed of fluid and organelles of the cell. There are many enzymes embedded in the membranes of organelles which act as catalysts for biochemical reactions taking place in a cell. Because the membrane of a living cell is not a thin film but a complex structure, it

is natural to consider the inner region of a membrane in between the two layers of phospholipidic molecules. In this direction, a first extension of standard P systems is considered in [69] and [70], i.e., P systems with inner regions of membranes, besides usual regions delimited by membranes.

The inner region of a membrane is hydrophobic because there are hydrocarbon chains of lipids inside the membrane and phosphoric acids outside. The structure of the membrane induces a selective permeability for molecules. Small molecules, such as  $H_2O$ ,  $O_2$ , or  $CO_2$ , easily go through membranes. On the other hand, large molecules and all ions cannot diffuse across membranes. However, the proteins embedded in the membrane control transportation of ions and large molecules. The membranes of living cells and the organelles in cells discriminate between the inside of the membrane and the outside.

The computational power of the family of P systems with inner regions of membranes is identical to that of standard P systems. In [69], it is proven that P systems with inner regions of membranes of degree 1 and with one catalyst are universal.

The complex process of photosynthesis is analyzed more in depth by extending the P systems with inner regions of membranes to **R**-subset transforming systems with membranes, as in [69] and [70], where real values are allowed for multiplicities.

Photosynthesis of plants is a process which occurs in *chloroplasts*. A chloroplast catches light energy, converts the energy into chemical energy, and produces starch from  $CO_2$ ,  $H_2O$ , and chemical energy. In a chloroplast, there are many membrane-surrounded structures called *thylakoids*. The region inside the thylakoid

membrane is called *lumen*; the space between the chloroplast envelope and thylakoids is called *stroma*. Photosynthetic reactions are classified into two groups: light reactions and dark reactions. Light reactions separate water into  $O_2$  and  $H^+$ , reduce NADP (Nicotinamide Adenine Dinucleotide Phosphate) to NADPH, and synthesize ATP (Adenosine Triphosphate). The enzymes which act as a catalyst of light reactions are embedded in the thylakoid membrane. Dark reactions make starch from  $CO_2$  and  $H_2O$  using the reduction power of NADPH and the chemical energy of ATP. Dark reactions occur in stroma. In moderate luminosity, the products of light reactions are all consumed by dark reactions. If the light is strong or dark reactions stop, then the products, NADPH at stroma and  $H^+$  at lumen, become harmful to the structure of chloroplasts. Plants have a mechanism, called *photoinhibition*, to depress light reactions in high luminosity.

The **R**-subset transforming system, called *Photo*, is constructed to model light reactions and photoinhibitions of photosynthesis, by considering probabilities associated to chemical reactions. *Photo* has three regions: the stroma, the inner region of the thylakoid membrane, and the lumen. The envelope membrane of chloroplast is the skin membrane of *Photo*. The system behaviour is then simulated on a computer with various parameter values and different initial **R**-subsets. When no light is present, the simulation of *Photo* shows soundness. The photosystem decreasing activity is effective in preventing damages caused by a low pH under strong light conditions. Photoinhibition reactions are important for the chloroplasts. *Photo* is sensitive to the threshold of pH suggesting that different thresholds correspond to different plants which grow under different light conditions. *Photo* is insensitive to

the threshold of NADP for photoinhibition. This may suggest that photoinhibition is triggered mainly by pH.

The results obtained by *Photo* are compared to the ones from a model of photosynthesis using a conventional method based on differential equations. The traditional model is a dynamical system of three differential equations which are highly nonlinear and cannot be solved analytically. Numerical integration on a computer gives the behavior of the variables of the differential equations, but it is much simpler to construct a model using P systems and to simulate it on a computer than to construct a system of differential equations and then integrate it numerically.

As said before, there are many other applications of P systems in biology, but we stop here, since we already have an image about the usefulness of this approach, as well as a hint about the many modifications of various ingredients of P systems which were introduced in this framework.

### 3.3 Applications in Computer Science

Inspired from biology, membrane systems are computer science devices equal in computing power to Turing machines and used for solving hard decision problems, but also in many other applications. We mention here only several applications: in computer graphics (where the compartmentalization seems to add a significant efficiency to well-known techniques based on L systems, [47]), in devising sorting and ranking algorithms [8], [9], handling 2D structures [29], etc.

### 3.3.1 Static Sorting P Systems

This section refers to the application of P systems for sorting problems (strong sorting, weak sorting, and ranking) that are very important in computer science and for which many algorithms, both sequential and parallel, were developed. As for now, the time complexity remains at least  $O(n \log(n))$  for the sequential case and  $O(\log^2 n)$  for the parallel case. In [8] and [9], we find a study of various algorithms based on different models of P systems and their time complexities with respect to the maximal number of integers to be sorted or to their sum. The feature shared by these algorithms is that the input components are placed in an initial input membrane, and the computation dissociates this input according to the relation order among the multiplicities of components. In this way, the sorting is interpreted as the order of elimination of the objects. The idea behind many of the algorithms is to consume objects from all components at once and, when one component is exhausted, to trigger a signal to find the next component to be eliminated. In other algorithms, a comparator is developed, which can be used practically in any sorting network design. For most of the algorithms, the time complexity will be also linear, while for others it will depend on the largest multiplicity. For solving a practical problem, the algorithm has to work in a deterministic or confluent way and this is because the answer of the problem has to be received in a specified time. In addition, the system has to stop or reach an equilibrium state after finishing the computation so that the output can be read.

Weak sorting is an algorithm which processes a multiset (the multiplicities represent the integers to be sorted) and gives as a result a multiset, the weak sorting

string, where the objects' multiplicities are ordered. The strong sorting algorithm outputs the objects with the same associated multiplicities as in the input multiset, but present in a multiset, the strong sorting string, in increasing order of their multiplicities. Finally, the ranking algorithm produces a word, the ranking string, of objects ordered according to their multiplicities from the input multiset.

A first algorithm for the integer sorting problem uses a P system with promoters and cooperating rules. In the case of promoters, the rules are applied only in the presence of certain objects which can evolve at the same time as objects whose evolutions they support. The time complexity for the strong sorting algorithm with P systems with promoters is  $2k + 1$ , where  $k$  is the number of elements to be sorted, and it is constant with respect to the values of the elements. The number of elements of the system's alphabet is exponential with respect to  $k$ . In this algorithm, the sorting is successful only if all the integers are different. To solve the general problem, the P system is changed to reduce the problem to the sorting of different numbers. By using inhibitors, it can be specified when the execution of some rules should not happen, and the production of objects is driven in the right order. The time complexity for the strong sorting algorithm with P systems with inhibitors is  $2k$ .

Another biologically inspired model is represented by P systems with weak priorities. In nature, systems evolve according to rules that usually are in some ordered relation. Moreover, systems evolve in time according to sequences of rules based on the priorities, in parallel, up to some moment when they reach an equilibrium state. Using weak priorities means that in one step of computation the rules are applied in a sequence according to the priority relation as much as possible for a

specific rule, and in the maximally parallel manner for those rules for which the relation is not defined or rules that have the same priority (with competition for objects if it is the case). The time complexity for the strong sorting algorithm with P systems with weak priorities is 2 plus the sum of all the elements to be sorted.

Using strong priority means that in one step of computation only the rules with the highest priority are applied in the maximally parallel manner, regardless of whether rules with a lower priority can be used for the remaining objects. The use of strong priority controls more strictly the computation process and, as a result, the degree of sensitivity is smaller than in the model where the weak priorities are used. Another one membrane P system is built, using strong priorities among rules with finite-states catalysts. A  $s$ -stable catalyst has  $s$  states  $c_1, \dots, c_s$  and by applying a rule this catalyst may switch between states. The time complexity for the strong sorting algorithm using P systems with strong priorities and  $s$ -stable ( $s = k + 1$ , where  $k$  is the number of integers to be sorted) catalysts is equal to the sum of the elements to be sorted, plus their maximum.

Another sorting algorithm is implemented by a P system with membrane dissolution. If a rule  $x \rightarrow y\delta$  is applied, then the objects of  $x$  are consumed, the objects of  $y$  are produced, the corresponding membrane is dissolved, and all its contents pass to the upper region. The time complexity for the weak sorting algorithm with P systems with object rewriting rules is  $2k^2 + 3k + 4$ , but the number of different objects used is exponential with respect to  $k$ , where  $k$  is the number of elements to be sorted.

If the P system allows mobile catalysts (that can be moved from one region to another), the time complexity for the strong sorting algorithm is linear with respect

to the maximum number of elements to be sorted, and it also depends on the number of components to be sorted and on the sum of the elements.

One of the sorting algorithms is Bead Sort. The construction from [12] uses a tissue-like P system implementing this algorithm. In such a system an antiport rule of the form  $(i, x|y, j)$  means the simultaneous exchange of objects of multiset  $x$  from cell  $i$  with objects of multiset  $y$  from cell  $j$ . The number of cells is  $k \times m$ , where  $m = \max\{n_1, \dots, n_k\}$ . The positive integers  $n_1, \dots, n_k$  to be sorted are represented by a set of “beads” and they slide along the “rods” to their appropriate places. This simple idea is also used in [9] for the construction of a tissue-like P system with “rods” represented by cells that can communicate (the membrane structure is of degree  $m \times k + k$ , where  $m \times k$  represents the  $m$  rods with  $k$  levels) and beads represented by objects from multiset  $x$  placed inside membranes (and a special object  $\#$  to represent the absence of a bead). Thus, the problem is solved in a purely communicative way. The time complexity of this solution is linear, but the features used are quite powerful (antiport rules of unbounded weight), the descriptive complexity of the solution is rather high ( $m \times k + k$  membranes), the initial data has to be a priori distributed among the  $m \times k$  cells, and the result is obtained in the environment.

A study on this topic was also done in [30], where the P systems with inhibitors/promoters and symport/antiport rules were used to develop comparators and then to organize them in a sorting network. The input data was placed in different membranes and the computation started operating on elements already dissociated. A P system with symport/antiport rules and priorities was constructed to compare the multiplicities of a single object. The result was not obtained in a halting con-

figuration but in a stable one, meaning that there were rules still applicable, but their application did not change the string/object contents of the membranes or the membrane structure. The time complexity was linear with respect to the number of components, and the number of membranes used in the computation was proportional to the number of components.

Also in [30], it is presented an algorithm that implements a weak sorting using a rewriting P system with symbol objects and with weak priorities for rules having at most two objects on the left hand side. The idea behind this algorithm is the parallel simulation of an odd-even sorting network. The time complexity for the weak sorting algorithm with P systems with rewriting rules is  $2k + 1$ , where  $k$  is the number of elements to be sorted. This idea of parallel simulation of an odd-even sorting network is further used in [9] for an evolution-communication P system with non-cooperative evolution rules without target indications and symport/antiport rules of weight 1.

We end this section about static sorting P systems by pointing out an interesting result concerning this topic: starting with objects that do not have any order and are mixed together in a multiset, the order is constructed by computing. Many types of membrane system models are investigated and they behave in slightly different ways when addressing the same problem.

### 3.3.2 Sub-LP Systems Used In Computer Graphics

In this section we describe a model which simulates the growth and the development of living plants, with a variety of information regarding the behavior of the modeled plants as well as graphical representations of the simulation. The approach from [47] develops a model and a specification language based on L systems models

and their use in graphics. This approach presents a more modular view on modeling in general and plant development in particular.

Among the commonly used models to represent plants graphically are those based on grammars. L systems model is the most widely studied for plant representation. This is a string-rewriting model, which was inspired by the developmental processes occurring in simple algae. String rewriting in L systems proceeds in parallel, unlike Chomsky grammars, where rewriting is sequential; this creates an analogy with cellular growth and division, which also proceeds in parallel in multicellular organisms. Many extensions of L systems have been suggested, and each increases the power of the model in a different direction. An advantage of L systems is that they are developmental mechanisms. Not only do they construct a plant structure as it would exist at one particular moment in time, but they describe its growth and development.

The most widely used plant simulation package is *L-Studio/cpfg* and it uses the combination of L systems and turtle interpreter (for more information see [53]). It consists of two parts: *L-Studio* is the integrated development environment where a designer creates and edits models, while *cpfg* is the software that runs simulations of the models and produces graphical output using a turtle interpreter. Here, a “turtle” is a cursor that moves and rotates in space, drawing lines in its path. The symbols in the L system string are interpreted as drawing commands by the turtle, thus producing plant-like shapes. These commands instruct the turtle to move forward, rotate, and draw lines, while many other commands exist for fine-tuning the drawing process.

Rewriting rules have been introduced in P systems with the specific motivation of capturing aspects of modularity typical of some of the existing extensions of L systems. In this respect, rule rewriting can be interpreted as an operation to develop a particular substring inside a region of the system in the form of a rule that can be moved from one region to another and used to insert the substring into another string present in the system. Both rules and strings can be moved, but rules can be moved only as a consequence of the rules applied to the strings currently associated with the various regions of the system. In a sense, this feature introduces a certain level of interaction among the strings in various regions of the system.

The new model that is introduced in [47], called *sub-LP system*, is based on recursive strings of symbols, parametric symbols, rewriting rules that include communication and dissolution (these may be either conditional or non-conditional), migrating numerical variables and migrating arithmetical rules. The rules are applied in parallel and it is assumed that no conflicts occur between rule targets. Membranes are used to separate the structure of the system into regions, each of which performs a computation. When regions do not communicate, each performs an isolated computation. Membranes are labeled with distinct positive integers. A membrane's region may contain any number (or none) of variables, arithmetical rules, string rewriting rules, and other membranes. Every membrane contains one and only one symbol string. Thus, sub-LP systems are hierarchically structured just like P systems. The top-level membrane is known as the skin membrane. The skin membrane and its symbol string are the only required components of a sub-LP system.

A software system has been prepared for testing the sub-LP systems models.

*SubLP-Studio* is a Java application (see [46]) which produces models and simulates their behavior. As it obtains symbol strings at each simulation step, it passes them to *cpfg* for rendering. Reusing *cpfg* makes the software simpler. Also, the comparison of L systems and sub-LP systems is more objective, since the interpreting mechanism is kept constant.

P systems are systems of nested membranes. This nested representation implies a tree-like hierarchy of regions, which has an analogy with branching in plant structures. This analogy is only loosely used in sub-LP systems, as branching is modeled by both grammatical rules and membrane nesting. Allowing this option gives the designer freedom to either exploit the representational compactness of L system-like grammatical rules, or to use membranes to localize computation.

### 3.3.3 An Analysis of a Public-Key Protocol with Membranes

This section presents an approach belonging to the membrane computing area used to address a well known combinatorial problem: the analysis of cryptographic protocols. In [65], is described and logically analyzed the Needham-Schroeder public key protocol (NSPK), which has a well known solution, so that the results can be compared and validated. The approach taken here consists of the exploration of the state space of the protocol for a systematic search of attacks. More interesting is the study of the representation and generation of states, rather than a new search strategy. This approach is motivated by the opinion that the representation of data is a central problem in biocomputing.

The goal of the logical analysis is to find an interleaving of elementary actions (sending and answering messages) that allows an intruder to obtain confidential in-

formation. Cryptographic protocols define the exchange of a few messages between parties in order to distribute some secret data like cryptographic keys or to authenticate themselves. These messages are built with cryptographic primitives, like encryption, signature, or hash functions, and therefore the security of protocols relies on the strength of the cryptographic functions in use. However, it appears that even though these functions are assumed unbreakable, the security of a protocol can be compromised by an unexpected interleaving of messages between honest agents and a malicious intruder which has some limited control over the communication network. Such attacks can be realized at almost no computational cost and they can have disastrous consequences. Various formal methods have been proposed for the automation of the analysis of the vulnerability of cryptographic protocols to logical attacks, both for searching for flaws of this kind or for the formal proof of their absence.

In this section, we describe an experiment using membranes for modeling the NSPK cryptographic protocol and finding attacks by state exploration. The declarative style of membrane computing framework is strongly advocated by the intruder-centric model which is generally considered in order to apply formal methods to cryptographic protocol verification. In this model, the agents executing the protocol communicate asynchronously via a unique channel which has been compromised by an intruder. The intruder is able to spy and divert every message on the channel, and to analyze read messages, with the restriction that he must know the appropriate encryption key in order to decipher an encrypted message. It can also build and send new messages, possibly under a fake identity. The global state of the system can hence be represented by a heterogeneous set containing the local states of each

agent (with a bounded memory), the messages known to the intruder, and the messages sent and not yet received by an agent. The actions of the agents, receiving and sending messages, as well as of the intruder, can be modeled using rewriting rules on multisets.

The description of the protocol involves two different kinds of components: entities and evolution rules. The entities are records and evolution rules are given by rewrite rules. A system state is a finite collection of entities of three kinds: agents, messages transmitted through the network, and messages components memorized by the intruder. The model is organized into the following parts: record definitions are used to describe the three kinds of entities; various predicates, which are used to select from the set of reacting entities a specific entity of a given kind – an agent or a message; rules specifying the abilities of the intruder to collect all the messages that have been exchanged between agents and extract pertinent information; rules specifying the abilities of the intruder to produce fake messages from the information gathered; rules specifying the receiving and sending of messages by agents (such rules are defined as reactions between an agent and a received message which fulfills some conditions); and rules implementing a state exploration procedure which halts with a predicate checking whether a bad state is reached, and whether the search of an attack is successful.

The idea to implement the logical analysis of NSPK is to aggregate all the entities involved into the protocol in a single multiset acting as a chemical solution containing the agents, the messages, and the revealed information. The agents and the intruder will react with messages to augment the solution with new information.

All information is in the solution at the same level. The basic idea is to generate all strings of bounded length made of four symbols representing an evolution of one of the agents. The combinatorial generation of such a string can be done randomly. The use of membranes permits to handle correctly the fact that an agent may have to react to more than one message, leading to more than one evolution of the state.

To validate the new model of the logical analysis, it is implemented using the **MGS** programming language. **MGS** is a research project devoted to the design and the development of a programming language dedicated to the simulation of biological processes [48]. Based on topological distances, **MGS** supports the notion of transformation: a localized computation specified by rules. Thus, **MGS** can potentially be used to process membranes, despite the fact that the **MGS** project focuses on the design of a programming language rather than the development of a well founded computational model.

The problem of finding attacks of protocols is highly undecidable, the state space being infinite for several reasons: the unboundedness of the number of agents present, the ability of agents to generate new random data, the unlimited size of terms generated by the intruder. The problem of protocol security becomes decidable when the number of agents considered is bounded (whenever there exists an attack, there exists an attack involving messages of a bounded size). Note that another version of this experiment is described and fully detailed in [66]; it goes further by generalizing the approach to the exploration of general state spaces and does not rely on the assumption that attacks involve messages of bounded size. The complete running code of the two versions has been implemented in **MGS** and it is detailed in [66].

### 3.3.4 Membrane Algorithms

In [71], a new type of approximate algorithm for optimization problems, called *membrane algorithm*, is proposed. A membrane algorithm consists of several regions separated by means of membranes; in each region, few tentative solutions of the optimization problem and a sub-algorithm are placed. In every region, the solutions are updated by the sub-algorithm simultaneously. The best and worst solutions, with respect to the optimization criterion, in a region are moved by transporting mechanisms to adjacent inner and outer regions, respectively. By repeating this process, a good solution will appear in the innermost region. The membrane algorithm repeats updating and transporting solutions until a termination condition is satisfied. The best solution in the innermost region is the output of the algorithm.

A membrane algorithm borrows nested membrane structures, rules in membrane separated regions, transporting mechanisms through membranes, and dynamic structures of rules and membranes from P systems, and uses all these membrane computing ingredients to solve **NP**-complete optimization problems approximately.

The algorithm solves the Traveling Salesman Problem (TSP) using as sub-algorithm in the innermost region the tabu search (search a neighbor of the tentative solution by exchanging two nodes in the solution and, in order to prevent a node from appearing in the same solution twice, a tabu-list which consists of nodes already exchanged is constructed). For an instance of the TSP, randomly there is constructed one tentative solution for the innermost region and two tentative solutions for every other region. For all the other regions, the sub-algorithms used resemble genetic algorithms (recombine the two solutions or, if the solutions are identical, then the

recombination produces no new solutions).

The membrane algorithm is implemented using the Java programming language. Computer experiments show that the membrane algorithms solve the TSP better than the simulated annealing algorithm. The more membranes a membrane algorithm has, the better obtained the results are (of course, the computation time is proportional to the number of membranes). All programs used in the computer experiments can be downloaded from

<http://www.comp.pu-toyama.ac.jp/nishida/>.

This algorithm is improved by incorporating the concepts of tissue P systems and of P systems with a dynamic membrane structure. A compound membrane algorithm has two phases: first, a number of membrane algorithms produce good solutions from randomly generated initial solutions; then, these solutions become the initial solutions of the second phase. The compound membrane algorithm always outputs almost strict solutions; but, on a single processor, the computation time of the compound membrane algorithm is, as expected, much longer than that of the simple membrane algorithm. However, because in the first phase the membrane algorithms work completely independent, the compound membrane algorithm can be easily implemented on a distributed computing system, and the computation time will then be only twice as long as that of a simple membrane algorithm. Another improvement is to use shrink membrane algorithm by incorporating dynamic membrane structures and having different membrane algorithms in the first and second phases of the compound membrane algorithm.

### 3.3.5 Computationally Hard Problems

In this section, we mainly follow the survey from [84], which shows that membrane computing provides efficient solutions to decision problems through families of cell-like membrane systems, stressing on the SAT problem, as we have a solution for it in Section 5.6.

There are many theoretical requirements for a P system to provide an algorithmic solution to an abstract decision problem, such as all computations of the system must halt (providing a positive or negative answer to a particular instance a problem). Also, the system must be confluent. This is a generalization of the notion of determinism because it is required that all possible computations to provide the same answer.

There are significant differences between the solutions, dividing them into two groups: the *semi-uniform* solutions, which associate with each instance of the problem one P system to solve it, and the *uniform* solutions, which associate with each possible size of the instances of the problem one P system to solve all instances of that size.

Another possible classification can be considered with respect to the existence or not in the system of a membrane where the input data is introduced before the computation starts. Usually, the semi-uniform solutions are performed by P systems without input, whereas the uniform solutions are performed by P systems with input.

In order to accept or reject an input, it should be enough to read the answer of any computation of the system. Hence, it is necessary to require a condition of confluence in the following sense: every computation of the system is a halting computation, and, on the same input, all computations have the same output. A

*recognizer P system* is a P system with external output such that all computations halt and, only in the last step of the computation, either object **yes** or object **no** (but not both) must be released into the environment.

Recognizer membrane systems without input membrane require the confluence condition: all branches of a computation associated with an instance eventually reach a unique configuration. Using this type of P systems, the decrease in the execution time from exponential to polynomial is achieved, but with the use of an exponential amount of space, this space is created in polynomial time. This is possible in various types of membrane systems through membrane division [77] (repeat the division of membranes in order to obtain  $2^n$  membranes in  $n$  steps), membrane creation [52] (new membranes are produced under the influence of the existing objects in a membrane), string replication [25] (in a rewriting membrane system using string objects, exponentially many strings are generated in linear time), or by using precomputed resources [79] (starting from an arbitrarily large initial membrane structure, without objects placed in its regions, both objects and rules related to a given problem are introduced in a specified membrane).

A construction is semi-uniform if the systems of the family solving the decision problem are constructed starting not from the size of an instance, but from an instance only (for each instance of the problem a P system associated with it is constructed). A semi-uniform solution to SAT problem is presented in [91] and for HPP in [83].

Now, we briefly recall some of the efficient solutions of **NP**-complete problems in the framework of P systems without input, described in a semi-uniform way.

In [5], a linear time solution to SAT through P systems with active mem-

branes without polarizations is presented, but using some membrane rules (merging, separation, and release).

The first efficient solution to HPP by P systems is presented in [59], but using rules for  $d$ -division, with an arbitrary  $d$  (a solution to Vertex Cover is also provided in this paper). This result about HPP was improved in [73] by using only rules for 2-division. Other efficient solutions to SAT and HPP are given in [25], through P systems using string replications, and in [36], through P systems using precomputed resources.

Recognizer P systems with input membrane solve hard problems in a uniform way in the following sense: all instances of a decision problem that have the same size are processed by the same system, to which an appropriate input that depends on the concrete instance is supplied. This method for solving problems provides a general purpose algorithmic solution, meaning that a system constructed to solve an instance of the problem can also be used for solving another instance of the same size.

The first efficient and uniform solutions to numerical **NP**-complete problems were given in [81] where a solution to the Knapsack problem was presented. A uniform solution to Multiset 0 – 1 Knapsack problem is given in the same framework in [72]. For SAT, there are many different types of P systems that solve it in a uniform manner, e.g. in [83]. Different efficient solutions to graph problems (Vertex Cover, Clique) in a uniform way are presented in [7]. Another general problem, the Common Algorithmic Decision Problem, is solved in quadratic time by a uniform construction in [84].

Solutions to **NP**-complete problems are looked for in this framework by making

use of appropriate families of recognizer P systems that can be constructed in a semi-uniform or uniform way. We have discussed here the differences between these constructions, and we have presented a short survey of some solutions known in the current literature of membrane systems.

## CHAPTER 4

### OPERATIONS FROM BRANE CALCULI

#### 4.1 Mate/Drip Operations in P Systems

Membrane computing tries to abstract computing models, in the Turing sense, from the structure and the functioning of the cell, making use especially of automata and language theoretic tools, while brane calculi (introduced in [23]) pay more attention to the biological reality and have as a primary target systems biology. Various operations with membranes appear in both areas, fully idealized in the former area, less idealized in the latter. For instance, an important distinction concerns the role the membranes play in the two fields: separators of compartments in membrane computing, with the computation done inside the regions, and main objects in brane calculi, with the emphasis put on the structure, properties, and evolution of membranes. In particular, in the latter case the membranes are the support of bio-chemistry, with the proteins embedded in them being central to the cell evolution, rather than the chemicals from the compartments.

We start from the four basic operations from brane calculi (*mate*, *drip*, *pino*, *exo*), and we consider them as operations in a P system; hence, they are used in a non-deterministic and maximally parallel manner. With membranes we associate *multisets of proteins*, which are supposed to be placed *on* membranes and visible/accessible from

both sides of them. Different classes of proteins associated to the membranes were considered: the *peripheral* proteins, which are linked only to the external or to the internal side of the membrane (proposed in [38] and also investigated in [19]), and *integral* (or trans-membrane) proteins (considered in [24]), which span the cellular bilayer and thus have part of their molecule on either sides of the membrane. We consider only the latter case, where the proteins are available for the rules placed in the regions inside and outside the membrane.

In the following results, we work only with *mate* and *drip* rules, and that is why we only define here these operations:

$$\text{mate: } [ ]_{ua} [ ]_v \rightarrow [ ]_{uxv},$$

$$\text{drip: } [ ]_{uav} \rightarrow [ ]_{ux} [ ]_v,$$

where  $u, v, x \in P^*$  and  $a \in P$ , for a specified alphabet  $P$  of proteins.

In a *mate/drip* rule, the length  $|uav|$  (the total multiplicity of the multiset represented by  $uav$ ) is called the *weight* of the rule. When using a *mate/drip* rule, the membranes from its left hand side are consumed and the membranes from the right hand side of the rule are produced instead. Similarly, the protein  $a$  specified in the left hand side of rules is consumed, and it is replaced by the multiset  $x$ . All other proteins which mark the membranes that are consumed remain unchanged, and they are transferred to the newly created membranes. In the case of *mate* rules, all proteins are placed on the new membrane; in the case of *drip* rules, the proteins of the old membrane which are not involved in the rule are non-deterministically distributed to the new membranes. Note that the evolution is parallel at the level of membranes,

but sequential at the level of each multiset marking a membrane: at most one protein,  $a$ , evolves by applying a rule, and at most one rule is applied to each membrane in one step (but using a rule is obligatory if this is possible).

**Definition 4.1** A  $P$  system with *mate/drip* rules of degree  $m$  is a device of the form:

$$\Pi = (P, \mu, u_1, u_2, \dots, u_m, R),$$

where:

1.  $P$  is an alphabet (finite and non-empty set) of proteins;
2.  $\mu$  is a membrane structure with at least two membranes,  $m \geq 2$ ;
3.  $u_1, \dots, u_m$  are multisets of proteins placed on membranes of  $\mu$  at the beginning of the computation (we assume that the membranes in  $\mu$  are labeled from 1 to  $m$ ; the skin membrane has label 1 and  $u_1 = \lambda$ );
4.  $R$  is a finite set of *mate/drip* rules, of the forms specified above, using proteins from the set  $P$ .

Note that the skin membrane has no protein associated, because no rule can be applied to it (it only delimits the system from its environment). In each step of a computation, each membrane and each protein can be involved in only one rule; the skin membrane never evolves.

A computation is successful only if it halts, and in the halting configuration, there are only two membranes, the skin and an inner one. The result of a successful computation is the number of proteins which mark the inner membrane in the halting

configuration. We can also take as the result of a computation the vector which describes the multiplicity of objects placed on the inner membrane in the halting configuration (as in [24]). The set of all numbers computed by  $\Pi$  is denoted by  $N(\Pi)$ , and the family of all sets  $N(\Pi)$  computed by P systems  $\Pi$  using at any moment during a halting computation at most  $m$  membranes, and *mate*, *drip* rules of weight at most  $p$ ,  $q$ , respectively, is denoted by  $NOP_m(mate_p, drip_q)$ ; if we consider the result as vectors of numbers, then we obtain  $PsOP_m(mate_p, drip_q)$ .

## 4.2 Computing Power

Here, we consider the *mate/drip* rules from brane calculi and some related operations for the cooperative/non-cooperative evolution of proteins on the membranes without changing the membrane structure:

$$\text{cooev} \quad [ ]_{ua} \rightarrow [ ]_{ux},$$

$$\text{ncooev} \quad [ ]_a \rightarrow [ ]_x,$$

where  $u, v, x \in P^*$  and  $a \in P$ , for a specified alphabet  $P$  of proteins.

In all rules we can also have polarizations, one of  $+$ ,  $-$ ,  $0$ , which can be changed when applying the rule; for instance, we can have a *mate* rule of the form

$$[ ]_{ua}^0 [ ]_v^- \rightarrow [ ]_{uxv}^+.$$

The family of vectors of numbers generated by a P systems with at most  $m$  membranes, using *mate* rule of weight at most  $p$  and *drip* rules of weight at most  $q$ , cooperative evolution rules and all three polarizations is denoted by  $PsOP_m(mate_p, drip_q, coev, 3pol)$ ; when only two polarizations are used we write  $2pol$

instead of  $3pol$ . When one type of rules is not used, then the respective indication ( $mate_p$ ,  $drip_q$ , or  $coev$ ) is omitted.

In [19], it is proven that  $NOP_5(mate_4, drip_4) = NRE$ , by simulating a register machine. This result holds also for vectors of numbers. With a smaller number of membranes and rules of smaller weights, we can generate at least the Parikh images of matrix languages.

**Theorem 4.1**  $PsMAT \subseteq PsOP_3(mate_2, drip_3)$ .

*Proof.* We simulate a matrix grammar  $G = (N_1 \cup N_2 \cup \{S\}, T, S, M)$  without appearance checking in the binary normal form.

We construct the P system with  $mate/drip$  rules

$$\Pi = (P, [ [ ] ], \lambda, X_{init}A_{init}c, R),$$

with the alphabet

$$\begin{aligned} P &= N_1 \cup N_2 \cup \{X_i \mid X \in N_1, 1 \leq i \leq n\} \\ &\cup \{x \mid x \in (N_2 \cup T)^*, |x| \leq 2\} \\ &\cup \{c, c', c'', f, \#\}, \end{aligned}$$

and the rules from the set  $R$  as constructed below.

We start from the initial configuration  $[ [ ]_{X_{init}A_{init}c}]_\lambda$  and, in this way, the matrix of type 1 is already simulated.

For a matrix  $m_i : (X \rightarrow Y, A \rightarrow x)$  of type 2, we introduce the rules from Table 4.1.

Table 4.1 Theorem 4.1 (simulating a matrix of type 2).

Step	Rules	Type
1	$[ ]_{XAc} \rightarrow [ ]_{X_i} [ ]_{Ac}$	drip
2	$[ ]_{cA} [ ]_{X_i} \rightarrow [ ]_{cxX_i}$	mate
3	$[ ]_{X_ic} \rightarrow [ ]_Y [ ]_c$	drip
4	$[ ]_Y [ ]_c \rightarrow [ ]_{Yc}$	mate

We replace the terminal matrices  $(X \rightarrow \lambda, A \rightarrow x)$  with  $(X \rightarrow f, A \rightarrow x)$ . For completing the computation (when  $f$  is introduced), we also consider the rules from Table 4.2.

Table 4.2 Theorem 4.1 (simulating a matrix of type 3).

Step	Rules	Type
1	$[ ]_{cf} \rightarrow [ ]_{c'} [ ]_f$	drip
2	$[ ]_f [ ]_{c'} \rightarrow [ ]_{c'}$	mate
3	$[ ]_{c'} \rightarrow [ ]_{c''} [ ]_\lambda$	drip
4	$[ ]_{c''} [ ]_\lambda \rightarrow [ ]_\lambda$	mate

In order to ensure that the computation halts only when simulating a terminal derivation in  $G$  (in the halting configuration  $[ [ ]_x ]_\lambda$ ), we introduce the following rules:

$$[ ]_c \rightarrow [ ]_{\#\#} [ ]_\lambda,$$

$$[ ]_f \rightarrow [ ]_{\#\#} [ ]_\lambda,$$

$$[ ]_A \rightarrow [ ]_{\#\#} [ ]_\lambda, \text{ for all } A \in N_2,$$

$$[ ]_{\#\#} \rightarrow [ ]_{\#} [ ]_{\#},$$

$$[ ]_{\#} [ ]_{\#} \rightarrow [ ]_{\#\#}.$$

We have the equality  $\Psi_T(L(G)) = Ps(\Pi)$ .

As an open problem remains, the question if this inclusion is proper.

In the following theorems, we also use evolution rules, alone or together with *mate/drip* rules.

**Theorem 4.2**  $PsOP_1(ncoev) = PsREG$ .

*Proof.*  $\subseteq$  Just out together all rules; we get a pure context-free grammar; hence, the generated language is context-free, and its Parikh image is in  $PsREG$ .

$\supseteq$  For a regular grammar  $G = (N, T, S, C)$ , we construct the P system

$$\Pi = (P, [ ], \lambda, S, R),$$

with the alphabet

$$P = N \cup T,$$

and the set  $R$  consisting of the following rules:

$$[ ]_A \rightarrow [ ]_{aB}, \text{ for } A \rightarrow aB \in C,$$

$$[ ]_A \rightarrow [ ]_a, \text{ for } A \rightarrow a \in C,$$

$$[ ]_A \rightarrow [ ]_A, \text{ for } A \in N.$$

**Theorem 4.3**  $PsMAT \subseteq PsOP_1(\text{coev}, 2\text{pol})$ .

*Proof.* For a matrix grammar  $G = (N_1 \cup N_2 \cup \{S\}, T, S, M)$  without appearance checking in the binary normal form, we construct the P system

$$\Pi = (P, [ ], X_{init}A_{init}, R),$$

with the alphabet

$$P = N_1 \cup N_2 \cup \{X_i \mid X \in N_1, 1 \leq i \leq n\} \\ \cup \{x \mid x \in (N_2 \cup T)^*, |x| \leq 2\},$$

and the rules from the set  $R$  as constructed below.

The computation starts from the initial configuration  $[ ]_{X_{init}A_{init}}^0$  and, in this way, the matrix of type 1 is already simulated.

For a matrix  $m_i : (X \rightarrow Y, A \rightarrow x)$  of type 2, we introduce the rules from Table 4.3.

Table 4.3 Theorem 4.3 (simulating a matrix of type 2).

Step	Rules	Type
1	$[ ]_X^0 \rightarrow [ ]_{X_i}^0$	coev
2	$[ ]_{X_iA}^0 \rightarrow [ ]_{X_i x}^+$	coev
3	$[ ]_{X_i}^+ \rightarrow [ ]_Y^0$	coev

If  $Y = \lambda$ , the previous rules simulate a type 3 matrix.

We also introduce the following rules in order to prevent the incorrect halting of the system:

$$[ ]_X^0 \rightarrow [ ]_X^0, \text{ for } X \in N_1,$$

$$[ ]_A^0 \rightarrow [ ]_A^0, \text{ for } A \in N_2.$$

**Theorem 4.4**  $PsOP_4(mate_2, drip_2, coev, 3pol) = PsRE$ .

*Proof.* We start from a matrix grammar with appearance checking in the binary normal form,  $G = (N, T, S, M, F)$ , where  $N = N_1 \cup N_2 \cup \{S, \#\}$ , and we construct the P system

$$\Pi = (P, [ [ [ ] ] ], \lambda, X_{init}A_{init}, c_1, R),$$

with the alphabet

$$\begin{aligned} P &= N_1 \cup N_2 \cup \{X_i \mid X \in N_1, 1 \leq i \leq n\} \\ &\cup \{x \mid x \in (N_2 \cup T)^*, |x| \leq 2\} \\ &\cup \{c_1, \dots, c_5, f, f', \#\}, \end{aligned}$$

and the rules from the set  $R$  as constructed below.

The computation starts from the initial configuration with polarizations  $[ [ ]_{X_{init}A_{init}}^0 [ ]_{c_1}^0 ]_\lambda^0$  and, in this way, the matrix of type 1 is already simulated.

A matrix  $m_i : (X \rightarrow Y, A \rightarrow x)$  of type 2 is simulated by the rules from Table 4.4.

Table 4.4 Theorem 4.4 (simulating a matrix of type 2).

Step	Rules	Type
1	$[ ]_X^0 \rightarrow [ ]_{X_i}^0,$	coev
	$[ ]_{c_1}^0 \rightarrow [ ]_{c_2}^0$	coev
2	$[ ]_{X_i A}^0 \rightarrow [ ]_{X_i x}^+,$	coev
	$[ ]_{c_2}^0 \rightarrow [ ]_{c_3}^0 [ ]_{c_4}^0$	drip
3	$[ ]_{X_i}^+ [ ]_{c_3}^0 \rightarrow [ ]_{X_i}^-,$	mate
	$[ ]_{c_4}^0 \rightarrow [ ]_{c_5}^0$	coev
4	$[ ]_{X_i}^- \rightarrow [ ]_Y^0,$	coev
	$[ ]_{c_5}^0 \rightarrow [ ]_{c_1}^0$	coev

A matrix  $m_i : (X \rightarrow Y, A \rightarrow \#)$  of type 3 is simulated by the rules from Table 4.5.

We replace the terminal matrices  $(X \rightarrow \lambda, A \rightarrow x)$  with  $(X \rightarrow f, A \rightarrow x)$ . For completing the computation (when  $Y = f$  in a type 2 matrix), we use the rules from Table 4.6.

To ensure a correct halting of the system, we also add the rules:

$$[ ]_{\#\#}^0 \rightarrow [ ]_{\#}^0 [ ]_{\#}^0,$$

$$[ ]_{\#}^0 [ ]_{\#}^0 \rightarrow [ ]_{\#\#}^0.$$

In this way we proven that  $PsOP_4(mate_2, drip_2, coev, 3pol) \supseteq PsRE$  and the proof ends.

Table 4.5 Theorem 4.4 (simulating a matrix of type 3).

Step	Rules	Type
1	$[ ]_X^0 \rightarrow [ ]_{X_i}^0,$	coev
	$[ ]_{c_1}^0 \rightarrow [ ]_{c_2}^0$	coev
2	$[ ]_{X_i A}^0 \rightarrow [ ]_{X_i \#\#\#}^0,$	coev
	$[ ]_{c_2}^0 \rightarrow [ ]_{c_3}^0 [ ]_{c_4}^0$	drip
3	$[ ]_{X_i}^0 [ ]_{c_3}^0 \rightarrow [ ]_{X_i}^-,$	mate
	$[ ]_{c_4}^0 \rightarrow [ ]_{c_5}^0$	coev
4	$[ ]_{X_i}^- \rightarrow [ ]_Y^0,$	coev
	$[ ]_{c_5}^0 \rightarrow [ ]_{c_1}^0$	coev

Table 4.6 Theorem 4.4 (simulating the end of computation).

Step	Rules	Type
1	$[ ]_f^0 \rightarrow [ ]_{f'}^0,$	coev
	$[ ]_{c_1}^0 \rightarrow [ ]_{c_2}^0$	coev
2	$[ ]_{f' A}^0 \rightarrow [ ]_{f' \#\#\#}^0,$	coev
	$[ ]_{c_2}^0 \rightarrow [ ]_{c_3}^0 [ ]_{c_4}^0$	drip
3	$[ ]_{f'}^0 [ ]_{c_3}^0 \rightarrow [ ]_{f'}^+,$	mate
	$[ ]_{c_4}^0 \rightarrow [ ]_{c_5}^0$	coev
4	$[ ]_{f'}^+ [ ]_{c_5}^0 \rightarrow [ ]_{f'}^-,$	mate
5	$[ ]_{f'}^- \rightarrow [ ]_\lambda^0$	coev

## CHAPTER 5

### P SYSTEMS WITH PROTEINS ON MEMBRANES

#### 5.1 The Model

In “standard” membrane computing (we refer the interested reader to [78], [79]), one works with multisets of objects placed in the regions delimited by the membranes of (in general, cell-like) membrane structures and the evolution of these objects is based in most cases on multiset rewriting rules and/or on communication rules, e.g., on symport/antiport rules ([75]). However, in biology, many reactions taking place in the compartments of living cells are controlled/catalysed by the proteins embedded in the membrane’s bilayer. For instance, it is estimated that in the animal cells, the proteins constitute about 50% of the mass of the membranes, the rest being lipids and small amounts of carbohydrates. There are several types of such proteins embedded in the membrane of the cell; one simple classification places these proteins into two classes, that of integral proteins (these molecules can “work” inside the membrane as well as in the region outside the membrane), and that of peripheral proteins (macromolecules that can only work in one region of the cell) – see [2]. In the new model, we try to capture features of both these types of proteins.

In turn, in brane calculi introduced in [23], one works only with objects – called proteins – placed on membranes, while the evolution is based on membrane handling

operations, such as exocytosis, phagocytosis, etc. The two approaches are somewhat dual to each other (membrane operations are used also in membrane computing), and it is just natural to combine features from the two areas. Investigations in this direction were already started in [24], using brane calculi operations in a P system framework (multisets of objects placed on membranes and handled only by membrane operations controlled by these objects), as well as in [22] and [28].

We consider a restrictive case, where the “main” information to process is encoded in the multisets from the regions of a P system, but these objects evolve under the control of a bounded number of proteins placed on membranes. Also, the rules we use are very restrictive: move objects across membranes, under the control of proteins, changing or not the objects and/or the proteins during these operations. In some sense, we have an extension of symport/antiport rules, with the mentioning that we always use minimal rules, dealing with only one protein, one object inside the region, and/or one object outside of it.

The control by means of proteins embedded in the membrane bilayer can be used also for more complex symport or antiport rules, maybe obtaining in this case easier universality proofs, but we do not go in this direction (as we have already seen, minimal symport – hence uniport – and minimal antiport lead to universality). Moreover, we can use the proteins not as reactants, as “partners” of the objects which evolve, but as promoters or inhibitors of rules handling objects from the compartments. The essential difference from the case we consider here is that the promoters/inhibitors do not diminish the parallelism; one single promoter/inhibitor can enhance/forbid the use of arbitrarily many rules.

A similar strategy considering operations with multisets controlled by objects placed on membranes was recently followed in [28]; however, with considerable differences from the case considered here, in [28], one provides an exchange between compartment multisets and membrane proteins, but the system can also use operations with membranes, etc.

In the P systems which we consider below, we use two types of objects, *proteins* and usual *objects*; the former are placed **on** the membranes, the latter are placed **in** the regions delimited by membranes. The fact that a protein  $p$  is on a membrane (with label)  $i$  is written in the form  $[_i p]$ . Both the regions of a membrane structure and the membranes can contain multisets of objects and of proteins, respectively. For instance, the expression

$$[_1 p_1^2 p_2 | a^3 b d \ [_2 p_3 | b^5 c d^2 \ ]_2 ]_1$$

indicates that we have a membrane structure with two membranes, labeled 1 and 2, with the inner membrane 2 containing the multiset of objects  $b^5 c d^2$  inside its region and the protein  $p_3$  on it, while in region 1 we have the multiset of objects  $a^3 b d$  and the respective membrane is marked (to follow the terminology from [28]) by the multiset of proteins  $p_1^2 p_2$ .

We consider the following types of rules for handling the objects and the proteins; in all of them,  $a, b, c, d$  are objects,  $p$  is a protein, and  $i$  is a label (“res” stands for “restricted”):

Table 5.1 Restricted rules.

Type	Rule	Effect
1res	$[_i p a \rightarrow [_i p b$	
	$a[_i p  \rightarrow b[_i p $	modify an object, but not move
2res	$[_i p a \rightarrow a[_i p $	
	$a[_i p  \rightarrow [_i p a$	move one object unmodified
3res	$[_i p a \rightarrow b[_i p $	
	$a[_i p  \rightarrow [_i p b$	modify and move one object
4res	$a[_i p b \rightarrow b[_i p a$	interchange two objects
5res	$a[_i p b \rightarrow c[_i p d$	interchange and modify two objects

In all cases above, the protein is not changed; it plays the role of a catalyst assisting the evolution of objects. A generalization is to allow rules of the forms as in Table 5.2 (now, “cp” means “change protein”), where  $p, p'$  are two proteins (possibly equal; if  $p = p'$ , then the rules of type  $cp$  become rules of type  $res$ ).

An intermediate case is considered, whereas of changing proteins, but in a restricted manner, by allowing at most two states for each protein,  $p, \bar{p}$ , and the rules either as in the first table (without changing the protein), or changing from  $p$  to  $\bar{p}$  and back (like in the case of bistable catalysts). Rules with such flip-flop proteins are denoted by  $nff$ ,  $n = 1, 2, 3, 4, 5$  (note that in this case we allow both rules which do not change the protein and rules which switch from  $p$  to  $\bar{p}$  and back).

In both cases of rules for type  $cp$  and for type  $ff$ , we can ask that the proteins are always moved in their complementary state (from  $p$  into  $\bar{p}$  and vice versa). Such

Table 5.2 Change protein rules.

Type	Rule	Effect (besides changing also the protein)
1cp	$[_i p a \rightarrow [_i p' b$	
	$a[_i p  \rightarrow b[_i p' $	modify an object, but not move
2cp	$[_i p a \rightarrow a[_i p' $	
	$a[_i p  \rightarrow [_i p' a$	move one object unmodified
3cp	$[_i p a \rightarrow b[_i p' $	
	$a[_i p  \rightarrow [_i p' b$	modify and move one object
4cp	$a[_i p b \rightarrow b[_i p' a$	interchange two objects
5cp	$a[_i p b \rightarrow c[_i p' d$	interchange and modify two objects

rules are said to be of *pure cp* or *ff* type, and we indicate the use of pure *cp* or *ff* rules by writing *cpp* and *ffp*, respectively.

We can use these rules in devices defined in the same way as the symport/antiport P systems (hence, with the environment containing objects, in arbitrarily many copies each – we need such a supply of objects, because we cannot create objects in the system), where also the proteins present on each membrane are mentioned.

**Definition 5.1** A *P system with proteins on membranes* is a device of the form

$$\Pi = (O, P, \mu, w_1/z_1, \dots, w_m/z_m, E, R_1, \dots, R_m, i_o),$$

where:

1.  $m$  is the degree of the system (the number of membranes);
2.  $O$  is a finite set of objects;
3.  $P$  is a finite set of proteins (with  $O \cap P = \emptyset$ );
4.  $\mu$  is the membrane structure;
5.  $w_1, \dots, w_m$  are the (strings representing the) multisets of objects present in the  $m$  regions of the membrane structure  $\mu$ ;
6.  $z_1, \dots, z_m$  are the multisets of proteins present on the  $m$  membranes of  $\mu$ ;
7.  $E \subseteq O$  is the set of objects present in the environment (in an arbitrarily large number of copies for each);
8.  $R_1, \dots, R_m$  are finite sets of rules associated with the  $m$  membranes of  $\mu$ ;
9.  $i_o$  is the output membrane, an elementary membrane of  $\mu$ .

The rules can be of the forms specified above, and they are used in a non-deterministic and maximally parallel way: in each step, a maximal multiset of rules is used, that is, no rule can be applied to the objects and the proteins which remain unused by the chosen multiset. As usual, each object and each protein can be involved in the application of only one rule, but the membranes are not considered as involved in the rule applications; hence, the same membrane can appear in any number of rules at the same time.

If, at one step, two or more rules can be applied to the same objects and proteins, then only one rule will be non-deterministically chosen. At each step, a

P system is characterized by a configuration consisting of all multisets of objects and proteins present in the corresponding membranes (we ignore the structure  $\mu$ , which will not be changed, and the objects from the environment). For example,  $C = w_1/z_1, \dots, w_m/z_m$  is the initial configuration, given by the definition of the P system. By applying the rules in a non-deterministic and maximally parallel manner, we obtain transitions between the configurations of the system. A finite sequence of configurations is called computation. A computation halts if it reaches a configuration where no rule can be applied to the existing objects and proteins.

Only halting computations are considered successful; thus, a non-halting computation will yield no result. With a halting computation, we associate a result in the form of the multiplicity of objects present in region  $i_o$  in the halting configuration. We denote by  $N(\Pi)$  the set of natural numbers computed in this way by a given system  $\Pi$ . A generalization would be to distinguish the objects and to consider vectors of natural numbers as the result of a computation, but we do not examine this case here.

We denote, in the usual way, by  $NOP_m(\text{pro}_r; \text{list-of-types-of-rules})$  the family of sets of natural numbers  $N(\Pi)$  generated by systems  $\Pi$  with at most  $m$  membranes, using rules as specified in the list-of-types-of-rules, and with at most  $r$  proteins present on a membrane. When parameters  $m$  or  $r$  are not bounded, we use  $*$  as a subscript.

## 5.2 Computational Results for the Generating Mode

Clearly, rules of type 3 are more general than those of type 2, and rules of type 5 are more general than those of type 4; in turn, rules  $cp$  are more general than

rules  $ff$ , which are more general than the rules  $res$ . Finally, non-restricted rules of a given type,  $cp$  or  $ff$ , are more general than using pure rules of the respective types. Thus, all proofs which involve rules of a particular form are also valid for rules of the more powerful form (and this observation will justify the corollaries of the theorems from next sections).

Then, it is easy to see that any rule  $[_i p|a \rightarrow [_i p'|b$  of type  $1cp$  can be simulated, in two steps, by rules of type  $3cp$ :  $[_i p|a \rightarrow \bar{b}[_i \bar{p}]$ ,  $\bar{b}[_i \bar{p}] \rightarrow [_i p'|b$ .

Therefore, the families of sets of numbers generated by systems using **only** one type of rules are included in one another as suggested by the diagrams from Figure 5.1.

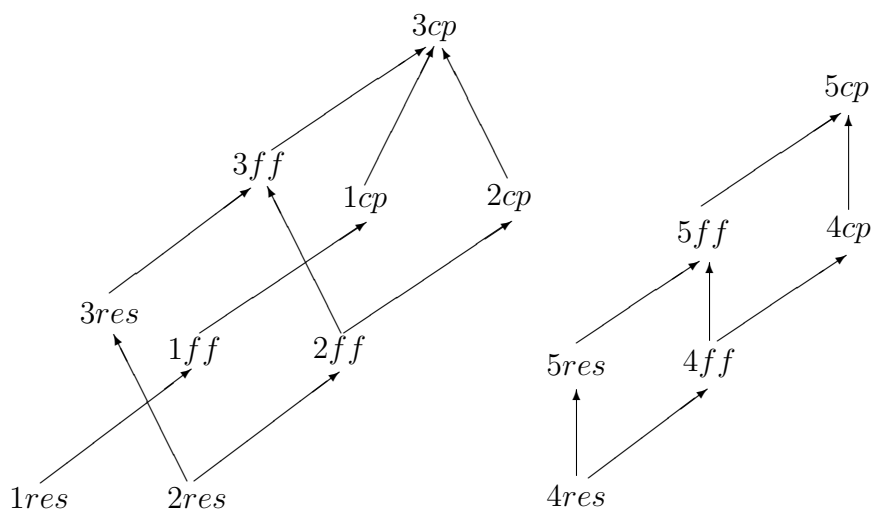


Figure 5.1 Relationships between the types of rules.

The rules of type  $2res$  correspond to uniport rules, while rules of type  $4res$  correspond to minimal antiport rules. It is important however to note that in our case the number of proteins never changes; hence, at a given step the number of rules which can be used is bounded by the number of proteins (hence the parallelism is

restricted in this way).

Fortunately enough, in the proof of the universality of P systems with minimal symport/antiport rules from [6],  $NOP_3(sym_1, anti_1) = NRE$ , the parallelism is also bounded. Consequently, we have the following result:

**Theorem 5.1**  $NOP_3(pro_*; \alpha\beta, \gamma\delta) = NRE$ , for all  $\alpha \in \{2, 3\}$ ,  $\beta, \delta \in \{res, ff, cp\}$ ,  $\gamma \in \{4, 5\}$ .

As expected, when we use the rules from above, where proteins control the operations of passing objects through membranes in the symport/antiport manner, improvements of this result can be obtained (while the proof is much simplified) – see Theorem 5.4.

Not for all combinations of rules we get the universality. For instance, because the rules of forms 1, 4, and 5, of any type *res*, *ff*, *cp*, never change the number of objects. Systems using only rules of these types can compute only sets of numbers consisting of a single number (the initial one present in the system).

Similarly, if we use rules of both types *1res* and *2res*, then we can produce only finite sets of numbers: the system cannot reach a configuration with more objects than the number of objects present inside it in its initial configuration and also halt. The reasoning is the following: in the skin membrane (the only membrane that can bring in the system new objects), we cannot have rules of types *1res* or *2res* bringing in the system objects which are present in the environment in arbitrarily many copies, because such a rule would be applied indefinitely, so the system will never halt. Thus, the only rules that can bring objects from the environment would be rules applied to

objects that were expelled from the system, and the number of such objects is finite (as many as they were present in the system at the beginning of the computation).

### 5.2.1 Universality for One Type of Rules

Now we start to investigate classes of P systems with rules like the ones above, which are computationally complete, able to characterize  $NRE$ , and begin by considering systems in which only one type of rules is used.

**Theorem 5.2**  $NO P_1(\text{pro}_2; 2\text{cpp}) = NRE$ .

*Proof.* We consider a register machine  $M = (m, B, l_0, l_h, R)$  without direct loops in the ADD instructions and we construct the system

$$\Pi = (O, P, [{}_1]_1, \lambda/l_0p, E, R_1, 1),$$

with the following components:

$$O = \{a_r \mid 1 \leq r \leq m\} \cup \{c_l \mid l \in B\} \cup \{c, d\},$$

$$P = \{l, l', l'' \mid l \in B\} \cup \{p, p', p''\} \cup \{p_l \mid l \in B\},$$

$$E = \{a_r \mid 1 \leq r \leq m\} \cup \{c_l \mid l \in B\} \cup \{c, d\},$$

and the following rules in  $R_1$ .

1. For an ADD instruction  $l_1 : (\text{ADD}(r), l_2, l_3) \in R$ , we consider the rules

$$a_r [{}_1 l_1] \rightarrow [{}_1 l_2] a_r,$$

$$a_r [{}_1 l_1] \rightarrow [{}_1 l_3] a_r.$$

When protein  $l_1$  is in membrane 1, one of these two rules is applied non-deterministically. This leads to a copy of object  $a_r$  being brought in region 1 and protein  $l_1$  being changed into  $l_2$  or  $l_3$ .

2. For a SUB instruction  $l_1 : (\text{SUB}(r), l_2, l_3) \in R$  we consider the following rules (we also specify the proteins present on the membrane):

Table 5.3 Theorem 5.2 (simulating a SUB instruction).

Step	Proteins	Rules
1	$l_1$ and $p$	$c_{l_1} [{}_1 l_1] \rightarrow [{}_1 l'_1] c_{l_1}$
2	$l'_1$ and $p$	$[{}_1 p] c_{l_1} \rightarrow c_{l_1} [{}_1 p_{l_1}]$ and $d [{}_1 l'_1] \rightarrow [{}_1 p'] d$
3	$p'$ and $p_{l_1}$	$[{}_1 p_{l_1}] a_r \rightarrow a_r [{}_1 l''_2]$ , if $a_r$ exists, and $c [{}_1 p'] \rightarrow [{}_1 p''] c$
4	$p''$ and ( $p_{l_1}$ or $l''_2$ )	$[{}_1 l''_2] c \rightarrow c [{}_1 l_2]$ or $[{}_1 p_{l_1}] c \rightarrow c [{}_1 l_3]$ , and $[{}_1 p''] d \rightarrow d [{}_1 p]$

When protein  $l_1$  is present on the membrane, we apply the rule  $c_{l_1} [{}_1 l_1] \rightarrow [{}_1 l'_1] c_{l_1}$ , which changes the protein  $l_1$  into  $l'_1$ , and moves one copy of  $c_{l_1}$  inside.

In the second step, we apply both rules at the same time. By applying the first rule, object  $c_{l_1}$  is sent out and protein  $p$  is changed into  $p_{l_1}$ ; while by applying the second rule, object  $d$  is brought inside and protein  $l'_1$  is changed into  $p'$ .

At step 3, we send inside the object  $c$  and we change protein  $p'$  into  $p''$ . If we have at least one copy of object  $a_r$  inside the region, we can also apply the rule  $[{}_1 p_{l_1}] a_r \rightarrow a_r [{}_1 l''_2]$ , which sends out object  $a_r$  and changes the protein  $p_{l_1}$  into  $l''_2$ .

At the last step, we send out object  $d$  while changing the protein  $p''$  into its original form,  $p$ . If at step 3 we have sent out a copy of object  $a_r$ , then we can apply rule  $[_1l_2''|c \rightarrow c[_1l_2|$ , which sends out object  $c$  and changes protein  $l_2''$  into  $l_2$ . If at step 3 we have not applied rule  $[_1p_{l_1}|a_r \rightarrow a_r[_1l_2''|$ , then we still have protein  $p_{l_1}$  on the membrane, and we apply rule  $[_1p_{l_1}|c \rightarrow c[_1l_3|$ , which sends out object  $c$  and changes protein  $p_{l_1}$  into  $l_3$ .

After applying all these rules, we change the protein  $l_1$  into  $l_2$  or  $l_3$ , depending on whether or not we can send out an object  $a_r$ , and this way we simulate the SUB instruction.

3. When the HALT label  $l_h$  is present on the membrane, no further instruction can be simulated, and the number of copies of  $a_1$  in membrane 1 is equal to the value of register 1 of  $M$ .

**Corollary 5.1**  $NOP_m(pro_r; 2cpp) = NOP_m(pro_r; 3cpp) = NRE$ , for all  $m \geq 1$ ,  $r \geq 2$ .

A similar result can be obtained for rules of type  $3ff$  (but without a bound on the number of proteins).

**Theorem 5.3**  $NOP_1(pro_*; 3ffp) = NRE$ .

*Proof.* We consider a register machine  $M = (m, B, l_0, l_h, R)$ . For each label  $l \in B$ , we consider a pair protein – object  $(l, g)$ ; in the proof, we use the same subscripts for  $l$  and  $g$ . The set of all objects  $g$  from a pair  $(l, g)$ , with  $l \in B$ , is denoted by  $C$ .

Let  $w(B)$  be the multiset which contains each  $l \in B$  exactly once. We construct the system

$$\Pi = (O, P, [ ]_1, g_0/w(B)p, E, R_1, 1),$$

with the following components:

$$O = \{g, g', g'', g''', g^{iv} \mid g \in C\} \cup \{a_r \mid 1 \leq r \leq m\},$$

$$P = \{l, l' \mid l \in B\} \cup \{p, p'\},$$

$$E = \{a_r \mid 1 \leq r \leq m\},$$

and the following rules in  $R_1$ .

We start with one copy of each  $l \in B$  present on the membrane, together with the protein  $p$ , and with the pair object of the initial label  $l_0$ , that is  $g_0$ , in the region.

1. For an ADD instruction  $l_1 : (\text{ADD}(r), l_2, l_3) \in R$ , we consider the rules from Table 5.4 (when specifying the proteins, we mention only those of interest for the use of the rules in that step).

We start with  $g_1$  inside and all labels from  $B$  on the membrane, and we end with one of the symbols  $g_2$  or  $g_3$  inside, plus one extra copy of  $a_r$ , and again with all labels on the membrane. At step 2, we apply only two rules in parallel: the first one, to bring one copy of  $a_r$  inside, and only one of the following four rules, depending on the protein present on membrane,  $p$  or  $p'$ , and non-deterministically between symbols  $g_2$  and  $g_3$ .

Table 5.4 Theorem 5.3 (simulating an ADD instruction).

Step	Proteins	Obj. inside	Rules
1	$l_1$ and $(p$ or $p')$	$g_1$	$[_1l_1 g_1 \rightarrow g'_1[_1l'_1 $
2	$l'_1$ and $(p$ or $p')$		$a_r[_1l'_1  \rightarrow [_1l_1 a_r$ and $g'_1[_1p  \rightarrow [_1p' g_2$ or $g'_1[_1p'  \rightarrow [_1p g_2$ or $g'_1[_1p  \rightarrow [_1p' g_3$ or $g'_1[_1p'  \rightarrow [_1p g_3$

2. For a SUB instruction  $l_1 : (\text{SUB}(r), l_2, l_3) \in R$ , we consider the rules from Table 5.5.

We start with object  $g_1$  inside, and at step 1, we send it out modified into  $g'_1$ . The rule  $[_1l_1|g_1 \rightarrow g'_1[_1l'_1|$  also changes the protein  $l_1$  into  $l'_1$ .

At step 2, object  $g'_1$  is moved inside and changed into  $g''_1$ , and the protein  $p$  is changed between its non-primed version and its primed version. If there is at least one copy of  $a_r$  inside, we can also apply the rule  $[_1l'_1|a_r \rightarrow a_r[_1l_1|$ , which sends out  $a_r$  and changes protein  $l'_1$  into  $l_1$ .

We now have two possibilities: one, when at previous step, we have sent out  $a_r$ , and one when we have not. In the first case, at step 3 when we have protein  $l_1$  on the membrane, we change it into  $l'_1$ , and we also send out object  $g''_1$ , modified into  $g'''_2$ .

At step 4, we change back protein  $l'_1$  into  $l_1$ , and we bring in object  $g'''_2$ , modified

Table 5.5 Theorem 5.3 (simulating a SUB instruction).

Step	Proteins	Obj. inside	Rules
1	$l_1$ and $(p$ or $p')$	$g_1$	$[{}_1l_1 g_1 \rightarrow g'_1[{}_1l'_1 $
2	$l'_1$ and $(p$ or $p')$		$[{}_1l'_1 a_r \rightarrow a_r[{}_1l_1 $ , if $a_r$ exists, and $g'_1[{}_1p  \rightarrow [{}_1p' g''_1$ or $g'_1[{}_1p'  \rightarrow [{}_1p g''_1$
3	$(l_1$ or $l'_1)$ and $(p$ or $p')$	$g''_1$	$[{}_1l_1 g''_1 \rightarrow g'''_2[{}_1l'_1 $ or $[{}_1l'_1 g''_1 \rightarrow g_3^{iv}[{}_1l_1 $
4	$(l_1$ or $l'_1)$ and $(p$ or $p')$		$g'''_2[{}_1l'_1  \rightarrow [{}_1l_1 g_2$ or $g_3^{iv}[{}_1p  \rightarrow [{}_1p' g_3$ or $g_3^{iv}[{}_1p'  \rightarrow [{}_1p g_3$

into  $g_2$ . In the second case, at step 3 when we have protein  $l'_1$  on the membrane, we change it into  $l_1$ , and we also send out object  $g''_1$ , modified into  $g_3^{iv}$ . In the last step, we bring in  $g_3^{iv}$ , modified into  $g_3$ , but we do not bring it through the  $l_1$  protein as we need it unchanged, so we bring it using the protein  $p/p'$ .

3. We also consider the following final rules:

$$[{}_1p|g_h \rightarrow g_h[{}_1p'|,$$

$$[{}_1p'|g_h \rightarrow g_h[{}_1p|,$$

which remove the pair object of the HALT label leaving in the system only objects from the output register.

When the computation in  $M$  stops, that is, the pair object of  $l_h$  is introduced in the system, the final rule is used and the computation in  $\Pi$  also stops. The number of copies of  $a_1$  in membrane 1 is equal to the value of register 1 of  $M$ .

As previously mentioned, no other classes of systems using only one type of rules can be universal.

### 5.2.2 Using Two Types of Rules

We pass now to the case when rules of two types (not proven above to be separately universal) are considered.

We start by giving the result mentioned above, improving the result in Theorem 5.1.

**Theorem 5.4**  $NOP_1(pro_2; 2res, 4cpp) = NRE$ .

*Proof.* As in the proof of Theorem 5.2, we consider a register machine  $M = (m, B, l_0, l_h, R)$  without direct loops in the **ADD** instructions.

Given such a register machine, we construct the system

$$\Pi = (O, P, [1]_1, c'c''/l_0p, E, R_1, 1),$$

with

$$O = E \cup \{c', c''\},$$

$$P = \{l, l', l'', l''', l^{iv} \mid l \in B\} \cup \{p\},$$

$$E = \{a_r \mid 1 \leq r \leq m\} \cup \{d\},$$

and the following rules in  $R_1$ .

1. For each ADD instruction  $l_1 : (\text{ADD}(r), l_2, l_3)$  in  $R$ , we introduce in  $R_1$  the rules from next table.

Table 5.6 Theorem 5.4 (simulating an ADD instruction).

Step	Proteins	Rules	Type
1	$l_1$ and $p$	$a_r[_1 l_1   c' \rightarrow c'[_1 l_2   a_r$ or	4cpp
		$a_r[_1 l_1   c' \rightarrow c'[_1 l_3   a_r$	4cpp
2	$(l_2 \text{ or } l_3)$ and $p$	$c'[_1 p   \rightarrow [_1 p   c'$	2res

The fact that the simulation of the ADD instruction is correct is obvious. The rules of type 4cp are in the strong form since the register machine does not have any direct loops. Step 2 above can be simultaneous with step 1 in simulating any other instruction because the protein  $p$  is not involved in step 1.

2. For each SUB instruction  $l_1 : (\text{SUB}(r), l_2, l_3) \in R$  we consider the rules from the next table, where the rules are given according to the steps when they are used.

When protein  $l_1$  is present on the membrane, the 4cp rule  $d[_1 l_1 | c'' \rightarrow c''[_1 l'_1 | d$  is applied like an antiport rule for objects  $d$  and  $c''$ . This rule also changes protein  $l_1$  into  $l'_1$ .

In the second step, object  $c''$  is brought inside and, if a copy of  $a_r$  is present in region, another copy of object  $d$  is interchanged with  $a_r$  and protein  $l'_1$  is modified into  $l''_1$ .

At step 3, if we have protein  $l'_1$  on the membrane (i.e., at step 2, there was no

Table 5.7 Theorem 5.4 (simulating a SUB instruction).

Step	Proteins	Rules	Type
1	$l_1$ and $p$	$d[_1l_1 c'' \rightarrow c''[_1l'_1 d$	4cpp
2	$l'_1$ and $p$	$d[_1l'_1 a_r \rightarrow a_r[_1l''_1 d$ , if $a_r$ exists, and $c''[_1p  \rightarrow [_1p c''$	4cpp 2res
3	$(l'_1$ or $l''_1)$ and $p$	$d[_1l''_1 c'' \rightarrow c''[_1l'''_1 d$ or $d[_1l'_1 c'' \rightarrow c''[_1l^{iv}_1 d$	4cpp 4cpp
4	$(l^{iv}_1$ or $l'''_1)$ and $p$	$d[_1l'''_1 d \rightarrow d[_1l_2 d$ or $d[_1l^{iv}_1 d \rightarrow d[_1l_3 d$ , and $c''[_1p  \rightarrow [_1p c''$	4cpp 4cpp 2res

copy of object  $a_r$  inside the region), we change it into  $l^{iv}_1$ . In the other case, when we have protein  $l''_1$  on the membrane, we change it into  $l'''_1$ . In both cases, we interchange the objects  $d$  and  $c''$ .

In the last step, object  $c''$  is sent inside (in order to be available for simulating future SUB instructions) by applying again the *2res* rule from step 2, and either protein  $l'''_1$  is changed into  $l_2$ , or protein  $l^{iv}_1$  is changed into  $l_3$ . When we change the protein, one copy of object  $d$  from the environment is interchanged with another copy of  $d$  from inside the region.

3. When the label  $l_h$  of the HALT instruction of  $M$  is introduced, we use the rules

$$[_1l_h|x \rightarrow x[_1l_h|, \text{ for } x \in \{c', c'', d\},$$

in order to “clean” the membrane of other objects than  $a_1$ , and the computation halts.

The equality  $N(M) = N(\Pi)$  is obvious, and this completes the proof.

**Corollary 5.2**  $NOP_1(pro_2; \alpha\beta, \gamma) = NRE$ , for all  $\alpha \in \{2, 3\}$ ,  $\beta \in \{res, ff, cp\}$ ,  $\gamma \in \{4cpp, 4cp, 5cpp, 5cp\}$ .

The problem of obtaining a similar result remains open for  $\alpha, \beta$  as in the Corollary 5.2 and  $\gamma \in \{4res, 5res, 4ff, 5ff\}$ .

**Theorem 5.5**  $NOP_1(pro_2; 2res, 1cpp) = NRE$ .

*Proof.* We consider a register machine  $M = (m, B, l_0, l_h, R)$  and we construct the system

$$\Pi = (O, P, [1]_1, \lambda/l_0p, E, R_1, 1),$$

with the following components:

$$O = \{a_r, a'_r, a''_r \mid 1 \leq r \leq m\} \cup \{c, c', d, d'\},$$

$$P = \{l, l', l'', l''' \mid l \in B\} \cup \{p, p'\},$$

$$E = \{a_r \mid 1 \leq r \leq m\} \cup \{c\},$$

and the following rules in  $R_1$ .

1. For an ADD instruction  $l_1 : (ADD(r), l_2, l_3) \in R$ , we consider the rules from Table 5.8.

When protein  $l_1$  is present on membrane 1, a copy of object  $a_r$  from the environment is changed into  $a'_r$ , and protein  $l_1$  gets primed. At step 2, the object

Table 5.8 Theorem 5.5 (simulating an ADD instruction).

Step	Proteins	Rules	Type
1	$l_1$ and $p$	$a_r[{}_1l_1] \rightarrow a'_r[{}_1l'_1]$	1cpp
2	$l'_1$ and $p$	$a'_r[{}_1l'_1] \rightarrow [{}_1l'_1 a'_r$	2res
3	$l'_1$ and $p$	$[{}_1l'_1 a'_r \rightarrow [{}_1l_2 a_r$ or	1cpp
		$[{}_1l'_1 a'_r \rightarrow [{}_1l_3 a_r$	1cpp

$a'_r$  is moved inside and, at step 3, is changed into  $a_r$ , while one of the proteins  $l_2, l_3$  is, non-deterministically, introduced.

2. For a SUB instruction  $l_1 : (\text{SUB}(r), l_2, l_3) \in R$ , we consider the next rules.

Table 5.9 Theorem 5.5 (simulating a SUB instruction).

Step	Proteins	Rules	Type
1	$l_1$ and $p$	$c[{}_1l_1] \rightarrow d[{}_1l''_1]$	1cpp
2	$l''_1$ and $p$	$[{}_1l''_1 a_r \rightarrow [{}_1l'''_1 a''_r$ , if $a_r$ exists, and	1cpp
		$d[{}_1p] \rightarrow d'[{}_1p']$	1cpp
3	$(l''_1$ or $l'''_1)$ and $p'$	$d'[{}_1l''_1] \rightarrow c'[{}_1l_3]$ or	1cpp
		$d'[{}_1l'''_1] \rightarrow c'[{}_1l_2]$ , and	1cpp
		$[{}_1p' a''_r \rightarrow a''_r[{}_1p']$	2res
4	$(l_2$ or $l_3)$ and $p'$	$c'[{}_1p'] \rightarrow c[{}_1p]$	1cpp

When protein  $l_1$  is present on the membrane, we apply the rule  $c[_1l_1] \rightarrow d[_1l_1'']$ , which changes the protein  $l_1$  into  $l_1''$ , and also changes one copy of  $c$  into  $d$ .

In the second step, the object  $d$  gets primed and, if we have at least one copy of object  $a_r$  inside the region, then we can also apply the rule  $[_1l_1''|a_r] \rightarrow [_1l_1''|a_r']$ , which changes  $a_r$  into  $a_r'$  and also the protein  $l_1''$  into  $l_1'''$ .

At step 3, if protein  $l_1'''$  is still present, then we change it into  $l_3$ , in the presence of object  $d'$ , which is transformed into  $c'$ . This corresponds to the case when no copy of  $a_r$  was present. If we have protein  $l_1'''$  present, then also  $a_r''$  should be present; we use the rules  $d'[_1l_1'''] \rightarrow c'[_1l_2]$  and  $[_1p'|a_r''] \rightarrow a_r''[_1p']$ , and in this way the simulation of the SUB rule is completed: the protein  $l_2$  is introduced (while changing  $d'$  into  $c'$ ), and  $a_r''$  is sent to the environment.

The last step (step 4) is just changing back the protein  $p'$  into  $p$  and the object  $c'$  into  $c$ , and thus we reach a configuration similar with the one when we started the simulation, meaning that another simulation of a rule from the register machine can proceed now.

3. When the HALT label  $l_h$  is present on the membrane, no further instruction can be simulated, and the number of copies of object  $a_1$  present in membrane 1 is equal to the value of the output register of  $M$ .

The observation that all rules of type  $1cp$  are pure completes the proof.

**Theorem 5.6**  $NOP_1(\text{pro}_*; 1\text{res}, 2\text{ffp}) = NRE$ .

*Proof.* We consider a register machine  $M = (m, B, l_0, l_h, R)$ . As in the proof of

Theorem 5.3, for each label  $l \in B$ , we consider a pair protein – object  $(l, g)$ ; in the proof, we use the same subscripts for  $l$  and  $g$ . The set of all objects  $g$  from a pair  $(l, g)$ , with  $l \in B$ , is denoted by  $C$ . Let  $w(B)$  be the multiset which contains each  $l \in B$  exactly once. We construct the system

$$\Pi = (O, P, [ ]_1, g_0/w(B)p, E, R_1, 1),$$

with the following components:

$$O = \{a_r \mid 1 \leq r \leq m\} \cup \{g, g', g'' \mid g \in C\},$$

$$P = \{l, l' \mid l \in B\} \cup \{p, p'\},$$

$$E = \{a_r \mid 1 \leq r \leq m\},$$

and the following rules in  $R_1$ .

1. For an **ADD** instruction  $l_1 : (\text{ADD}(r), l_2, l_3) \in R$ , we consider the rules from Table 5.10.

First, we send out the object  $g_1$ , change the protein  $l_1$  into  $l'_1$ , and then we bring one copy of  $a_r$  inside, and we change the protein back into  $l_1$  according with the *2ff* rule. One of the two *1res* rules is non-deterministically applied and object  $g_1$  is changed into  $g_2$  or  $g_3$ . In the last step,  $g_2$  or  $g_3$  is moved inside the membrane through either the protein  $p$  or  $p'$ .

2. For a **SUB** instruction  $l_1 : (\text{SUB}(r), l_2, l_3) \in R$ , we consider the rules from Table 5.11.

First, we change  $g_1$  into  $g'_1$ , and then we send it out changing also the protein  $l_1$  into  $l'_1$ .

Table 5.10 Theorem 5.6 (simulating an ADD instruction).

Step	Proteins	Rules	Type
1	$l_1$ and $(p$ or $p')$	$[_1l_1 g_1 \rightarrow g_1[_1l'_1 $	2ffp
2	$l'_1$ and $(p$ or $p')$	$a_r[_1l'_1  \rightarrow [_1l_1 a_r$ and $g_1[_1p  \rightarrow g_2[_1p $ or $g_1[_1p'  \rightarrow g_2[_1p' $ , or $g_1[_1p  \rightarrow g_3[_1p $ or $g_1[_1p'  \rightarrow g_3[_1p' $	2ffp 1res 1res 1res 1res
3	$l_1$ and $(p$ or $p')$	$g_2[_1p  \rightarrow [_1p' g_2$ or $g_2[_1p'  \rightarrow [_1p g_2$ , or $g_3[_1p  \rightarrow [_1p' g_3$ or $g_3[_1p'  \rightarrow [_1p g_3$	2ffp 2ffp 2ffp 2ffp

At step 3, we change object  $g'_1$  into  $g''_1$  and, if a copy of  $a_r$  is present inside, we send it out and we change back protein  $l'_1$  into  $l_1$ .

At step 4, if protein  $l_1$  is present on the membrane, object  $g''_1$  is changed into  $g_2$ , or, if protein  $l'_1$  is present, object  $g''_1$  is moved inside and the protein is changed into  $l_1$ .

In the last step, object  $g_2$  is brought inside, or object  $g''_1$  is changed into  $g_3$ .

Table 5.11 Theorem 5.6 (simulating a SUB instruction).

Step	Proteins	Rules	Type
1	$l_1$ and $(p$ or $p')$	$[_1l_1 g_1 \rightarrow [_1l_1 g'_1$	1res
2	$l_1$ and $(p$ or $p')$	$[_1l_1 g'_1 \rightarrow g'_1[_1l'_1 $	2ffp
3	$l'_1$ and $(p$ or $p')$	$[_1l'_1 a_r \rightarrow a_r[_1l_1 $ , if $a_r$ exists, and	2ffp
		$g'_1[_1p  \rightarrow g''_1[_1p $ or	1res
		$g'_1[_1p'  \rightarrow g''_1[_1p' $	1res
4	$(l_1$ or $l'_1)$ and $(p$ or $p')$	$g''_1[_1l_1  \rightarrow g_2[_1l_1 $ or	1res
		$g''_1[_1l'_1  \rightarrow [_1l_1 g''_1$	2ffp
5	$l_1$ and $(p$ or $p')$	$g_2[_1p  \rightarrow [_1p' g_2$ or	2ffp
		$g_2[_1p'  \rightarrow [_1p g_2$ , or	2ffp
		$[_1p g''_1 \rightarrow [_1p g_3$ or	1res
		$[_1p' g''_1 \rightarrow [_1p' g_3$	1res

3. We also consider the following final rules:

$$[_1p|g_h \rightarrow g_h[_1p'|,$$

$$[_1p'|g_h \rightarrow g_h[_1p|,$$

which remove the pair object of the **HALT** label.

When the computation in  $M$  stops, that is, the pair object of  $l_h$  is introduced in the system, one of the final rules is used and the computation in  $\Pi$  also stops. The number of copies of  $a_1$  in membrane 1 is equal to the value of register 1 of  $M$ .

**Corollary 5.3**  $NOP_1(pro_*, 3res, 2ffp) = NOP_1(pro_*, 2ffp, 1cp) = NOP_1(pro_*, 1ff, 2ffp) = NRE$ .

In this way, many pairs of types of rules lead to the characterizations of  $NRE$ , but the problem remains open (even for the case of several membranes being used) for the following pairs of types of rules  $(1res, 3res)$ ,  $(1ff, 2res)$ ,  $(1ff, 3res)$ ,  $(2ff, 3res)$ , as well as for pairs involving rules of types  $4\beta$ ,  $5\beta$ , for  $\beta \in \{res, cp, ff\}$ .

### 5.3 One Protein Case

In this section, we consider the weaker case where only one protein is placed on a membrane. Using matrix grammars without appearance checking (as in Definition 2.9), we simulate one-membrane P systems with any type of rules (from the previous defined).

**Theorem 5.7**  $PsOP_1(pro_1; any - rules) \subseteq PsMAT$ .

*Proof.* Let us consider a P system (with one membrane) with proteins on membranes,  $\Pi = (O, P, [ ]_1, p/w, E, R_1, 1)$ , for some  $p \in P$ ,  $w \in O^*$ .

For each symbol  $a \in O$ , we consider the new symbols  $a'$ ,  $a_o$ ,  $a_i$ , and for each  $p \in P$ , we consider the new symbols  $p'$ ,  $p''$ . The set  $\{a' \mid a \in O\}$  is denoted by  $O'$ . For a string  $w \in O^*$ , we denote by  $w_o$ ,  $w_i$  the strings obtained by replacing each symbol  $a \in O$ , which appears in  $w$  by  $a_o$ ,  $a_i$ , respectively. The intuition is that  $a_i$  and  $a_o$  are versions of  $a \in O$  present inside or outside, respectively, the unique membrane of the system, in a string which will describe below a configuration of  $\Pi$ .

We construct the following matrix grammar (without appearance checking)

$$G = (N, T, S, M), \text{ with}$$

$$N = \{a_i, a_o \mid a \in O\} \cup \{p', p'' \mid p \in P\} \cup \{S, X\},$$

$$T = O \cup O' \cup P,$$

and the following matrices in  $M$ :

1.  $(S \rightarrow Xp'w_i)$ .

The idea is to represent a configuration of  $\Pi$  with the multiset  $u \in O^*$  inside, protein  $p$  on the membrane, and a multiset  $v \in (O - E)^*$  outside, in the form  $v_o p' u_i$ ; the objects of  $E$  appearing in the system are explicitly considered in  $u_i$ , but if they are outside, then we ignore them, not considering them in  $v_o$ . That is why we consider the non-terminal  $X$  in the left of the protein-symbol as a “source” of copies of objects in  $E$ .

2. For each rule  $a[_1 p_1 | b \rightarrow c[_1 p_2 | d \in R_1$ , where  $a, b, c, d \in O$ ,  $p_1, p_2 \in P$ , we introduce the matrices:

$$(p'_1 \rightarrow p'_2, a_o \rightarrow c_o, b_i \rightarrow d_i), \text{ if } c \notin E,$$

$$(p'_1 \rightarrow p'_2, a_o \rightarrow \lambda, b_i \rightarrow d_i), \text{ if } c \in E.$$

We simulate the rule of  $R_1$  in an obvious way, not introducing the symbol  $c_o$  if  $c$  is an object from  $E$ . Clearly, this  $5cpp$  rule covers also the other cases: restricted, flip-flop, changing objects and not moving.

3. For each rule  $a[_1 p_1 | \rightarrow [_1 p_2 | b \in R_1$ , where  $a, b \in O$ ,  $p_1, p_2 \in P$ , we introduce

the matrices:

$$(p'_1 \rightarrow p'_2, a_o \rightarrow b_i), \text{ if } a \notin E,$$

$$(p'_1 \rightarrow p'_2, X \rightarrow b_i), \text{ if } a \in E.$$

This time we move inside objects from the environment, modified or not; again, we also cover the case of restricted and of flip-flop rules.

4. For each rule  $[_1p_1|a \rightarrow b[_1p_2] \in R_1$ , where  $a, b \in O$ ,  $p_1, p_2 \in P$ , we introduce the matrices:

$$(p'_1 \rightarrow p'_2, a_i \rightarrow b_o), \text{ if } b \notin E,$$

$$(p'_1 \rightarrow p'_2, a_i \rightarrow \lambda), \text{ if } b \in E.$$

By these matrices, we simulate the rules which move objects out of the membrane. In this way, all types of rules are covered; hence, each computation of  $\Pi$  leading to a configuration of the form  $v[_1p|u]_1$  is simulated in  $G$  by a derivation which reaches the sentential form  $v_o p' u_i$ .

5. We also add to  $M$  the following matrices:

$$(p' \rightarrow p''), \text{ for all } p \in P,$$

$$(p'' \rightarrow p'', X \rightarrow \lambda), \text{ for all } p \in P,$$

$$(p'' \rightarrow p'', a_o \rightarrow a'), \text{ for all } a \in O - E,$$

$$(p'' \rightarrow p'', a_i \rightarrow a), \text{ for all } a \in O,$$

$$(p'' \rightarrow p).$$

At any time during the derivation, we can change the primed protein to a double primed one, and this entails the end of the derivation. In the presence of a double primed protein, we remove the auxiliary non-terminal  $X$ , we change each  $a_o$  into  $a'$  and each  $a_i$  into  $a$ . In this way, a terminal string of  $G$  is obtained.

Each halting computation in  $\Pi$  is simulated by a terminal derivation in  $G$ , but the converse is not true: terminating the derivation in  $G$  as above does not guarantee that the derivation corresponds to a halting computation in  $\Pi$ . Therefore, we have to filter out from  $L(G)$  all strings which describe configurations where at least a rule of  $\Pi$  can be used, and this can be done by intersecting  $L(G)$  with a regular language.

Indeed, the set of configurations where at least a rule can be applied is described by the language

$$L_A = \bigcup_{p \in P} O'^* \{a'pb \mid a[_1p|b \rightarrow c[_1q|d \in R_1, a, b \in O \cup \{\lambda\}, q \in P\} O'^*.$$

(We have covered the case of all types of rules by taking  $a, b \in O \cup \{\lambda\}$ .)

This is a regular language, hence

$$L_{nA} = O'^* P O'^* - L_A,$$

is also regular; the language  $L_{nA}$  contains all strings of the form of the strings generated by  $G$  which describe configurations of  $\Pi$  where no rule from  $R$  can be applied.

This means that the intersection  $L(G) \cap L_{nA}$  contains exactly the terminal strings generated by  $G$  which correspond to halting computations in  $\Pi$ . Because  $MAT$  is closed under intersection with regular languages, it follows that this language is in  $MAT$ .

What remains to do is to remove the symbols which do not participate in the output

of a computation in  $\Pi$ , and this can be done by the morphism  $h : (O \cup P \cup O')^* \rightarrow O^*$  defined by  $h(\alpha) = \lambda$ , for  $\alpha \in P \cup O'$ , and  $h(a) = a$ , for  $a \in O$ . The family  $MAT$  is closed under morphisms; thus, the language  $L = h(L(G) \cap L_{nA})$  is a matrix one.

Consequently,  $Ps(\Pi) = \Psi_O(L)$ , hence  $Ps(\Pi) \in PsMAT$ , and this concludes the proof.

Therefore, using only one protein restricts the computing power of P systems with proteins on membranes (such systems are no longer universal), but two proteins (Theorems 5.2, 5.5) or more (Theorems 5.3, 5.6) lead to computationally complete systems.

Theorem 5.7 answers partially a question formulated to us by prof. Jürgen Dassow, during the 10th International Conference on Developments in Language Theory, Santa Barbara, CA, USA, June 2006.

#### 5.4 Accepting and Computing Systems

Besides the generative approach, there are two other related ways of using a P system: in the *accepting* mode and in the *computing* mode. In both cases, an input is provided to the system, depending on the system's type. For instance, in a symbol-object P system, besides the initial multisets present in the regions of the membrane structure, we can introduce a multiset  $w_0$  in a specified region by adding the objects of  $w_0$  to the objects present in that region. In the string case, a string can be added, possibly inserted in one of the existing strings. The computation proceeds, and if it halts, then we say that the input is accepted (or recognized).

In the computing mode, we do not only have to halt, but we also collect an output, from a specified output region, internal to the system or the environment.

An important distinction appears between systems which behave deterministically and those which work in a non-deterministic way. Such a distinction does not make much sense in the generative mode, especially if only halting computations provide a result at their end: such a system can generate only a single result. In the case of computing functions or solving decidability problems, the determinism is obligatory.

**Theorem 5.8**  $NO P_1^{acc}(pro_2; 2cpp) = NRE$ .

*Proof.* We consider a deterministic register machine  $M = (m, B, l_0, l_h, R)$  without direct loops in the ADD instructions and with the input placed on the first register. To simulate the register  $M$ , we construct the following system:

$$\Pi = (O, P, [1]_1, d/l_0^{od}p, E, R_1, 1)$$

with the following components

$$\begin{aligned} O &= \{a_r \mid 1 \leq r \leq m\} \cup \{c_l \mid l \in B\} \cup \{c, d\}, \\ P &= \{l, l', l'' \mid l \in B\} \cup \{p, p', p''\} \cup \{p_l \mid l \in B\} \cup \{l_0^{od}, l_0^{ev}\}, \\ E &= \{a_r \mid 1 \leq r \leq m\} \cup \{c_l \mid l \in B\} \cup \{c\}. \end{aligned}$$

and the following rules in  $R_1$ .

First, we need to obtain the integer  $x$  as input by bringing inside the membrane  $x$  copies of object  $a_1$ . In a sequence of odd and even steps, we chose non-

deterministically between a rule that sends in another copy of  $a_1$  and a rule that stops the input phase and changes the protein to  $l_0$  (see Table 5.12).

Table 5.12 Theorem 5.8 (simulating the input).

Step	Proteins	Rules
odd	$l_0^{od}$ and $p$	$[_1 l_0^{od}   d \rightarrow d[_1 l_0  $ or $a_1[_1 l_0^{od}   \rightarrow [_1 l_0^{ev}   a_1$
even	$l_0^{ev}$ and $p$	$[_1 l_0^{ev}   d \rightarrow d[_1 l_0  $ or $a_1[_1 l_0^{ev}   \rightarrow [_1 l_0^{od}   a_1$

1. For an ADD instruction  $l_1 : (\text{ADD}(r), l_2) \in R$ , we consider the rule:

$$a_r[_1 l_1 | \rightarrow [_1 l_2 | a_r$$

When protein  $l_1$  is in membrane 1, a copy of object  $a_r$  is brought in region 1 and protein  $l_1$  is changed into  $l_2$ .

2. A SUB instruction  $l_1 : (\text{SUB}(r), l_2, l_3) \in R$  is simulated in the same manner as in Theorem 5.2, Table 5.3.
3. When the HALT label  $l_h$  is present on the membrane, no further instruction can be simulated and the system halts, meaning that number  $x$  is accepted (if the computation does not stop, input  $x$  is not accepted by  $\Pi$ ).

With similar constructions, we can obtain P systems working in accepting mode from the results for the generative mode obtained in Theorems 5.3, 5.5, and 5.6.

## 5.5 A Small Universal P System

There are many results dealing with the descriptive complexity of P systems. In most of the cases, size parameters (as the number of objects, the number of membranes, the number of rules per membrane, the size of rules, the number of objects, etc.) are investigated. In this section, we consider another complexity parameter, the number of rules, for the case of P systems with proteins on membranes. There are a few results of universal P systems with minimal number of rules and we briefly mention them here.

In [85], a particular class of tissue P systems, splicing tissue P systems, having only 8 rules, is proven to be universal. Another result characterizes the P systems with symport/antiport rules and the proof is based on the constructions from [55], which show that it is possible to obtain a universal register machine with 8 registers and 32 instructions of three types (addition, subtraction, and test-for-zero). In this case, a universal register machine is simulated by a P system with only one membrane and 44 antiport rules ([35]).

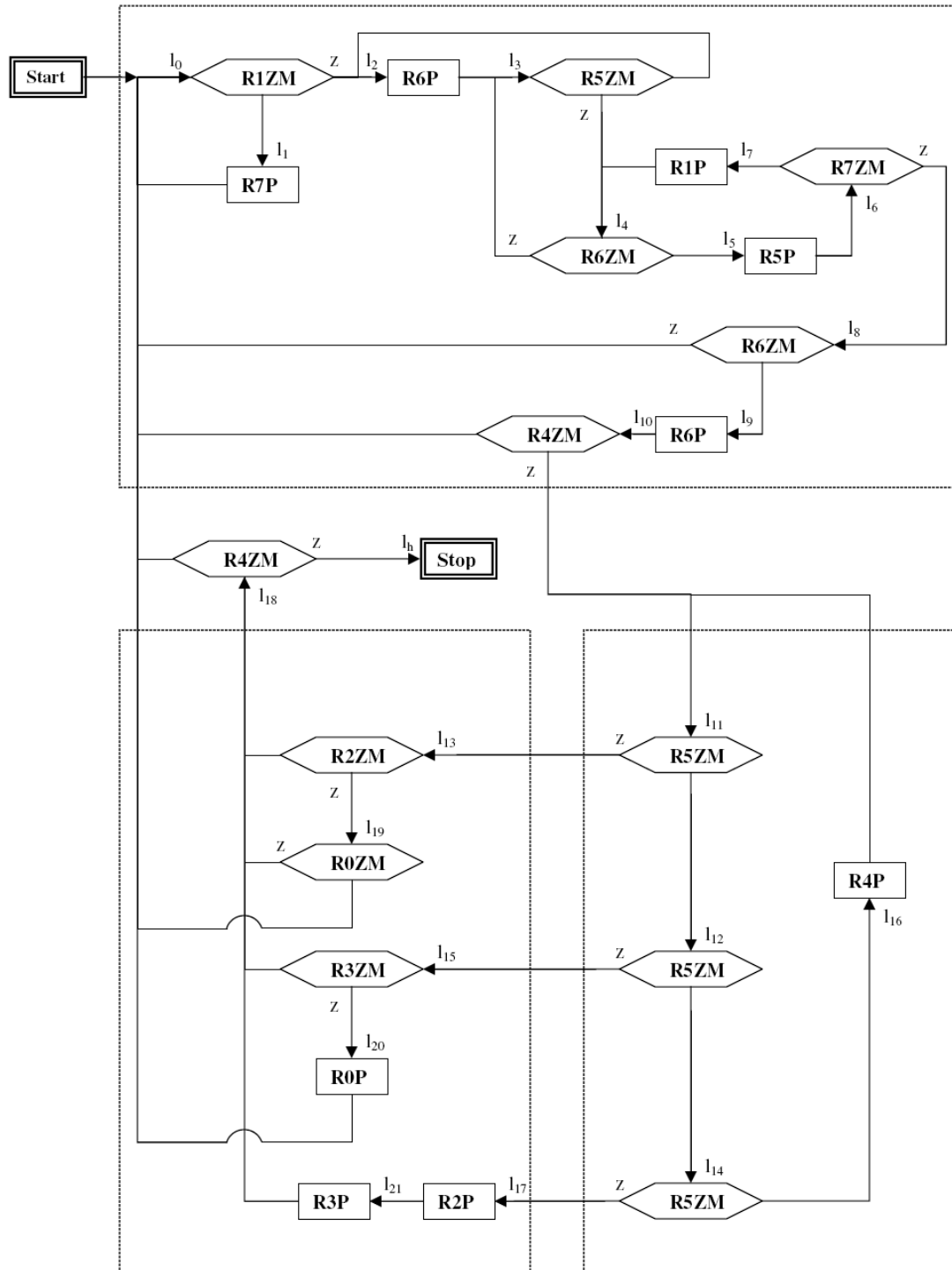
In the following, we recall some concepts and results concerning register machines from [55] and [67]. Various classes of register machines exist, which differ from each other by the form of their instructions. In this section, we work with deterministic register machines which consist of a finite number of registers,  $R_0, \dots, R_{m-1}$  and a finite set of operations of the forms (with the notations from [55]):

1.  $[RiP]$  – add 1 to the content of register  $R_i$  (similar with  $ADD(i)$ );
2.  $[RiM]$  – subtract 1 from the content of register  $R_i$  if it is a positive value (if  $R_i$

is empty, its value remains unchanged);

3.  $\langle Ri \rangle$  – check whether or not the value of register  $R_i$  is positive;
4.  $\langle RiZM \rangle$  – test whether the value of register  $R_i$  is positive or not and subtract 1 from the content of  $R_i$  in the first case (similar with **SUB(i)**).

The power and the efficiency of a register machine depends on the type of instructions that are used. The main result from [55] is the construction of  $U_{32}$ , the universal register machines with 32 instructions of forms  $[RiP]$ ,  $\langle Ri \rangle$ , and  $[RiM]$  (the halting operation is not counted in this case). Using different types of instructions, this result can be improved, and we refer to  $U_{22}$ , the universal register machines with 22 instructions of forms  $[RiP]$  and  $\langle RiZM \rangle$ , which has the same instructions as in Definition 2.8, for the deterministic case. The functioning of  $U_{22}$  is depicted in the scheme from Figure 5.5, following the notations and the construction of  $U_{32}$  from [55].

Figure 5.2 The universal register machine  $U_{22}$ .

The previous universal register machines is defined as  $U_{22} = (8, B, l_0, l_h, R)$ , where  $B = \{l_0, l_1, \dots, l_{21}, l_h\}$  is the set of labels and  $R$  is the set of instructions presented in Figure 5.5 (9 additions, 13 subtractions, and one halting instruction).

$l_0 : (\text{SUB}(1), l_1, l_2),$	$l_1 : (\text{ADD}(7), l_0),$
$l_2 : (\text{ADD}(6), l_3),$	$l_3 : (\text{SUB}(5), l_2, l_4),$
$l_4 : (\text{SUB}(6), l_5, l_3),$	$l_5 : (\text{ADD}(5), l_6),$
$l_6 : (\text{SUB}(7), l_7, l_8),$	$l_7 : (\text{ADD}(1), l_4),$
$l_8 : (\text{SUB}(6), l_9, l_0),$	$l_9 : (\text{ADD}(6), l_{10}),$
$l_{10} : (\text{SUB}(4), l_0, l_{11}),$	$l_{11} : (\text{SUB}(5), l_{12}, l_{13}),$
$l_{12} : (\text{SUB}(5), l_{14}, l_{15}),$	$l_{13} : (\text{SUB}(2), l_{18}, l_{19}),$
$l_{14} : (\text{SUB}(5), l_{16}, l_{17}),$	$l_{15} : (\text{SUB}(3), l_{18}, l_{20}),$
$l_{16} : (\text{ADD}(4), l_{11}),$	$l_{17} : (\text{ADD}(2), l_{21}),$
$l_{18} : (\text{SUB}(4), l_0, l_h),$	$l_{19} : (\text{SUB}(0), l_0, l_{18}),$
$l_{20} : (\text{ADD}(0), l_0),$	$l_{21} : (\text{ADD}(3), l_{18}),$
$l_h : \text{HALT.}$	

Figure 5.3 The instructions of the universal register machine  $U_{22}$ .

If we consider the P system with proteins on membranes and rules of type  $2cpp$ , from Theorem 5.2, and we suppose that it simulates the register machine  $U_{22}$ , we obtain a universal P system with 113 rules ( $9 \cdot 1 + 13 \cdot 8 = 113$ ).

This result can be improved if we consider new rules for two consecutive instructions ADD--SUB. For instance, instead of simulating separately the instructions  $l_5$  and  $l_6$ , we can use the rules from Table 5.13 and simulate them in the same time.

Table 5.13 The simulation of  $l_5 : (\text{ADD}(5), l_6)$  and  $l_6 : (\text{SUB}(7), l_7, l_8)$ .

Step	Proteins	Rules
1	$l_5$ and $p$	$c_{l_5}[{}_1l_5  \rightarrow [{}_1l'_5 c_{l_5}$
2	$l'_5$ and $p$	$[{}_1p c_{l_5} \rightarrow c_{l_5}[{}_1pl_5 $ and $a_5[{}_1l'_5  \rightarrow [{}_1p' a_5$
3	$p'$ and $pl_5$	$[{}_1pl_5 a_7 \rightarrow a_7[{}_1l''_7 $ , if $a_7$ exists, and $c[{}_1p'  \rightarrow [{}_1p c$
4	$pl_5$ or $l''_7$	$[{}_1l''_7 c \rightarrow c[{}_1l_7 $ or $[{}_1pl_5 c \rightarrow c[{}_1l_8 $

The same argument can be applied for the sequence of instructions  $l_9, l_{10}$ . We finally get a universal P system with proteins on membranes and 109 rules of type  $2cpp$  ( $7 \cdot 1 + 11 \cdot 8 + 2 \cdot 7 = 109$ ).

## 5.6 Solving SAT in Polynomial Time

In this chapter, we propose a way to solve NP-complete problems by simulating the cell replication machinery. Since the cell division is usually linked to a protein receptor on the plasma membrane, we are modeling this process with a sequence of steps: a rule simulating the binding of the signaling molecule to its corresponding

receptor will be simulated, and then the bound receptor is viewed as a new protein that starts the division process for the cell.

Using membrane division rules, we are able to solve hard problems (**NP**-complete) such as SAT in polynomial time. Several such results have been obtained recently (see, e.g., [3], [61]), all using the trade-off between space and time made possible by the membrane division rules. Our approach is novel as it refers to the systems in which the parallelism is restricted by the number of proteins embedded in the membranes. Even in this case we are able to obtain fast solutions for SAT. Once the biology research gives way to the manipulation of cell division, we believe that such an approach could be both feasible and energy efficient, thus being the best approach in solving computationally hard problems.

Satisfiability (SAT) is the problem of deciding whether a boolean formula in propositional logic has an assignment that evaluates to true. SAT occurs as a problem and as a tool in applications, and it is considered a fundamental problem in theory, since many problems can be reduced to it. Traditional methods treat SAT as a discrete decision problem.

We assume that all SAT instances are in *conjunctive normal form*, i.e., the conjunction of clauses, where each clause is a disjunction of variables or of their negation. We may write an instance  $\gamma$ , with  $n$  variables, in conjunctive normal form using  $m$  clauses, as follows:  $\gamma = c_1 \wedge c_2 \wedge \dots \wedge c_m$  and  $c_i = y_{i,1} \vee y_{i,2} \vee \dots \vee y_{i,k_i}$ , where  $y_{i,j} \in \{x_l, \neg x_l \mid 1 \leq l \leq n\}$ , for  $1 \leq j \leq k_i, 1 \leq i \leq m$ .

**Example 5.1** For  $n = 3$  variables, we may have the instance  $\gamma = c_1 \wedge c_2$  with  $m = 2$

clauses, where  $c_1 = y_{1,1} \vee y_{1,2} \vee y_{1,3}$ ,  $c_2 = y_{2,1}$  and  $y_{1,1} = x_1$ ,  $y_{1,2} = x_2$ ,  $y_{1,3} = \neg x_3$ ,  $y_{2,1} = \neg x_2$ . If  $(x_1, x_2, x_3) = (0, 0, 0)$ , we have  $c_1 = 1$  and  $c_2 = 1$ , thus  $\gamma = 1$ .

In order to solve SAT using P systems with proteins on membranes, we will need to encode the instance to be solved by the system using multisets of objects (since in the P system one cannot have an order imposed on the objects such that they become strings). A solution to the encoding issue is given below and will be used in the following construction.

To encode an instance  $\gamma$ , we use the following notations.

$$\text{code}(\gamma) = \text{code}(c_1)\text{code}(c_2) \dots \text{code}(c_m), \text{ and } \text{code}(c_i) = \alpha_{i1}\alpha_{i2} \dots \alpha_{in},$$

with

$$\alpha_{ij} = \begin{cases} b_{i,j}, & \text{if } x_j \text{ appears in } c_i; \\ b'_{i,j}, & \text{if } \neg x_j \text{ appears in } c_i; \\ d_{i,j}, & \text{if } x_j \text{ and } \neg x_j \text{ do not appear in } c_i, \text{ for } 1 \leq i \leq m, 1 \leq j \leq n. \end{cases}$$

**Example 5.2** For instance, if we have  $\gamma$  as in Example 5.1, we obtain the following when using the encoding idea given above:  $\text{code}(c_1) = b_{1,1}b_{1,2}b'_{1,3}$ ,  $\text{code}(c_2) = d_{2,1}b'_{2,2}d_{2,3}$ , and  $\text{code}(\gamma) = b_{1,1}b_{1,2}b'_{1,3}d_{2,1}b'_{2,2}d_{2,3}$ .

In addition to the rules from Section 5.1, we need to define a new division rule in the context of P systems with proteins on membranes. To *divide a membrane*, we use the following type of rule, where  $p, p', p''$  are proteins (possible equal):

$$[_i p | ]_i \rightarrow [_i p' | ]_i [_i p'' | ]_i$$

The membrane  $i$  is assumed not to have any polarization and it can be non-elementary.

The rule does not change the membrane label  $i$  and instead of one membrane, at next

step we will have two membranes with the same label  $i$  and the same contents, objects and/or other membranes (although the rule specifies only the proteins involved).

To be consistent in notation, in the following construction we denote by  $a[_i p | b]_i$  the presence of object  $a$  outside of membrane  $i$ , object  $b$  inside of it, and protein  $p$  on membrane  $i$ .

We can now pass to the construction for solving SAT using membrane division; before doing so, we state some basic observations. A clause is satisfied if at least one of the positive variables contained in the clause is assigned the value *true* or a negated variable is assigned the value *false*. If a clause is not satisfied by one variable (i.e. a positive variable with the assignment *false* or a negated variable assigned the value *true*), then we will move to the next variable in order and check that one whether it satisfies the clause. If we reach the  $n^{\text{th}}$  variable and it still does not satisfy the clause, then the particular truth assignment does not satisfy the whole instance  $\gamma$ . On the other hand, as soon as we satisfy a clause  $i$  by the variable  $j$ , we move immediately to the clause  $i + 1$  and variable 1 to continue this process. When reaching and satisfying the last clause (the clause  $m$ ), we know that the instance  $\gamma$  is satisfied by the current truth assignment.

We start with an instance  $\gamma$  of SAT, with  $n$  variables and  $m$  clauses, encoded as above into  $code(\gamma)$  and placed into the input membrane. We construct the P system with proteins on membranes and membrane division

$$\Pi = (O, P, \mu, w_1/z_1, \dots, w_5/z_5, E, R_1, \dots, R_5),$$

where

$$\begin{aligned}
O &= \{d, e, f, g, g', \mathbf{yes}, \mathbf{no}\} \cup \{a_i, f_i, t_i \mid 1 \leq i \leq n\} \\
&\cup \{b_{i,j}, b'_{i,j}, d_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}, \\
P &= \{p, p', p'_n, r_0\} \cup \{q_i \mid 1 \leq i \leq n+2\} \cup \{p_i \mid -(3n+2) \leq i \leq n\} \\
&\cup \{p_i^t, p_i^f \mid 1 \leq i \leq n\} \\
&\cup \{c_i \mid 0 \leq i \leq 2nm+5n+8\} \cup \{r_{i,j}, r'_{i,j}, r''_{i,j} \mid 1 \leq i \leq m+1, 1 \leq j \leq n+1\}, \\
E &= \emptyset, \quad \mu = [{}_1[{}_2[{}_3 \ ]_3]_2[{}_4 \ ]_4[{}_5 \ ]_5]_1, \\
w_1 &= dg, \quad w_2 = a_1 a_2 \dots a_n d, \quad w_3 = \mathit{code}(\gamma) f, \quad w_4 = e t_1 t_2 \dots t_n f_1 f_2 \dots f_n, \quad w_5 = g, \\
z_1 &= p, \quad z_2 = p_{-(3n+2)}, \quad z_3 = r_0, \quad z_4 = q_1, \quad z_5 = c_0, \\
R_1 &= \{[{}_1 p | \mathbf{yes}]_1 \rightarrow \mathbf{yes} [{}_1 p' | ]_1, [{}_1 p | g']_1 \rightarrow \mathbf{no} [{}_1 p | ]_1\}, \\
R_2 &= \{d [{}_2 p_i | d]_2 \rightarrow d [{}_2 p_{i+1} | d]_2 \mid -(3n+2) \leq i \leq -1\} \\
&\cup \{[{}_2 p_i | ]_2 \rightarrow [{}_2 p_{i+1}^t | ]_2 [{}_2 p_{i+1}^f | ]_2 \mid 0 \leq i \leq n-1\} \\
&\cup \{t_i [{}_2 p_i^t | a_i]_2 \rightarrow a_i [{}_2 p_i | t_i]_2, f_i [{}_2 p_i^f | a_i]_2 \rightarrow a_i [{}_2 p_i | f_i]_2 \mid 1 \leq i \leq n\} \\
&\cup \{e [{}_2 p_n | ]_2 \rightarrow [{}_2 p_n | e]_2, [{}_2 p_n | \mathbf{yes}]_2 \rightarrow \mathbf{yes} [{}_2 p'_n | ]_2\}, \\
R_3 &= \{e [{}_3 r_0 | ]_3 \rightarrow [{}_3 r_{1,1} | e]_3, [{}_3 r_{m+1,1} | f]_3 \rightarrow \mathbf{yes} [{}_3 r_{m+1,1} | ]_3\} \\
&\cup \{[{}_3 r_{i,j} | d_{i,j}]_3 \rightarrow d_{i,j} [{}_3 r'_{i,j+1} | ]_3, d_{i,j} [{}_3 r'_{i,j+1} | ]_3 \rightarrow [{}_3 r_{i,j+1} | d_{i,j}]_3, \\
&\quad t_j [{}_3 r_{i,j} | b_{i,j}]_3 \rightarrow b_{i,j} [{}_3 r'_{i+1,j} | t_j]_3, [{}_3 r'_{i+1,j} | t_j]_3 \rightarrow t_j [{}_3 r_{i+1,1} | ]_3, \\
&\quad f_j [{}_3 r_{i,j} | b'_{i,j}]_3 \rightarrow b'_{i,j} [{}_3 r'_{i+1,j} | f_j]_3, [{}_3 r'_{i+1,j} | f_j]_3 \rightarrow f_j [{}_3 r_{i+1,1} | ]_3, \\
&\quad f_j [{}_3 r_{i,j} | b_{i,j}]_3 \rightarrow b_{i,j} [{}_3 r''_{i,j} | f_j]_3, [{}_3 r''_{i,j} | f_j]_3 \rightarrow f_j [{}_3 r_{i,j+1} | ]_3, \\
&\quad t_j [{}_3 r_{i,j} | b'_{i,j}]_3 \rightarrow b'_{i,j} [{}_3 r''_{i,j} | t_j]_3, [{}_3 r''_{i,j} | t_j]_3 \rightarrow t_j [{}_3 r_{i,j+1} | ]_3,
\end{aligned}$$

$$\begin{aligned} & [{}_3r_{i,j}|t_j]_3 \rightarrow t_j[{}_3r_{i,j+1}]_3, [{}_3r_{i,j}|f_j]_3 \rightarrow f_j[{}_3r_{i,j+1}]_3 \\ & | \text{ for } 1 \leq i \leq m, 1 \leq j \leq n \}, \end{aligned}$$

$$\begin{aligned} R_4 &= \{ [{}_4q_i]_4 \rightarrow [{}_4q_{i+1}]_4 [{}_4q_{i+1}]_4 \mid 1 \leq i \leq n+1 \} \cup \{ [{}_4q_{n+2}|e]_4 \rightarrow e[{}_4q_{n+2}]_4 \} \\ &\cup \{ [{}_4q_{n+2}|t_i]_4 \rightarrow t_i[{}_4q_{n+2}]_4, [{}_4q_{n+2}|f_i]_4 \rightarrow f_i[{}_4q_{n+2}]_4 \mid 1 \leq i \leq n \}, \\ R_5 &= \{ g[{}_5c_i|g]_5 \rightarrow g[{}_5c_{i+1}|g]_5 \mid 0 \leq i \leq 2nm+5n+5 \} \\ &\cup \{ g[{}_5c_{2nm+5n+7}|g]_5 \rightarrow g'[{}_5c_{2nm+5n+8}|g']_5 \}. \end{aligned}$$

The rules that are used by the system  $\Pi$  are of one of the forms *3res*, *2cp*, *5cp*, or membrane division. Note that on each membrane in the system we have only one protein. Initially, the environment is empty and will be used to receive the output, the answer **yes** or **no** (no other objects are sent out in the environment during the computation).

### 1. Preliminary Phase

We start by generating  $2^{n+1}$  copies of  $t_i$  and  $f_i$ , for  $1 \leq i \leq n$ , in region 4. In the first  $n+1$  steps, we apply the following membrane division rules:

$$[{}_4q_i]_4 \rightarrow [{}_4q_{i+1}]_4 [{}_4q_{i+1}]_4, 1 \leq i \leq n+1.$$

In the initial configuration, we have protein  $q_1$  on membrane 4, and after applying the membrane division rules, in the first  $n+1$  steps, we get protein  $q_{n+2}$  on all  $2^{n+1}$  membranes labeled 4. Now, we can send out, to membrane 1, all objects from the elementary membranes 4, in  $2n+1$  steps, by applying the following *3res* rules:

$$[{}_4q_{n+2}|t_i]_4 \rightarrow t_i[{}_4q_{n+2}]_4,$$

$$[{}_4q_{n+2}|f_i]_4 \rightarrow f_i[{}_4q_{n+2}|]_4,$$

$$[{}_4q_{n+2}|e]_4 \rightarrow e[{}_4q_{n+2}|]_4, \quad 1 \leq i \leq n.$$

In parallel with these rules, in the first  $3n + 2$  steps, we apply the following *5cp* rule in membrane 2:

$$d[{}_2p_i|d]_2 \rightarrow d[{}_2p_{i+1}|d]_2, \quad -(3n + 2) \leq i \leq -1.$$

## 2. Generating Truth-Assignments Phase

When protein  $p_0$  is present on membrane 2, we start generating truth-assignments. The following sequence of rules is applied, and after  $2n$  steps, we get  $2^n$  membranes labeled 2, all having similar contents: the initial membrane 3 and the multiset of objects  $x_1x_2 \dots x_nd$ , where  $x_i \in \{t_i, f_i\}$ , for  $1 \leq i \leq n$ .

$$[{}_2p_i|]_2 \rightarrow [{}_2p_{i+1}^t|]_2 [{}_2p_{i+1}^f|]_2, \quad 0 \leq i \leq n - 1,$$

$$t_i[{}_2p_i^t|a_i]_2 \rightarrow a_i[{}_2p_i|t_i]_2, \quad f_i[{}_2p_i^f|a_i]_2 \rightarrow a_i[{}_2p_i|f_i]_2, \quad 1 \leq i \leq n.$$

So we are now  $5n + 2$  steps from the start of the simulation. We can now check the clauses, starting with the first one. The computation will take place in region 3, where we have the input,  $code(\gamma)$ . At this moment, we have in the membranes labeled 2 all the possible truth-assignments for the  $n$  boolean variables appearing in  $\gamma$ . On the membrane 3, we currently have the protein  $r_0$ . We start checking each clause by changing the protein (which will be some variant of  $r$ ) on membrane 3. We change the protein  $r_{i,j}$  according to the  $i^{th}$  clause and the  $j^{th}$  variable that we check. In order to have clause  $c_i$  satisfied,

we need at least one variable  $y_{i,j}$  present in  $c_i$  to be true; for  $\gamma$  to be satisfied, we need all clauses to be true.

### 3. *Checking Phase*

When we finish generating truth-assignments in region 2, we have protein  $p_n$  on membrane 2, and in 2 steps, we start the checking phase by moving the object  $e$  from region 1 to region 2 and then into region 3:

$$e[{}_2p_n|]_2 \rightarrow [{}_2p_n|e]_2,$$

and then

$$e[{}_3r_0|]_3 \rightarrow [{}_3r_{1,1}|e]_3.$$

Now we start a sequence of pairs of steps, an even step followed by an odd one, and so on. At each moment, there is one protein on membrane 3 that gets primed (or double primed) in the even steps and then lose the prime in the odd steps.

If  $x_j$  and  $\neg x_j$  are not present in  $c_i$ , in the even steps we apply the rules:

$$[{}_3r_{i,j}|d_{i,j}]_3 \rightarrow d_{i,j}[{}_3r'_{i,j+1}|]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

In the odd steps the following rules are used and we move to the next variable to check:

$$d_{i,j}[{}_3r'_{i,j+1}|]_3 \rightarrow [{}_3r_{i,j+1}|d_{i,j}]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

If  $x_j$  is present and it is true, then clause  $c_i$  is satisfied, and we move to the next clause. In the even steps the following rules are used:

$$t_j[{}_3r_{i,j}|b_{i,j}]_3 \rightarrow b_{i,j}[{}_3r'_{i+1,j}|t_j]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

In the odd steps,  $t_j$  is sent back to region 2, and we move to check the next clause by applying the following rules:

$$[{}_3r'_{i+1,j}|t_j]_3 \rightarrow t_j[{}_3r_{i+1,1}]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

If  $\neg x_j$  is present and  $x_j$  is false, then clause  $c_i$  is satisfied and, in the even steps, the following rules are applied:

$$f_j[{}_3r_{i,j}|b'_{i,j}]_3 \rightarrow b'_{i,j}[{}_3r'_{i+1,j}|f_j]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

We move to the next clause and  $f_j$  is sent back to membrane 2 by using the following rules in the odd steps:

$$[{}_3r'_{i+1,j}|f_j]_3 \rightarrow f_j[{}_3r_{i+1,1}]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

For the cases when the current variable  $j$  does not make the clause true, we use the following rules at the even steps (the move to the next variable happens at the next step):

$$f_j[{}_3r_{i,j}|b_{i,j}]_3 \rightarrow b_{i,j}[{}_3r''_{i,j}|f_j]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n, \quad \text{or}$$

$$t_j[{}_3r_{i,j}|b'_{i,j}]_3 \rightarrow b'_{i,j}[{}_3r''_{i,j}|t_j]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

At the next step, the protein  $r''_{i,j}$  will be changed into  $r_{i,j+1}$  so that the checking can continue with the next variable:

$$[{}_3r''_{i,j}|f_j]_3 \rightarrow f_j[{}_3r_{i,j+1}]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n, \quad \text{or}$$

$$[{}_3r''_{i,j}|t_j]_3 \rightarrow t_j[{}_3r_{i,j+1}]_3, \quad 1 \leq i \leq m, \quad 1 \leq j \leq n.$$

If the protein  $r_{i,n+1}$  is reached on membrane 3, then the clause  $c_i$  is not satisfied and there is no further move ( $\gamma$  is false). If the protein  $r_{m+1,1}$  is reached, then all clauses  $c_1, c_2, \dots, c_m$  are satisfied and we stop ( $\gamma$  is true). The checking phase takes  $2nm$  steps.

#### 4. Answering Phase

At the end of the computation, we have to send out the answer, **yes** or **no**. In three steps, the object **yes** is sent out in the environment, and the total number of steps needed to get the **yes** answer is  $2nm + 5n + 7$ .

First, we apply

$$[_3r_{m+1,1}|f]_3 \rightarrow \mathbf{yes}[_3r_{m+1,1}]_3,$$

then

$$[_2p_n|\mathbf{yes}]_2 \rightarrow \mathbf{yes}[_2p'_n]_2,$$

and finally

$$[_1p|\mathbf{yes}]_1 \rightarrow \mathbf{yes}[_1p']_1.$$

In parallel, in the membrane 5 (which is used as a counter), the following rules are applied:

$$g[_5c_i|g]_5 \rightarrow g[_5c_{i+1}|g]_5, \quad 0 \leq i \leq 2nm + 5n + 5.$$

Simultaneously with sending out the object **yes** from region 1, the following rule is applied:

$$g[_5c_{2nm+5n+7}|g]_5 \rightarrow g'[_5c_{2nm+5n+8}|g']_5.$$

If the object **yes** is not sent out at the step  $2nm + 5n + 7$  (thus, we still have the protein  $p$ , not  $p'$ , on membrane 1), then, in the step  $2nm + 5n + 8$ , we apply the rule:

$$[_1p|g']_1 \rightarrow \mathbf{no}[_1p|]_1.$$

and the computation is completed.

It is now clear that the solution to the satisfiability problem of the instance  $\gamma$  is given by the system in linear time, observation that completes the proof.

### 5.7 Using a More Restrictive Type of Rules

Let us consider a type of rules more restrictive than those of type  $1cp$ , namely of the following forms:

$$\begin{aligned} [_ip|a &\rightarrow [_ip'|a, \\ a[_ip| &\rightarrow a[_ip'|, \end{aligned}$$

where  $a$  is an object and  $p, p'$  are proteins (we change the protein, but we do not move or change the object which assists the change of the protein). We say that these rules are of type  $0cp$ .

Another type of rules which deserve to be considered are usual non-cooperative multiset-rewriting rules, which we write in the form

$$[_ia \rightarrow u]_i,$$

for  $a \in O, u \in O^*$ .

Rules of these types can be considered together with rules of all types from Section 5.1, for all combinations which are not universal or are not known to be so. One such combination is considered in the next theorem:

**Theorem 5.9**  $NOP_2(ncoo, 2res, 0cp) = NRE$ .

*Proof.* We start from the observation that a set of numbers is in  $NRE$  if it is the length set of a recursively enumerable language over the one-letter alphabet and from the fact that each recursively enumerable language can be generated by a matrix grammar with appearance checking. Let us consider such a grammar, in the binary normal form given by Lemma 1.3.7 from [39], hence, the form  $G = (N, \{a\}, S, M, F)$ , where  $N = N_1 \cup N_2 \cup \{S, \#\}$ , with these three sets mutually disjoint, and the matrices in  $M$  of the following forms:

1.  $(S \rightarrow XA), X \in N_1, A \in N_2,$
2.  $(X \rightarrow Y, A \rightarrow x), X, Y \in N_1, A \in N_2, x \in (N_2 \cup \{a\})^*, |x| \leq 2,$
3.  $(X \rightarrow Y, A \rightarrow \#), X, Y \in N_1, A \in N_2,$
4.  $(X \rightarrow \lambda, A \rightarrow x), X \in N_1, A \in N_2, x \in a^*, |x| \leq 2.$

Moreover, there is only one matrix of type 1, and  $F$  consists exactly of all rules of the form  $A \rightarrow \#$  from matrices of type 3. A matrix of type 4 is used only once – in the end of a derivation.

Starting from such a grammar, we replace each rule  $A \rightarrow x$  from matrices of types 2 and 4 having  $|x| \leq 1$  with  $A \rightarrow xd^j$ , where  $d$  is a new symbol and  $|xd^j| = 2$ .

We denote the obtained grammar with  $G'$  and we assume that its matrices of types 1, 2, and 3 are labeled injectively with  $m_1, \dots, m_n$ .

We now construct the P system with proteins on membranes (of degree 2):

$$\Pi = (O, P, \mu, w_1/z_1, w_2/z_2, E, R_1, R_2, i_o), \text{ with}$$

$$O = \{\alpha, \alpha', \alpha'' \mid \alpha \in N_2 \cup \{a, d\}\} \cup \{b, c, c', \#\},$$

$$P = \{X, X_i, X'_i, X''_i \mid X \in N_1, 1 \leq i \leq n\} \cup \{p\},$$

$$\mu = [{}_1[{}_2]_2]_1,$$

$$w_1 = bcA, \quad z_1 = \lambda,$$

$$w_2 = \lambda, \quad z_2 = X, \text{ for } (S \rightarrow XA) \text{ being the initial matrix of } G,$$

$$E = \emptyset,$$

$$i_o = 2,$$

and the sets  $R_1, R_2$  of rules constructed as follows (we present them with comments about their use in the simulation of matrices of  $G'$ ).

For any matrix  $m_i : (X \rightarrow Y, A \rightarrow x)$  of type 2 we introduce in  $R_2$  the following rules:

- $A[{}_2X] \rightarrow [{}_2X]A$
- $[{}_2X]A \rightarrow [{}_2X_i]A$

The object  $A$  enters membrane 2 in the presence of protein  $X$ , thus starting the simulation of matrix  $m_i$ ; only one rule of this form can be used because there is only one occurrence of  $X$  on the membrane. If instead of the second rule we

use the next *ncoo* rule, then two non-terminals will be present inside membrane 2 and the computation will never stop (see below).

- $[_2A \rightarrow \alpha'\beta'']_2$ , for  $x = \alpha\beta$ ,  $\alpha, \beta \in N_2 \cup \{a, d\}$

The second rule of the matrix is simulated inside membrane 2, and no other rule can be used as long as  $X_i$  is present on the membrane.

- $[_2X_i|\alpha' \rightarrow [_2X'_i|\alpha'$

- $[_2X'_i|\alpha' \rightarrow \alpha'[_2X'_i|$

The primed object  $\alpha'$  introduced when simulating the rule  $A \rightarrow \alpha\beta$  helps the protein  $X_i$  to get primed, and in the presence of  $X'_i$  exits membrane 2.

- $[_2\alpha' \rightarrow \#]_2$

- $[_2\# \rightarrow \#]_2$

Only one primed object can exit membrane 2 in one step, because there is only one protein on the membrane; if there are further primed objects in membrane 2, then they introduce the trap-object  $\#$ , which will evolve forever by means of the rule  $[_2\# \rightarrow \#]_2$ .

- $[_2X'_i|\beta'' \rightarrow \beta''[_2X'_i|$

- $\beta''[_2X'_i| \rightarrow \beta''[_2X''_i|$

The double primed object  $\beta''$  exits membrane 2 and, from outside, helps the protein to get double primed. Note that  $\beta''$  cannot exit before  $\alpha'$  otherwise,  $\alpha'$  introduces the trap-object.

- $b[_2X'_i|] \rightarrow [_2X'_i|]b$

- $[_2b \rightarrow \#]_2$

The change of  $X'_i$  into  $X''_i$  should be done immediately; otherwise,  $b$  enters membrane 2 and introduces here the trap-object.

- $b[_2X''_i|] \rightarrow b[_2Y|]$

After the protein on membrane 2 gets double primed, the auxiliary object  $b$  changes  $X''_i$  into  $Y$ , thus completing the simulation of the first rule of the matrix; this happens simultaneously with using the rules of  $R_1$  presented below, and this completes the simulation of the matrix  $m_i$ . The possible copies of  $d$  do not further evolve in membrane 1, while the copies of  $a$  wait in the skin region until the end of the simulation and then are introduced in membrane 2 (see below).

For a matrix like the one above, we introduce in  $R_1$  the following rules:

- $[_1\alpha' \rightarrow \alpha]_1$

- $[_1\beta'' \rightarrow \beta]_1$

For a matrix  $m_i : (X \rightarrow \lambda, A \rightarrow x)$  of type 4, we introduce in  $R_2$  and  $R_1$  the same rules as above, with the exception of the rule  $b[_2X''_i|] \rightarrow b[_2Y|]$ , which is replaced with the rule:

- $b[_2X''_i|] \rightarrow b[_2p|]$

The protein  $p$  is produced on membrane 2 in the end of derivations of  $G'$ ; then, we also add the following rules:

- $A[{}_2p] \rightarrow [{}_2p]A$
- $[{}_2A \rightarrow \#]_2$ , for all  $A \in N_2$

If the derivation in  $G'$  is not terminal, then any remaining non-terminal  $A$  can enter membrane 2, and either evolve here by a rule  $[{}_2A \rightarrow \alpha'\beta'']_2$  associated with a rule of  $G'$ , and this leads to introducing  $\#$  (because the primed symbols cannot leave the membrane), or it directly introduces the trap-object. Using the rule  $[{}_2A \rightarrow \#]_2$  during the simulation of a matrix as above leads to a computation which never stops; hence, no wrong result is obtained.

We also introduce in  $R_2$  the following rule:

- $a[{}_2p] \rightarrow [{}_2p]a$

It is used to introduce in the output membrane each occurrence of  $a$ , after completing the simulation of a terminal derivation in  $G'$ . Note that if the computation halts, then membrane 2 contains only copies of the object  $a$  and nothing else.

For a matrix  $m_i : (X \rightarrow Y, A \rightarrow \#)$  of type 3, we introduce in  $R_2$  the following rules:

- $c[{}_2X] \rightarrow c[{}_2X_i]$
- $c[{}_2X_i] \rightarrow [{}_2X_i]c$

The auxiliary object  $c$  non-deterministically chooses a matrix  $m_i$  of type 3 to simulate, and, after changing the protein into  $X_i$ , enters membrane 2.

- $A[_2X_i| \rightarrow [_2X_i|A$

The move of object  $c$  across membrane 2, in the presence of protein  $X_i$ , is done in competition with the move of object  $A$  from the second rule of matrix  $m_i$ ; if this object exists and if it enters membrane 2, then it evolves here either by a rule of the form  $A \rightarrow \alpha'\beta''$  from a matrix of types 2 or 4, or by the rule  $[_2A \rightarrow \#]_2$  considered above. In the first case, the objects  $\alpha', \beta''$  cannot exit the membrane, because we have rules for them only for proteins  $X_i$  with  $i$  associated with matrices of other types than of type 3 – the matrices are injectively labeled. Thus, in all cases, the computation never halts.

- $[_2c \rightarrow c']_2$
- $[_2X_i|c' \rightarrow [_2X'_i|c'$

If any occurrence of  $A$  exists, then it has to enter membrane 2, either before  $c$  or after, because  $c$  spends one step inside membrane 2, changing to  $c'$ , and only after that it can change the protein from  $X_i$  to  $X'_i$ .

- $[_2X'_i|c' \rightarrow c'[_2X'_i|$
- $c'[_2X'_i| \rightarrow c'[_2Y|$

After changing the protein to  $X'_i$ , object  $c'$  can exit membrane 2 and, from outside, it can help the protein to change into  $Y$ , thus completing the simulation of the matrix.

What still remains to be done is to return the auxiliary object  $c'$  to its non-primed version, and this is done by the following rule which is introduced in  $R_1$ :

- $[_1c' \rightarrow c]_1$

From the previous explanations, it is clear that the computation in  $\Pi$  halts if and only if it corresponds to a correct simulation of a terminal derivation in  $G'$ . Because the object  $d$  does not evolve and does not enter the output membrane, it follows that the number of copies of  $a$  introduced in membrane 2 is exactly the length of the string produced by the derivation in grammar  $G$ , hence  $N(\Pi)$  is the length set of  $G$ .

It remains as an open problem to consider the case of using rules of type  $0cp$  in the same system with rules of types  $1res$  or  $4res$  (or of a more powerful type than these two).

## CHAPTER 6

### REWRITING P SYSTEMS WITH SYMPORT RULES

#### 6.1 The Model

In standard membrane computing (with symbol objects), the rules are used in a non-deterministic and maximally parallel manner, but in our case, the rewriting rules are applied in a sequential way and, because at each step there is only one string present in the system, the symport rules are also applied in a sequential mode, at most one at a time. Another particular aspect is that we do not need to bring in objects from the environment, which we assume to be empty at the beginning of the computation.

We introduce now the new type of P systems that uses context-free rewriting rules for the evolution of objects placed inside compartments of a cell, and symport rules for communication between membranes. The strings circulate across membranes depending on their membership to regular languages given by means of regular expressions.

**Definition 6.1** A *rewriting-symport P system* is a construct of the form

$$\Pi = (V, T, H, \mu, w, (R_h, C_h)_{h \in H}, h_0),$$

where:

1.  $V$  is the total alphabet,
2.  $T \subseteq V$  is the terminal alphabet,
3.  $H$  is a finite set of labels for membranes,
4.  $\mu$  is the membrane structure, with the membranes labeled, in a one-to-one manner, with elements of  $H$ ,
5.  $w \in V^*$  is the starting string, placed in the skin region at the beginning of the computations,
6.  $R_h$ , with  $h \in H$ , are finite sets of context-free rewriting rules of the form  $a \rightarrow x$ , for  $a \in V$  and  $x \in V^*$ , associated with regions  $h$  of  $\mu$ ,
7.  $C_h$ , with  $h \in H$ , are finite sets of symport rules of the form  $(E, in)$  or  $(E, out)$ , where  $E$  is a regular expression over  $V$ , associated with membranes  $h$  of  $\mu$ ,
8.  $h_0 \in H$  is (the label of) the output region of the system.

Note that we work with a single string introduced initially in the system, namely, in the skin region; this restrictive assumption can be relaxed, using sets of strings (maybe empty), placed in each region of the initial configuration. Note also the fact that the rewriting rules are associated with the regions (they correspond to the bio-chemistry taking place in the compartments of a cell), while the communication rules are associated with the membranes (and they correspond to the selective protein channels embedded on membranes).

The idea is that the rewriting rules process the string present in the system, in the usual string rewriting mode, while the symport rules transport the string across membranes. A string  $z$  present in a region  $h$  can exit this region if  $z \in L(E)$ , for a rule  $(E, out) \in C_h$ . A string  $z$  present in the region surrounding membrane  $h$  can enter region  $h$  if  $z \in L(E)$ , for a rule  $(E, in) \in C_h$ . In both cases, we say that the rules  $(E, out)$  and  $(E, in)$  were applied to the string  $z$ .

In each step, only one rule can be applied to the string present in the system, and this rule can be any rewriting or any communicating rule which can act on the string. That is, the rewriting is sequential (only one rule is applied to the string), and there is no priority between rewriting and communicating rules, the kind of rules and the specific rule to apply are chosen non-deterministically.

The computation starts in the initial configuration, which is defined by  $\mu$  and  $w$ , and continues indefinitely. Each string  $x \in T^*$ , which appears in region  $h_0$  of the system is said to be computed by system  $\Pi$ , and introduced into the language  $L(\Pi)$  (generated/computed by  $\Pi$ ). No restriction to halting computations is imposed, while a string  $x \in T^*$  already introduced in  $L(\Pi)$  can continue its evolution, maybe producing further strings which will be included in  $L(\Pi)$ .

**Example 6.1** Let  $\Pi$  be the rewriting-symport P system

$$\Pi = (V, T, \{1, 2\}, [{}_1[{}_2 \ ]_2]_1, A, (R_1, \emptyset), (R_2, C_2), 2),$$

where

$$V = \{A, B, a\},$$

$$\begin{aligned}
T &= \{a\}, \\
R_1 &= \{A \rightarrow BB\}, \\
R_2 &= \{B \rightarrow A, B \rightarrow a\}, \\
C_2 &= \{(B^+, in), (A^+, out)\}.
\end{aligned}$$

The computation starts with the axiom  $A$  (non-terminal symbol) in the skin region; assume that we have a string  $A^n$ , for  $n \geq 1$ . The rule  $A \rightarrow BB$  must be used for all occurrences of  $A$  and only after obtaining the string  $B^{2n}$  we can move to region 2. If we use the rule  $B \rightarrow A$ , then all occurrences of  $B$  must be replaced by  $A$ , and the string  $A^{2n}$  can then go to region 1. Thus, the doubling of the number of occurrences of  $A$  can be repeated. If the rule  $B \rightarrow a$  is used in region 2, then the rule  $B \rightarrow A$  should not be used; otherwise, the string will remain blocked in region 2, without turning to a terminal string. Therefore,  $L(\Pi) = \{a^{2^n} \mid n \geq 1\}$ .

## 6.2 Computational Results

The control ensured by the regular expressions from the symport rules reminds the conditional grammars from the regulated rewriting area, [39], so it is no surprise that rewriting-symport P systems are universal. We start by presenting an easy proof of this result, also obtaining interesting suggestions from the proof about restrictive forms of the communication rules.

**Theorem 6.1** Rewriting-symport P systems with two membranes can compute all recursively enumerable languages.

*Proof.* Let us consider a type-0 grammar  $G = (N, T, S, P)$  in Kuroda normal form, hence with  $P = P_1 \cup P_2$ , where  $P_1$  contains context-free rules of the forms  $A \rightarrow BC$ ,  $A \rightarrow a$ ,  $A \rightarrow \lambda$ , where  $A, B, C \in N$ ,  $a \in T$ , and

$$P_2 = \{r_i : AB \rightarrow CD \mid A, B, C, D \in N, 1 \leq i \leq k\}$$

is a set of non-context-free rules (which we assume to be injectively labeled with  $r_1, \dots, r_k$ ).

We construct the P system

$$\Pi = (V, T, \{1, 2\}, [{}_1[{}_2]_2]_1, S, (R_1, \emptyset), (R_2, C_2), 1),$$

where

$$V = N \cup T \cup \{\alpha'_i, \alpha''_i \mid \alpha \in N, 1 \leq i \leq k\},$$

$$R_1 = P_1 \cup \{A \rightarrow C'_i, B \rightarrow D''_i \mid r_i : AB \rightarrow CD \in P_2, 1 \leq i \leq k\},$$

$$R_2 = \{\alpha'_i \rightarrow \alpha, \alpha''_i \rightarrow \alpha \mid \alpha \in N, 1 \leq i \leq k\},$$

$$C_2 = \{((N \cup T)^* \{\alpha'_i \beta''_i \mid \alpha, \beta \in N, 1 \leq i \leq k\} (N \cup T)^*, in), ((N \cup T)^*, out)\}.$$

The equality  $L(G) = L(\Pi)$  is obvious: a sentential form of the grammar  $G$  (initially,  $S$ ) placed in region 1 can be rewritten by the context-free rules of  $P_1$  any number of steps; if any rule  $A \rightarrow C'_i$  or  $B \rightarrow D''_i$  associated with a rule  $r_i : AB \rightarrow CD \in P_2$ , for  $1 \leq i \leq k$ , is used, then both these rules should be used on neighboring positions; otherwise, the string cannot enter membrane 2 and hence it will never produce a terminal string; in membrane 2, we just return the symbols  $C'_i, D''_i$  to  $C, D$ , and the string is sent back to the skin region. Any terminal derivation in  $G$  corresponds to a successful (and halting) computation in  $\Pi$ .

What is of interest in this proof is the particular form of the regular expressions from the symport rules: they are either of the form  $U^*$  or of the form  $U^*FU^*$ , for a given alphabet  $U$  and a finite language  $F$ . This suggests to us to consider regular expressions of the following normal forms:

$$(nf1) \ E = U_1^*WU_2^* \text{ or } E = U^*,$$

$$(nf2) \ E = U^*W, \ E = WU^*, \text{ or } E = U^*,$$

$$(nf3) \ E = U^*,$$

where  $U, U_1, U_2$  are alphabets and  $W$  is a set of symbols.

Note that the finite set  $W$  consists of symbols, not of pairs of symbols as in the proof of Theorem 6.1. The alphabets  $U, U_1, U_2$  may be empty, hence (nf1) is more general than (nf2), which is more general than (nf3).

We denote by  $ELSP_m(rw, sym_{nfj})$  the family of languages generated by rewriting-symport P systems with at most  $m$  membranes, using symport rules with the regular expressions of the form (nfj),  $j = 1, 2, 3$ . Of course, the alphabets  $U$  and the set  $W$  can be different from a rule to another one. If the number of membranes is not bounded, then we replace the subscript  $m$  with  $*$ . If  $T = V$  (hence no specific terminal alphabet is considered), then we remove the front letter E (we say then that the system is non-extended, a terminology usual in the L systems area [86]).

The following inclusions are direct consequences of the definitions (the appearance of  $[E]$  means that the respective relation is true both with E in the two sides of the relation and without E).

**Lemma 6.1** (i)  $[E]LSP_m(rw, sym_{nfj}) \subseteq [E]LSP_{m+1}(rw, sym_{nfj}) \subseteq [E]LSP_*(rw, sym_{nfj}) \subseteq RE$ , for all  $m \geq 1$ , and  $j = 1, 2, 3$ .

(ii)  $[E]LSP_m(rw, sym_{nf3}) \subseteq [E]LSP_m(rw, sym_{nf2}) \subseteq [E]LSP_m(rw, sym_{nf1})$ , for all  $m \geq 1$  or  $m = *$ .

(iii)  $LSP_m(rw, sym_{nfj}) \subseteq ELSP_m(rw, sym_{nfj})$ , for all  $m \geq 1$  or  $m = *$ , and  $j = 1, 2, 3$ .

Somewhat surprisingly, taking into account the weak control imposed by symport rules of this type, already the systems with regular expressions of the form  $(nf2)$  are universal (even in the non-extended case).

**Theorem 6.2**  $LSP_*(rw, sym_{nf2}) = RE$ .

*Proof.* In view of Lemma 6.1, we only prove the inclusion  $RE \subseteq LSP_*(rw, sym_{nf2})$ , and to this aim we use the so-called *rotate-and-simulate* technique much used in the DNA computing area.

As in the proof of Theorem 6.1, we consider a type-0 grammar  $G = (N, T, S, P)$  in Kuroda normal form, with  $P = P_1 \cup P_2$ , where  $P_1$  is the set of context-free rules, of the forms  $A \rightarrow BC, A \rightarrow a, A \rightarrow \lambda$ , with  $A, B, C \in N, a \in T$ , and

$$P_2 = \{r_i : AB \rightarrow CD \mid A, B, C, D \in N, 1 \leq i \leq k\}$$

is the set of non-context-free rules (injectively labeled with  $r_1, \dots, r_k$ ). Let  $X$  be a new symbol, and denote  $U = N \cup T \cup \{X\}$ .

We construct the rewriting-symport P system

$$\Pi = (V, V, H, \mu, XXS, (R_1, \emptyset), (R_i, C_i)_{1 \leq i \leq k}, (R_{i'}, C_{i'})_{1 \leq i \leq k},$$



$$C_{f_2} = \{(XT^*, in)\}.$$

This system functions as follows. The sentential forms  $z$  of grammar  $G$  are present in the system  $\Pi$  in the form  $z_2XXz_1$ , for  $z = z_1z_2$ , with  $z_1, z_2 \in (N \cup T)^*$  (hence circularly permuted, with the new symbol  $X$  marking the correct beginning of the original string – by means of  $XX$ ). The context-free rules of  $P$  will be simply applied to the string in the skin region of  $\Pi$ . The non-context-free rules  $r_i : AB \rightarrow CD \in P_2, 1 \leq i \leq k$  are simulated with the help of membranes  $i$  and  $i'$ , in the rotate-and-simulate style: the symbols  $A, B$  are removed from the right hand end of the string and  $C, D$  are appended in the left hand of the string – we will discuss this in detail below.

In order to make available in the end of the strings the necessary symbols  $AB$ , the strings can be circularly permuted, with the help of membranes with label  $\alpha \in U$ . This is done as follows. Assume that we have a string  $z\alpha$  in the skin region, with  $z \in U^*, \alpha \in U$ . By means of the rule  $\alpha \rightarrow \bar{\alpha}$ , one occurrence of  $\alpha$  is barred; if this is not the end one, the string will never enter membrane  $f_1$ . If the barred symbol is the rightmost one, and no symbol different from those in  $U$  appears in the string, then we can move the string in the membrane with the label  $\alpha$  (note that this membrane is precisely associated with  $\alpha$ ), where  $\bar{\alpha}$  is removed, and the front symbol,  $\beta$ , of  $z$  (there is such a symbol, because  $z$  consists of at least the two copies of  $X$ ) introduces  $\hat{\alpha}$ . The string can exit membrane  $\alpha$  only if both these rules are used (with the rule  $\beta \rightarrow \hat{\alpha}\beta$  used indeed for  $\beta$  being the first letter of  $z$ ). In the skin region, the front letter  $\hat{\alpha}$  is replaced by  $\alpha$ ; hence, we get the string  $\alpha z$ . This process can be repeated

as many times as needed (and it handles in the same way terminal and non-terminal symbols of  $G$ , as well as the occurrences of  $X$ ).

Assume now that we have in the skin region a string of the form  $zAB$ , for some  $r_i : AB \rightarrow CD \in P_2$ , and we want to simulate this rule. The end occurrence of  $B$  can be replaced with  $D_i$  – and only in this case the string can lead to a terminal string in  $L(\Pi)$ . The string  $zAD_i$  enters region  $i$  (for the membrane  $i$  uniquely associated with the rule  $r_i$ ). There are two rules which can be used here,  $D_i \rightarrow \lambda$  and  $A \rightarrow C'_i$ . No further step is possible if we do not use both these rules, with the latter one applied to the occurrence of  $A$  from the right hand end of the string. Therefore, we get the string  $zC'_i$  which passes to membrane  $i'$ . Again, two rules must be used here,  $C'_i \rightarrow \lambda$  and  $\alpha \rightarrow C''D\alpha$ , for some  $\alpha \in U$  (the second rule rewrites the front symbol of  $z$ ); otherwise, the string cannot leave this membrane. If the string leaves membrane  $i'$ , then it is of the form  $C''Dz$ . In region  $i$  the front symbol  $C''$  is replaced with  $C'''$ , and only in this case the string can leave membrane  $i$  (if we use again the rule  $A \rightarrow C'_i$ , even if  $A$  is the rightmost symbol of  $z$ , the string remains blocked in region  $i$ ). Thus, we return to the skin region with the string  $C'''Dz$ ; we replace  $C'''$  with  $C$  and the simulation of the rule  $r_i$  is complete.

Note that any other way of using the rules leads to a deadlock, either with the string blocked in a membrane  $i$  or  $i'$ , or with it in the skin region, but unable to enter any lower level membrane.

The simulation of rules in  $G$  can continue as long as necessary. At any moment, a string of the forms  $XXz$  or  $XzX$  can enter the membrane  $f_1$ , where it can lose one or both symbols  $X$ . In the latter case, the string remains forever here. If the string  $z$

was a terminal one, and if we have erased only one occurrence of  $X$ , still preserving one in the leftmost position, then the string can enter the output membrane,  $f_2$ , where the last symbol  $X$  is removed.

Thus, the only way to send a string from  $T^*$  to the output membrane is to follow correctly a derivation in  $G$  and to have in the end the string in the same order of symbols as in the grammar  $G$  (this is ensured by the condition to have at least one symbol  $X$  in the leftmost position when entering membrane  $f_2$ ). Note the way the selection by the symport rule of membrane  $f_2$  also ensures that the string is terminal; hence, it is not necessary to take explicitly the alphabet  $T$  as a subset of  $V$ . Consequently,  $L(G) = L(\Pi)$ .

The proof of a similar result, but with a bound on the number of membranes, is left as an open problem.

**Corollary 6.1**  $[E]LSP_*(rw, sym_{nf_2}) = [E]LSP_*(rw, sym_{nf_1}) = RE$ .

For the most restricted form of regular expressions,  $(nf_3)$ , the power of our systems is strictly smaller: in this case, we get a characterization of the family of *ET0L* languages.

**Theorem 6.3**  $ET0L = LSP_m(rw, sym_{nf_3}) = ELSP_m(rw, sym_{nf_3})$ , for all  $m \geq 3$  or  $m = *$ .

*Proof.* (i)  $ET0L \subseteq LSP_m(rw, sym_{nf_3})$ .

Let  $G = (V, T, w, P_1, P_2)$  be an *ET0L* system in the two-table normal form.

Let us consider the alphabets  $V_i = \{a_i \mid a \in V\}$ ,  $i = 1, 2$ , where  $a_1, a_2$  are new symbols

associated with  $a \in V$ . We also consider the morphisms (codings)  $h_i : V^* \longrightarrow V_i^*$ , defined by  $h_i(a) = a_i$ , for all  $a \in V, i = 1, 2$ .

We construct the P system

$$\Pi = (U, U, \{1, 2, 3\}, [{}_1[{}_2]_2[{}_3]_3]_1, w, (R_1, \emptyset), (R_2, C_2), (\emptyset, C_3), 3),$$

where

$$U = V \cup V_1 \cup V_2,$$

$$R_1 = \{a \rightarrow h_1(x) \mid a \rightarrow x \in P_1\}$$

$$\cup \{a \rightarrow h_2(x) \mid a \rightarrow x \in P_2\},$$

$$R_2 = \{a_1 \rightarrow a, a_2 \rightarrow a \mid a \in V\},$$

$$C_2 = \{(V_1^*, in), (V_2^*, in), (V^*, out)\},$$

$$C_3 = \{(T^*, in)\}.$$

The rules of the tables  $P_1, P_2$  of  $G$  can be simulated in the skin region, with the symbols introduced by these rules having subscripts 1 for rules in  $P_1$  and subscripts 2 for rules in  $P_2$ . These rules should not be mixed; otherwise, the string remains blocked in region 1. Moreover, all symbols of the string should be rewritten; otherwise, the string cannot enter membrane 2. In membrane 2, the symbols lose their subscripts and only after that the string can exit. The process can be iterated. At any moment, any terminal string can enter membrane 3 and it remains there.

Consequently,  $L(G) = L(\Pi)$ , which concludes the proof of the inclusion  $ET0L \subseteq LSP_m(rw, sym_{nf3})$ .

(ii)  $ELSP_*(rw, sym_{nf3}) \subseteq ETOL$ .

Consider now the following rewriting-symport P system with an arbitrary number of membranes,

$$\Pi = (V, T, H, \mu, w, (R_h, C_h)_{h \in H}, h_0).$$

For simplicity, we relabel the membranes with numbers; hence, we assume  $H = \{1, 2, \dots, m\}$ , where  $m$  is the number of membranes from  $\mu$ . Without loss of the generality, we may also assume that the skin membrane has now label 1 and the output membrane has label  $m$ .

For each symbol  $\alpha \in V$ , we consider the symbols  $\alpha_i$ , for  $0 \leq i \leq m$ . Note that we have also used the subscript 0; we denote  $H_0 = H \cup \{0\}$  and  $V_{H_0} = \{\alpha_i \mid \alpha \in V, 0 \leq i \leq m\}$ . For each  $i \in H_0$ , we consider the morphism (coding)  $f_i : V^* \longrightarrow V_{H_0}^*$  defined by  $f_i(\alpha) = \alpha_i$ , for  $\alpha \in V$ .

We construct the  $ETOL$  system  $G = (W, T, f_1(w), \mathcal{P})$ , with

$$W = V_{H_0} \cup T \cup \{\#\},$$

where  $\#$  is a new symbol, and with the tables constructed below.

The idea behind this construction is to encode the fact that the string of the P system  $\Pi$  is present in region  $i$  by adding the subscript  $i$  to all symbols of the strings – with 0 representing the environment of the system (the string can exit the system and can come back by symport rules from  $C_1$ ). Then, the  $ETOL$  system will work on such a string with subscripted symbols – the rules from  $R_i$  will rewrite symbols of the form  $\alpha_i$  and produce symbols with the same subscript. In order to prevent the application of rules from a region  $j$  on the strings present in region  $i$ , we introduce

the trap-symbol  $\#$  for each symbol  $\alpha_j$  in all tables associated with rules from regions  $j$  different from  $i$ . These ideas are implemented in the next tables in the following way:

1. For  $i \in H$  and each rule  $r : \alpha \rightarrow x \in R_i$ , we introduce in  $\mathcal{P}$  the table

$$\begin{aligned}
 P_r &= \{\alpha_i \rightarrow f_i(x)\} \\
 &\cup \{\alpha_i \rightarrow \alpha_i \mid \alpha \in V\} \\
 &\cup \{\alpha_j \rightarrow \# \mid \alpha \in V, j \in H_0 - \{i\}\} \\
 &\cup \{a \rightarrow a \mid a \in T\} \cup \{\# \rightarrow \#\}.
 \end{aligned}$$

The rule  $\alpha \rightarrow x$  is simulated in the form  $\alpha_i \rightarrow f_i(x)$ . Note that the table  $P_r$  contains also the rule  $\alpha_i \rightarrow \alpha_i$ ; hence, any number of occurrences of  $\alpha_i$  – maybe zero, and then the string remains unchanged – can be rewritten by this rule. It is not necessary to rewrite all, as otherwise imposed by the parallelism of the *ETOL* system if we would not have the rule  $\alpha_i \rightarrow \alpha_i$ . If the table is applied to a “wrong” string, i.e., to the string of  $\Pi$  present in a region different from  $i$ , then the trap-symbol is introduced. This symbol is never removed; hence, the string will never become terminal. The rules  $a \rightarrow a$ , for  $a \in T$ , are introduced in order to make the table complete.

2. Consider now the communication rules. Let  $(U^*, in) \in C_i$  be such a rule, and let  $j$  be the label of the region surrounding membrane  $i$  in  $\mu$  (with  $j = 0$  for  $i = 1$ ). Then, we introduce in  $\mathcal{P}$  a table which we denote by  $P(U, j \rightarrow i)$ , with the notation telling that the string composed of symbols from  $U$  can move from

region  $j$  to region  $i$ . Now let  $(U^*, out)$  be a rule from some  $C_k$ , and let  $l$  be the region surrounding region  $k$ . We introduce in  $\mathcal{P}$  a table  $P(U, k \rightarrow l)$ , with the same meaning as above. Thus, for all such rules we have to consider tables of the same form:

$$\begin{aligned}
P(U, j \rightarrow i) &= \{\alpha_j \rightarrow \alpha_i \mid \alpha \in U\} \\
&\cup \{\alpha_j \rightarrow \# \mid \alpha \notin U\} \\
&\cup \{\alpha_k \rightarrow \# \mid \alpha \in V, k \in H_0 - \{j\}\} \\
&\cup \{a \rightarrow a \mid a \in T\} \cup \{\# \rightarrow \#\}.
\end{aligned}$$

If such a table is used, and the string in the *ETOL* system corresponds to a string from the region  $j$  of  $\Pi$ , then the subscripts of all symbols are changed from  $j$  to  $i$ , which corresponds to moving the string in region  $i$ ; it is important to note that only symbols from  $U$  can change the subscripts, while symbols not in  $U$  will introduce the trap-symbol. The trap-symbol is introduced also if we apply this table to a string which is not in region  $j$  (hence, the current subscripts of symbols are different from  $j$ ).

3. Finally, we introduce the following table:

$$\begin{aligned}
P_m &= \{a_m \rightarrow a \mid a \in T\} \\
&\cup \{\alpha_m \rightarrow \# \mid \alpha \in V - T\} \\
&\cup \{\alpha_j \rightarrow \# \mid \alpha \in V, j \in H_0 - \{m\}\} \\
&\cup \{a \rightarrow a \mid a \in T\} \cup \{\# \rightarrow \#\}.
\end{aligned}$$

This table can be applied in such a way not to introduce the trap-object only if the string is of the form  $f_m(w)$ , for  $w \in T^*$ , and in this way we get a string which is terminal with respect to  $G$ .

From the previous explanations, we have the equality  $L(G) = L(\Pi)$ , and this proves the inclusion  $ELSP_*(rw, sym_{nf3}) \subseteq ET0L$ .

Together with Lemma 6.1 (iii) for  $j = 3$ , the previous inclusions prove the theorem.

# CHAPTER 7

## CONCLUSIONS, OPEN PROBLEMS, AND FURTHER RESEARCH

This dissertation investigates variants of membrane systems as models for molecular computing. Thus, we follow the standard approach of research in membrane computing: defining a new model of computation for membrane systems, and investigating the computational power of such computing device. Specifically, we address issues concerning the power of bio-inspired communication mechanisms with proteins placed on membranes, and the power of rewriting P systems with communication by symport rules.

In the case of P systems with *mate/drip* rules, we prove that  $PsMAT \subseteq PsOP_3(mate_2, drip_3)$ , but it remains as an open problem the question if the converse inclusion is also true.

We have introduced and investigated a class of P systems where the multisets of objects from the compartments of the membrane structure evolve under the control of multisets of proteins placed on the membranes. Several types of rules were considered and in many cases universality was obtained, even for systems with the minimal number of membranes, one. In some cases, also the number of proteins present at any moment on the membrane is rather small (this can be considered as a descriptive

complexity measure of the systems). In [76], we left as an open problem the possibility to bound the number of proteins also in Theorem 5.3 and Theorem 5.6.

Some of these questions had been already answered in [56]. The unbound number of proteins for Theorem 5.3 is reduced to 7 and for Theorem 5.6 to 10 proteins; also other combinations of rules are considered. We give a complete list of the universality results from [56], mentioning only the types of rules used and the bounded number of proteins placed on a membrane (see Table 7.1).

Table 7.1 Universality results from [56].

Rules	Number of Proteins
3ffp	7
2ffp, 4ffp	7
2ffp, 1ffp	7
2ffp, 1res	10
2ffp, 3res	9
2ffp, 5res	8
1ffp, 2res	9
1ffp, 3res	8
4ffp, 3res	9

Another question is whether rules of pure forms are strictly weaker than rules of the general form of types  $cp$  and  $ff$ .

Here we had investigated the computational power of several classes of P sys-

tems of the types mentioned above; for many cases, we got characterizations of recursively enumerable sets of numbers, hence Turing completeness, while for some restricted cases only singleton sets can be computed. Some combinations of types of rules remain to be further investigated. No other classes of P systems using only one type of rules can be universal besides the ones already investigated (*2cpp*, *3cpp*, and *3ffp*). In all cases, we work in the maximally parallel semantics; other possibilities (sequential use of rules [37], [41], minimally parallel use of rules [4], etc.) remain as research topics. Another possible direction is to redefine the result of a computation, e.g., taking the length of the computation as result, in the sense of [51] or [68].

It remains as an open problem to consider the case of using rules of type *0cp*, considered in Section 5.7, in the same system with rules of types *1res* or *4res* (or of a more powerful type than these two).

In the case of rewriting-symport P systems, the proof of a similar result to the one from Theorem 6.2, but with a bound on the number of membranes, is left as an open problem.

## BIBLIOGRAPHY

- [1] L.M. Adleman: Molecular Computation of Solutions to Combinatorial Problems. In *Science*, vol. 226, 1994, 1021–1024.
- [2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002.
- [3] A. Alhazov: Solving SAT by Symport/Antiport P Systems with Membrane Division. In *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects), Sevilla, Spain, 2005*, Fénix Editora, Sevilla, 2005, 1–6.
- [4] A. Alhazov: Minimal Parallelism and Number of Membrane Polarizations. In *Pre-proceedings of Membrane Computing, International Workshop, Leiden, The Netherlands, 2006*, WMC7, Leiden, 2006, 74–87.
- [5] A. Alhazov, T.-O. Ishdorj: Membrane Operations in P Systems with Active Membranes. In *Proceedings of the Second Brainstorming Week on Membrane Computing, Sevilla, Spain, 2004*, BWMC 2004, Report RGNC 01/04, University of Sevilla, 2004, 37–52.
- [6] A. Alhazov, M. Margenstern, V. Rogozhin, Y. Rogozhin, S. Verlan: Communicative P Systems with Minimal Cooperation. In *Membrane Computing. 5th International Workshop, Milan, Italy, 2004. Revised Selected and Invited Papers*, WMC5, LNCS 3365, Springer-Verlag, Berlin, 2005, 162–178.
- [7] A. Alhazov, C. Martín-Vide, L. Pan: Solving Graph Problems by P Systems with Restricted Elementary Active Membranes. In *Aspects of Molecular Computing*, LNCS 2950, Springer, Berlin, 2004, 1–22.
- [8] A. Alhazov, D. Sburlan: Static Sorting Algorithms for P Systems. In *Pre-Proceedings of the Workshop on Membrane Computing, Tarragona, Spain, 2003*, WMC 2003, GRLMC Report 28/03, Tarragona, 2003, 17–40.
- [9] A. Alhazov, D. Sburlan: Static Sorting P Systems. In [34], 215–252.
- [10] I.I. Ardelean, D. Besozzi, M.H. Garzon, G. Mauri, S. Roy: P System Models for Mechanosensitive Channels. In [34], 43–81.
- [11] I.I. Ardelean, M. Cavaliere: Modeling Biological Processes by Using a Probabilistic P System Software. In *Natural Computing*, vol. 2(2), 2003, 173–197.

- [12] J.J. Arulanandham: Implementing Bead-Sort with P Systems. In *Unconventional Models of Computation 2002*, LNCS 2509, Springer, Berlin, 2002, 115–125.
- [13] J. Bartosik: Paun's Systems in Modeling of Human Resource Management. In *Proceedings of Second Conference of Tools and Methods of Data Transformation*, WSU Kielce, 2004.
- [14] G. Bel Enguix, M.D. Jiménez-Lopez: Linguistic Membrane Systems and Applications. In [34], 347–388.
- [15] F. Bernardini, M. Gheorghe: Population P Systems. In *Journal of Universal Computer Science*, vol. 10(5), 2004, 509–539.
- [16] F. Bernardini, M. Gheorghe, N. Krasnogor, R.C. Muniyandi, M.J. Pérez-Jiménez, F.J. Romero-Campero: On P Systems as a Modelling Tool for Biological Systems. In *Pre-Proceedings of Sixth International Workshop on Membrane Computing, Vienna, Austria, 2005*, WMC6, Vienna, 2005, 193–213.
- [17] F. Bernardini, V. Manca: Dynamical Aspects of P Systems. In *BioSystems*, vol. 70, 2002, 85–93.
- [18] F. Bernardini, A. Păun: Universality of Minimal Symport/Antiport: Five Membranes Suffice. In *Proceedings of Membrane Computing. International Workshop, Tarragona, Spain, 2003, Revised Papers*, WMC 2003, LNCS 2933, Springer, Berlin, 2004, 43–54.
- [19] D. Besozzi, D. Busi, G. Franco, R. Freund, Gh. Păun: Two Universality Results for (Mem)Brane Systems. In *Proceedings of the Fourth Brainstorming Week on Membrane Computing, Sevilla, Spain, 2006*, BWMC 2006, vol. I, Fénix Editora, Sevilla, 2006, 49–62.
- [20] L. Bianco: Introduction to Psim. Manuscript, 2004. Available at: <http://psystems.disco.unimib.it/software/Bianco/psim.pdf>
- [21] L. Bianco, F. Fontana, G. Franco, V. Manca: P Systems for Biological Dynamics. In [34], 83–128.
- [22] N. Busi: On the Computational Power of the Mate/Bud/Drip Brane Calculus: Interleaving vs. Maximal Parallelism. In *Pre-Proceedings of Sixth Workshop on Membrane Computing, Vienna, Austria, 2005*, WMC6, Vienna, 2005, 235–252.
- [23] L. Cardelli: Brane Calculi – Interactions of Biological Membranes. In *Computational Methods in Systems Biology. International Conference, Paris, France, 2004. Revised Selected Papers*, CMSB 2004, LNCS 3082, Springer-Verlag, Berlin, 2005, 257–280.
- [24] L. Cardelli, Gh. Păun: An Universality Result for a (Mem)Brane Calculus Based on Mate/Drip Operations. In *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects), Sevilla, Spain, 2005*, Fénix Editora,

- Sevilla, 2005, 75–94, and in *International Journal of Foundations of Computer Science*, vol. 17(1), 2006, 49–68.
- [25] J. Castellanos, Gh. Păun, A. Rodríguez-Patón: P Systems with Worm-Objects. In *IEEE 7th International Conference on String Processing and Information Retrieval, La Coruña, Spain, 2000*, SPIRE 2000, La Coruña, 64–74, and CDMTCS TR 123, University of Auckland, 2000 ([www.cs.auckland.ac.nz/CDMTCS](http://www.cs.auckland.ac.nz/CDMTCS)).
- [26] M. Cavaliere: Evolution–Communication P Systems. In *Membrane Computing, International Workshop, Curtea de Argeş, Romania, 2002. Revised Papers*, WMC-CdeA 2002, LNCS 2597, Springer, Berlin, 2003, 134–145.
- [27] M. Cavaliere, I.I. Ardelean: Modeling Respiration in Bacteria and Respiration/Photosynthesis Interaction in Cyanobacteria Using a P System Simulator. In [34], 129–158.
- [28] M. Cavaliere, A. Riscos-Núñez, R. Brijder, G. Rozenberg: Membrane Systems with Marked Membranes. Manuscript, 2005.
- [29] R. Ceterchi, R. Gramatovici, N. Jonoska, K.G. Subramanian: Generating Picture Languages with P Systems. In *Proceedings of the Brainstorming Week on Membrane Computing*, BWMC 2003, Technical Report 26/03, Rovira i Virgili University, Tarragona, 2003, 85–100.
- [30] R. Ceterchi, C. Martín-Vide: P Systems with Communication for Static Sorting. In *Pre-Proceedings of Brainstorming Week on Membrane Computing, Tarragona, Spain, 2003*, BWMC 2003, Technical Report 26/03, Rovira i Virgili University, Tarragona, 2003, 101–117.
- [31] H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On Trace Languages Generated by Spiking Neural P Systems. In *Proceedings of the Fourth Brainstorming Week on Membrane Computing, Sevilla, Spain, 2006*, BWMC 2006, vol. I, 2006, 207–224.
- [32] G. Ciobanu: Modeling Cell-Mediated Immunity by Means of P Systems. In [34], 159–180.
- [33] G. Ciobanu, D. Dumitriu, D. Huzum, G. Moruz, B. Tanasă: Client-Server P Systems in Modeling Molecular Interaction. In *Membrane Computing, International Workshop, Curtea de Argeş, Romania, 2002. Revised Papers*, WMC-CdeA 2002, LNCS 2597, Springer, Berlin, 2003, 203–218.
- [34] G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing*, Springer-Verlag (Natural Computing Series), Berlin, 2006.
- [35] E. Csuhaj-Varju, M. Margenstern, G. Vaszil, S. Verlan: Small Computationally Complete Symport/Antiport P Systems. In *Proceedings of the Fourth Brainstorming Week on Membrane Computing, Sevilla, Spain, 2006*, BWMC 2006, vol. I, 2006, 267–281.

- [36] E. Czeizler: Self-Activating P Systems. In *Membrane Computing. International Workshop, Curtea de Argeş, Romania, 2002. Revised Papers*, WMC-CdeA 2002, LNCS 2597, Springer, Berlin, 2003, 234–246.
- [37] Z. Dang, O.H. Ibarra: On P Systems Operating in Sequential and Limited Parallel Modes. In *Pre-Proceedings of the Workshop on Descriptive Complexity of Formal Systems, London, Canada, 2004*, DCFS 2004, London, 2004, 164–177.
- [38] V. Danos, S. Pradalier: Projective Brane Calculus. In *Computational Methods in Systems Biology: International Conference, Paris, France, 2004. Revised Selected Papers*, CMSB 2004, LNCS 3082, Springer-Verlag, Berlin, 2005, 134–148.
- [39] J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
- [40] G. Franco, V. Manca: A Membrane System for the Leukocyte Selective Recruitment. In *Membrane Computing. International Workshop, Tarragona, Spain, 2003, Revised Papers*, WMC 2003, LNCS 2933, Springer, Berlin, 2004, 181–190.
- [41] R. Freund: Asynchronous P Systems and P Systems Working in the Sequential Mode. In *Membrane Computing. International Workshop, Milano, Italy, 2004*, WMC5, LNCS 3365, Springer-Verlag, Berlin, 2005, 36–62.
- [42] R. Freund, L. Kari, M. Oswald, P. Sosik: Computationally Universal P Systems without Priorities: Two Catalysts Are Sufficient. *Theoretical Computer Science*, vol. 330(2), 2005, 251–266.
- [43] R. Freund, M. Oswald: Tissue P Systems with Symport/Antiport rules of One Symbol Are Computationally Universal. In *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects)*, Sevilla, Spain, 2005, 187–200.
- [44] R. Freund, A. Păun: P Systems with Active Membranes and without Polarizations. In *Proceedings of the Second Brainstorming Week on Membrane Computing, Sevilla, Spain, 2004*, BWMC 2004, Report RGNC 01/04, University of Sevilla, 2004, 193–205, and in *Soft Computing*, vol. 9(9), 2005, 657–663.
- [45] M.R. Garey, D.S. Johnson: *Computer and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.
- [46] A. Georgiou: *SubLP-Studio* Software, 2003. Available at: <http://psystems.disco.unimib.it/software.html>
- [47] A. Georgiou, M. Gheorghe, F. Bernardini: Membrane-Based Devices Used in Computer Graphics. In [34], 253–282.
- [48] J.-L. Giavitto, G. Malcolm, O. Michel: Rewriting Systems and the Modeling of Biological Systems. In *Comparative and Functional Genomics*, vol. 5, 2004, 95–99.

- [49] R. Gramatovici, G. Bel Enguix: Parsing with P Automata. In [34], 389–410.
- [50] T. Head: Formal Language Theory and DNA: An Analysis of the Generative Capacity of Specific Recombinant Behaviors. In *Bulletin of Mathematical Biology*, vol. 49, 1987, 737-759.
- [51] O.H. Ibarra, A. Păun: Counting Time in Computing with Cells. In *Proceedings of DNA Based Computing, London, Canada, 2005*, DNA11, London, 2005, 25–36.
- [52] M. Ito, C. Martín-Vide, Gh. Păun: Characterization of Parikh Sets of ETOL Languages in Terms of P Systems. In *Words, Semigroups, and Transducers, World Scientific*, Singapore, 2001, 239–254.
- [53] R. Karwowski: *L-studio* v. 3.1. Department of Computer Science, University of Calgary, 2001. Available at:  
<http://www.cpsc.ucalgary.ca/Research/bmv/lstudio>.
- [54] W. Korczynski: Paun’s Systems and Accounting. In *Pre-Proceedings of Sixth International Workshop on Membrane Computing, Vienna, Austria, 2005*, WMC6, Vienna, 2005, 461–464.
- [55] I. Korec: Small Universal Register Machines. In *Theoretical Computer Science*, vol. 168, 1996, 267–301.
- [56] S.N. Krishna: Combining Brane Calculus and Membrane Computing. In *Proceedings of Bio-Inspired Computing – Theory and Applications Conference, Wuhan, China, September 2006, Membrane Computing Section, BIC-TA 2006*.
- [57] S.N. Krishna, A. Păun: Results on Catalytic and Evolution-Communication P Systems. In *New Generation Computing*, vol. 22(4), 2004, 377–394.
- [58] S.N. Krishna, Gh. Păun: P Systems with Mobile Membranes. In *Natural Computing*, vol. 4(3), 2005, 255-274.
- [59] S.N. Krishna, R. Rama: A Variant of P Systems with Active Membranes: Solving NP-Complete Problems. In *Romanian Journal of Information Science and Technology*, vol. 2(4), 1999, 357–367.
- [60] S.N. Krishna, R. Rama, H. Ramesh: Further Results on Contextual and Rewriting P Systems. In *Fundamenta Informaticae*, vol. 64(1-4), 2005, 241–253.
- [61] A. Leporati, C. Zandron: A Family of P Systems which Solve 3-SAT. In *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects), Sevilla, Spain, 2005*, Fénix Editora, Sevilla, 2005, 247–256.
- [62] A. Lindenmayer: Mathematical Models for Cellular Interaction in Development. In *Journal of Theoretical Biology*, vol. 18, 1968, 280–315.
- [63] L. M. Loew, J. C. Schaff: The Virtual Cell – A software Environment for Computational Cell Biology. In *TRENDS in Biotechnology*, vol. 19(10), 2001, 401–406.

- [64] M. Madhu, K. Krithivasan: Improved Results About the Universality of P Systems. In *Bulletin of the EATCS*, vol. 76, 2002, 162–168.
- [65] O. Michel, F. Jacquemard: An Analysis of a Public Key Protocol with Membranes. In [34], 283–302.
- [66] O. Michel, F. Jacquemard, J.-L. Giavitto: Three Variations on the Analysis of the Needham-Schroeder Public Key Protocol with MGS. Technical Report LaMI-98-2004, University d'Évry - CNRS, 2004.
- [67] M.L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, 1967.
- [68] H. Nagda, A. Păun, A. Rodriguez-Paton: P Systems with Symport/Antiport and Time. In *Pre-proceedings of Membrane Computing, International Workshop, Leiden, The Netherlands, 2006*, WMC7, Leiden, 2006, 429–442.
- [69] T.Y. Nishida: Simulations of Photosynthesis by a  $K$ -Subset Transforming System with Membranes. In *Fundamenta Informaticae*, vol. 49(1–3), 2002, 249–259.
- [70] T.Y. Nishida: A Membrane Computing Model of Photosynthesis. In [34], 181–202.
- [71] T.Y. Nishida: Membrane Algorithms: Approximate Algorithms for NP-Complete Optimization Problems. In [34], 303–314.
- [72] L. Pan, C. Martín-Vide: Solving Multiset 0 – 1 Knapsack Problem by P Systems with Input and Active Membranes. In *Proceedings of the Second Brainstorming Week on Membrane Computing, Sevilla, Spain, 2004*, BWMC 2004, Report RGNC 01/04, University of Sevilla, 2004, 342–353.
- [73] A. Păun: On P Systems with Membrane Division. In *Unconventional Models of Computation*, Springer, London, 2000, 187–201.
- [74] A. Păun: Membrane Systems with Symport/Antiport. Universality Results. In *Pre-Proceedings of Second Workshop on Membrane Computing*, Curtea de Argeş, Romania, 2002, 333–344.
- [75] A. Păun, Gh. Păun: The Power of Communication: P Systems with Symport/Antiport. In *New Generation Computing*, vol. 20(3), 2002, 295–306.
- [76] A. Păun, B. Popa: P Systems with Proteins on Membranes. In *Fundamenta Informaticae*, vol. 72(4), 2006, 467–483.
- [77] A. Păun, B. Popa: P Systems with Proteins on Membranes and Membrane Division. In *Proceedings of the 10th International Conference on Developments in Language Theory, Santa Barbara, CA, USA, 2006*, DLT 2006, LNCS 4036, Springer, Berlin, 2006, 292–303.

- [78] Gh. Păun: Computing with Membranes. In *Journal of Computer and System Sciences*, vol. 61(1), 2000, 108–143 (and Turku Center for Computer Science – TUCS Report 208, November 1998, [www.tucs.fi](http://www.tucs.fi)).
- [79] Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
- [80] Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodríguez-Patón: Symport/Antiport P Systems with Three Objects Are Universal. In *Fundamenta Informaticae*, vol. 64(1-4), 2005, 353–367.
- [81] M.J. Pérez-Jiménez, A. Riscos-Núñez: A Linear Time Solution to the Knapsack Problem Using Active Membranes. In *Membrane Computing. International Workshop, Tarragona, Spain, 2003, Revised Papers*, WMC 2003, LNCS 2933, Springer, Berlin, 2004, 250–268.
- [82] M.J. Pérez-Jiménez, F.J. Romero-Campero: A Study of the Robustness of the EGFR Signalling Cascade Using Continuous Membrane Systems. In *Mechanisms, Symbols, and Models Underlying Cognition. First International Work-Conference on the Interplay between Natural and Artificial Computation, Las Palmas, Spain, 2005*, IWINAC 2005, LNCS 3561, Springer, Berlin, 2005, 268–278.
- [83] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: Complexity Classes in Models of Cellular Computing with Membranes. In *Natural Computing*, vol. 2(3), 2003, 265–285.
- [84] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: Computationally Hard Problems Addressed Through P Systems. In [34], 315–346.
- [85] Y. Rogozhin, S. Verlan: On the Rule Complexity of Universal Tissue P Systems. In *Membrane Computing, International Workshop, Vienna, Austria, 2005, Selected and Invited Papers*, WMC6, LNCS 3850, Springer-Verlag, Berlin, 2006, 356–363.
- [86] G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems*. Academic Press, New York, 1980.
- [87] G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1987.
- [88] Y. Suzuki, H. Tanaka: Chemical Evolution among Artificial Proto-Cells. In *Artificial Life*, vol. 7, 2000, 54–63.
- [89] Y. Suzuki, H. Tanaka: Modeling p53 Signaling Pathways by Using Multiset Processing. In [34], 203–214.
- [90] M. Tomita, K. Hashimoto, K. Takahashi, Y. Matsuzaki, R. Matsushima, K. Saito, K. Yugi, F. Miyoshi, H. Nakano, S. Tanida, Y. Saito, A. Kawase, N. Watanabe, T. Shimizu, and Y. Nakayama: The E-CELL Project – Towards Integrative Simulation of Cellular Processes. In *New Generation Computing*, vol. 18(1), 2000, 1–12.

- [91] C. Zandron, G. Mauri, C. Ferreti: Universality and Normal Forms on Membrane Systems. In *Proceedings of International Workshop on Grammar Systems, Bad Ischl, Austria, 2000*, Bad Ischl, 2000, 61–74.