



DEPARTMENT OF APPLIED INFORMATICS  
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS  
COMENIUS UNIVERSITY, BRATISLAVA

---

# BIOLOGICALLY INSPIRED COMPUTATION MODELS

(Written part of dissertation exam)

MICHAL KOVÁČ

---

**Advisor:** doc. RNDr. Damas Gruska, PhD.

Bratislava, 2013



I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgment has been made in the text.

.....

# Acknowledgments

I am very grateful to all the people I had the privilege to work with, especially I would like to thank my advisor RNDr. Damas Gruska PhD. for his constant interest, encouragement and help that I deeply appreciate. The present work was possible due to many beneficial consultations and intensive cooperation.

I would also like to thank to the members of Laboratory of Comparative and Functional Genomics of Eukaryotic Organelles and their head, Prof. Lubomír Tomáška for familiarization with the work of biologists.

I thank to my family, colleagues, teammates and all the friends who supported me especially last weeks before the deadlines for their patience, support and understanding.

# Abstract

Author: Michal Kováč

Title: Biologically inspired computation models

University: Comenius University in Bratislava

Faculty: Faculty of Mathematics, Physics and Informatics

Department: Department of Applied Informatics

Supervisor: doc. RNDr. Damas Gruska, PhD.

Bratislava, 2013

This work discusses the research in the membrane systems, an emerging field of natural computing. Many variants of membranes systems have already been studied, most of them uses parallel rewriting and are computationally complete. Various sequential models have been proposed, however, in many cases they are weaker than their parallel variant.

We propose some other sequential models, provide universality proofs for them and suggest further research.

**Kľúčové slová:** Výpočtové modely inšpirované biológiou, Membránové systémy, P systémy

# Abstrakt

Autor: Michal Kováč

Názov minimovej práce: Biologically inspired computation models

Škola: Univerzita Komenského v Bratislave

Fakulta: Fakulta matematiky, fyziky a informatiky

Katedra: Katedra aplikovanej informatiky

Vedúci minimovej práce: doc. RNDr. Damas Gruska, PhD.

Bratislava, 2013

V tejto práci sa zaoberáme výskumom v oblasti membránových systémov. Veľa variantov už bolo preskúmaných, väčšinou pri výpočte používajú paralelizmus a sú Turingovsky úplné. Navrhlo sa aj veľa sekvenčných modelov, ale väčšina z nich má slabšiu výpočtovú silu ako ich paralelný variant.

V druhej časti predkladáme niektoré sekvenčné modely, u ktorých dokazujeme Turingovu úplnosť a navrhujeme modely, ktoré plánujeme preskúmať v rámci dizertačnej práce.

**Kľúčové slová:** Výpočtové modely inšpirované biológiou, Membránové systémy, P systémy

# Contents

<b>Introduction</b>	<b>1</b>
<b>I Overview of the topic</b>	<b>2</b>
<b>1 Preliminaries</b>	<b>3</b>
1.1 Formal languages . . . . .	3
1.2 Formal grammars . . . . .	4
1.3 Chomsky hierarchy . . . . .	4
1.4 Matrix grammars . . . . .	5
1.5 Register machines . . . . .	5
1.6 Lindenmayer systems . . . . .	6
1.7 Multisets . . . . .	7
1.8 Multiset languages . . . . .	8
1.9 Bisimulations . . . . .	8
<b>2 Natural computing</b>	<b>10</b>
2.1 Bioinformatics . . . . .	10
2.2 Biomolecular computing . . . . .	11
2.3 Biologically inspired computing models . . . . .	11
2.3.1 Neural networks . . . . .	11
2.3.2 Evolutionary algorithms . . . . .	12
2.3.3 Membrane systems . . . . .	12
<b>3 P systems</b>	<b>15</b>
3.1 Definitions . . . . .	15
3.2 P system variants . . . . .	17
3.2.1 Accepting vs generating . . . . .	18

3.2.2	Active vs passive membranes . . . . .	18
3.2.3	Context in rules . . . . .	19
3.2.4	Rules with priorities . . . . .	21
3.2.5	Energy in P systems . . . . .	21
3.2.6	Symport / antiport rules . . . . .	21
3.2.7	Parallelism options . . . . .	22
3.3	Case studies . . . . .	23
3.3.1	Vultures in Pyrenees . . . . .	23
3.3.2	Cellular Signalling Pathways . . . . .	23
3.3.3	Solving SAT in linear time . . . . .	23
3.3.4	Implementation of P systems in vitro . . . . .	24
<b>II</b>	<b>Project proposal</b>	<b>25</b>
<b>4</b>	<b>Project proposal</b>	<b>26</b>
4.1	Sequential P systems without priorities with cooperative rules	26
4.2	P systems with vacuum . . . . .	27
<b>5</b>	<b>Current status of the research</b>	<b>28</b>
5.1	Sequential P systems without priorities with cooperative rules and inhibitors . . . . .	28
5.2	Universality results of P systems with vacuum . . . . .	33
	<b>Conclusions</b>	<b>35</b>



# List of Figures

2.1	The membrane structure of a P system and its associated tree [?]	13
3.1	P system computing a Fibonacci sequence [?]	17

# Introduction

There are a lot of areas in the theoretical computer science that are motivated by other science fields. Computation models motivated by biology forms a large group of them. They include neural networks, computational models based on DNA evolutionary algorithms, which have already found their use in computer science and proved that it is worth to be inspired by biology. L-systems are specialized for describing the growth of plants, but they have also found the applications in computer graphics, especially in fractal geometry. Other emerging areas are still awaiting for their more significant uses.

One of them is the membrane computing. It is relatively young field of natural computing - in comparison: neural networks have been researched since 1943 and membrane systems since 1998 [?].

Membrane systems (P systems) are distributed parallel computing devices inspired by the structure and functionality of cells. Recently, many P system variants have been developed in order to simulate the cells more realistically or just to improve the computational power.

In chapter 1 we recall some computer science basics we will use through the work. Natural computing including models inspired by nature will be introduced in chapter 2. One of these models, P systems, with the current state of the research in their variants, is presented in chapter 3. Part II consists of two chapters, chapter 4 about dissertation project, overview of the current research and future plans and chapter 5 with details about our results.

# Part I

## Overview of the topic

# Chapter 1

## Preliminaries

This chapter is about some basic notions of computer science which will be used through the work. We start by defining formal languages and basic models (grammars, machines) that define language families and end by defining multiset languages.

### 1.1 Formal languages

Our study is based on the classical theory of formal languages. We will recall some definitions:

**Definition 1.1.1** *An **alphabet** is a finite nonempty set of symbols.*

**Definition 1.1.2** *A **string** over an alphabet is a finite sequence of symbols from alphabet.*

The length of the string  $s$  is denoted by  $|s|$ . We denote by  $V^*$  the set of all strings over an alphabet  $V$ . By  $V^+ = V^* - \{\varepsilon\}$  we denote the set of all nonempty strings over  $V$ .

**Definition 1.1.3** *A **language** over the alphabet  $V$  is any subset of  $V^*$ .*

**Definition 1.1.4** *A **family of languages** is a set of languages.*

## 1.2 Formal grammars

**Definition 1.2.1** A formal grammar is a tuple  $G = (N, T, P, \sigma)$ , where

- $N, T$  are disjoint alphabets of non-terminal and terminal symbols,
- $\sigma \in N$  is the initial non-terminal,
- $P$  is a finite set of rewriting rules of the form  $u \rightarrow v$ , with  $u \in (N \cup T)^*N(N \cup T)^*$  and  $v \in (N \cup T)^*$ .

**Definition 1.2.2** A rewriting step in the grammar  $G$  is a binary relation  $\Rightarrow$  on  $(N \cup T)^*$ , where  $x \Rightarrow y$  only if  $\exists w_1, w_2 \in (N \cup T)^+$  and a rule  $u \rightarrow v \in P$  such that  $x = w_1uw_2$  and  $y = w_1vw_2$ .

**Definition 1.2.3** Language defined by a grammar  $G$  is a set  $L(G) = \{w \in T^* \mid \sigma \Rightarrow w\}$ .

Languages that can be generated by a formal grammar are the recursively enumerable languages  $RE$ .

## 1.3 Chomsky hierarchy

In this section we introduce several well-known families of languages.

**Definition 1.3.1 Regular grammar** is formal grammar, where rewriting rules are of the form  $u \rightarrow v$ , where  $u \in N$  and  $v \in T^*(N \cup \{\varepsilon\})$ .

**Definition 1.3.2 Regular languages** are languages generated by regular grammars. They are denoted  $R$ .

**Definition 1.3.3 Context-free grammar** is formal grammar, where rewriting rules are of the form  $u \rightarrow v$ , where  $u \in N$  and  $v \in (N \cup T)^*$ .

**Definition 1.3.4 Context-free languages** are languages generated by context-free grammars. They are denoted  $CF$ .

**Definition 1.3.5 Context-sensitive grammar** is formal grammar, where rewriting rules are of the form  $u \rightarrow v$ , where  $u \in (N \cup T)^*N(N \cup T)^*$ ,  $v \in (N \cup T)^*$  and  $|u| < |v|$ .

**Definition 1.3.6** *Context-sensitive languages* are languages generated by context-sensitive grammars. They are denoted  $CS$ .

These families of languages forms the Chomsky hierarchy by means of inclusions:  $R \subset CF \subset CS \subset RE$ .

## 1.4 Matrix grammars

**Definition 1.4.1** A **matrix grammar** is a tuple  $G = (N, T, M, \sigma)$ , where:

- $N, T$  are disjoint alphabets of non-terminal and terminal symbols,
- $\sigma \in N$  is the initial non-terminal,
- $M$  is a finite set of matrices, which are sequences of context-free rules of the form  $u \rightarrow v$ , where  $u \in N$  and  $v \in (N \cup T)^*$ .

**Definition 1.4.2** A **rewriting step**  $x \Rightarrow y$  holds only if there is a matrix  $(u_1 \rightarrow v_1, u_2 \rightarrow v_2, \dots, u_n \rightarrow v_n) \in M$  such that for each  $1 \leq i \leq n$  the following holds:  $x_i = x'_i u_i x''_i$  and  $x_{i+1} = x'_i v_i x''_i$ , where  $x_i, x'_i, x''_i \in (N \cup T)^*$  and  $x_1 = x$  and  $x_{n+1} = y$ .

**Example 1.4.1** Consider the matrix grammar  $G = (\{\sigma, X, Y\}, \{a, b, c\}, M, \sigma)$ , where  $M$  contains three matrices:  $[S \rightarrow XY], [X \rightarrow aXb, Y \rightarrow cY], [X \rightarrow ab, Y \rightarrow c]$ . There are only context-free rules, yet the grammar generate the context-sensitive language  $\{a^n b^n c^n | n \geq 1\}$ .

The family of matrix grammars is denoted  $MAT$ .

It is known that  $CF \subset MAT \subset RE$ . Interestingly,  $MAT \cap a^* \subset R$  (see [?]).

## 1.5 Register machines

**Definition 1.5.1** A  **$n$ -register machine** is a tuple  $M = (n, P, i, h)$ , where:

- $n$  is the number of registers,
- $P$  is a set of labeled instructions of the form  $j : (op(r), k, l)$ , where  $op(r)$  is an operation on register  $r$  of  $M$ , and  $j, k, l$  are labels from the set  $Lab(M)$  (which numbers the instructions in a one-to-one manner),

- $i$  is the initial label, and
- $h$  is the final label.

The machine is capable of the following instructions:

- $(add(r), k, l)$  : Add one to the contents of register  $r$  and proceed to instruction  $k$  or to instruction  $l$ ; in the deterministic variants usually considered in the literature we demand  $k = l$ .
- $(sub(r), k, l)$  : If register  $r$  is not empty, then subtract one from its contents and go to instruction  $k$ , otherwise proceed to instruction  $l$ .
- $halt$  : This instruction stops the machine. This additional instruction can only be assigned to the final label  $h$ .

A deterministic  $m$ -register machine can analyze an input  $(n_1, \dots, n_m) \in N_0^m$  in registers 1 to  $m$ , which is recognized if the register machine finally stops by the halt instruction with all its registers being empty (this last requirement is not necessary). If the machine does not halt, the analysis was not successful.

## 1.6 Lindenmayer systems

In 1968, a Hungarian botanist and theoretical biologist Aristid Lindenmayer introduced [?] a new string rewriting algorithm named Lindenmayer systems (or L-systems for short). They are used by biologists and theoretical computer scientists to mathematically model growth processes of living organisms, especially plants. The difference with Chomsky grammars is that rewriting is parallel, not sequential.

The simplest version of L-systems assumes that the development of a cell is free of influence of other cells. This type of L-systems is called  $0L$  systems, where “0” stands for zero-sided communication between cells.

**Definition 1.6.1**  $0L$  system is a triple  $(\Sigma, P, \omega)$ , where  $\Sigma$  is an alphabet,  $\omega$  is a word over  $\Sigma$  and  $P$  is a finite set of rewriting rules of the form  $a \rightarrow x$ , where  $a \in \Sigma, x \in \Sigma^*$ .

It is assumed there is at least one rewriting rule for each letter of  $\Sigma$ .  $0L$  system works in parallel way, so all the symbols are rewritten in each step.

**Example 1.6.1** Consider the 0L system with alphabet  $\Sigma = \{a, b\}$ , initial word  $\omega = a$  and rewriting rules  $P = \{a \rightarrow b, b \rightarrow ab\}$ . Since in this system there is exactly one rule for every letter of the alphabet, the rewriting is deterministic and the generated words will be  $\{a, b, ab, bab, abbab, \dots\}$ .

1L systems allows the rewriting rules to include context of size 1, so it allows for rules of type  $yaz \rightarrow x$ .

L-systems with tables ( $T$ ) have several sets of rewriting rules instead of just one set. At one step of the rewriting process, rules belonging to the same set have to be applied. The biological motivation for introducing tables is that one may want different rules to take care of different environmental conditions (heat, light, etc.) or of different stages of development.

**Definition 1.6.2** An extended (E0L) system is a pair  $G_1 = (G, \Sigma_T)$ , where  $G = (\Sigma, P, \omega)$  is an 0L system, where  $\Sigma_T \subseteq \Sigma$ , referred to as the terminal alphabet. The language generated by  $G_1$  is defined by  $L(G_1) = L(G) \cap \Sigma_T^*$ .

Such languages are called E0L languages. E0L languages with tables are called ET0L languages.

It is known that  $CF \subset E0L \subset ET0L \subset CS$  (see section 1.3 for definitions of  $CF$  and  $CS$ ).

## 1.7 Multisets

**Definition 1.7.1** A multiset over a set  $X$  is a mapping  $M : X \rightarrow \mathbb{N}$ .

We denote by  $M(x), x \in X$  the multiplicity of  $x$  in the multiset  $M$ .

**Definition 1.7.2** The **support** of a multiset  $M$  is the set  $\text{supp}(M) = \{x \in X \mid M(x) \geq 1\}$ .

It is the set of items with at least one occurrence.

**Definition 1.7.3** A multiset is **empty** when its support is empty.

A multiset  $M$  with finite support  $X = \{x_1, x_2, \dots, x_n\}$  can be represented by the string  $x_1^{M(x_1)} x_2^{M(x_2)} \dots x_n^{M(x_n)}$ . As elements of a multiset can also be strings, we separate them with the pipe symbol, e.g. *element|element|other\_element*.



**Definition 1.7.4** *Multiset inclusion.* We say that multiset  $M_1$  is included in multiset  $M_2$  if  $\forall x \in X : M_1(x) \leq M_2(x)$ . We denote it by  $M_1 \subseteq M_2$ .

**Definition 1.7.5** *The union of two multisets*  $M_1 \cup M_2$  *is a multiset where*  $\forall x \in X : (M_1 \cup M_2)(x) = M_1(x) + M_2(x)$ .

**Definition 1.7.6** *The difference of two multisets*  $M_1 - M_2$  *is a multiset where*  $\forall x \in X : (M_1 - M_2)(x) = M_1(x) - M_2(x)$ .

**Definition 1.7.7** *Product of multiset*  $M$  *with natural number*  $n \in \mathbb{N}$  *is a multiset where*  $\forall x \in X : (n \cdot M)(x) = n \cdot M(x)$ .

## 1.8 Multiset languages

The number of occurrences of a given symbol  $a \in V$  in the string  $w \in V^*$  is denoted by  $|w|_a$ .

**Definition 1.8.1**  $\Psi_V(w) = (|w|_{a_1}, |w|_{a_2}, \dots, |w|_{a_n})$  is called a Parikh vector associated with the string  $w \in V^*$ , where  $V = \{a_1, a_2, \dots, a_n\}$ .

**Definition 1.8.2** For a language  $L \subseteq V^*$ ,  $\Psi_V(L) = \{\Psi_V(w) | w \in L\}$  is the Parikh mapping associated with  $V$ .

**Example 1.8.1** Consider an alphabet  $V = \{a, b\}$  and a language  $L = \{a, ab, ba\}$ .  $\Psi_V(L) = \{(1, 0), (1, 1)\}$ . Notice that Parikh image of  $L$  has only 2 element while  $L$  has 3 elements.

**Definition 1.8.3** If  $FL$  is a family of languages, by  $PsFL$  is denoted the family of Parikh images of languages in  $FL$ .

## 1.9 Bisimulations

**Definition 1.9.1** A state transition system is a pair  $(S, \rightarrow)$ , where  $S$  is a set of states and  $\rightarrow \subseteq S \times S$  is a binary transition relation over  $S$ .

If  $p, q \in S$ , then  $(p, q) \in \rightarrow$  is usually written as  $p \rightarrow q$ . This represents the fact that there is a transition from state  $p$  to state  $q$ .

**Definition 1.9.2** A labelled state transition system (LTS) is a tuple  $(S, A, \rightarrow)$ , where  $S$  is a set of states,  $A$  is a set of labels and  $\rightarrow \subseteq S \times A \times S$  is a ternary transition relation.

If  $p, q \in S$  and  $a \in A$ , then  $(p, a, q) \in \rightarrow$  is usually written as  $p \xrightarrow{a} q$ . This represents the fact that there is a transition from state  $p$  to state  $q$  with a label  $a$ .

**Definition 1.9.3** Let  $(S_1, A, \rightarrow)$  and  $(S_2, A, \rightarrow)$  be two labelled transition systems. A **simulation** is a binary relation  $R \subseteq S_1 \times S_2$  such that if  $(s_1, s_2) \in R$  then for each  $s_1 \xrightarrow{a} t_1$  there is some  $s_2 \xrightarrow{a} t_2$  such that  $(t_1, t_2) \in R$ .

**Definition 1.9.4** Let  $(S_1, A, \rightarrow)$  and  $(S_2, A, \rightarrow)$  be two labelled transition systems. A **bisimulation** is a binary relation  $R \subseteq S_1 \times S_2$  such that if  $(s_1, s_2) \in R$  then:

1. for each  $s_1 \xrightarrow{a} t_1$  there is some  $s_2 \xrightarrow{a} t_2$  such that  $(t_1, t_2) \in R$ ,
2. for each  $s_2 \xrightarrow{a} t_2$  there is some  $s_1 \xrightarrow{a} t_1$  such that  $(t_1, t_2) \in R$ .

# Chapter 2

## Natural computing

Recently, an interdisciplinary research between the fields of Computer Science and Biology has been rapidly growing. Natural computing consists of three classes of methods:

- those that are based on the use of computers to synthesize natural phenomena
- those that employ natural materials (e.g., molecules) to compute
- those that take inspiration from nature for the development of novel problem-solving techniques

### 2.1 Bioinformatics

Bioinformatics has undergone a fast evolving process, especially the areas of genomics and proteomics. Bioinformatics can be seen as the application of computing tools and techniques for the management of biological data. Just to mention a few:

- the design of efficient algorithms for DNA sequence alignment,
- the investigation of methods for prediction of the 3D structure of molecules and proteins,
- the development of data structures to effectively store huge amount of structured data.

## 2.2 Biomolecular computing

Biomolecular computing make use of molecules such as DNA and proteins to perform computations involving storing, retrieving and processing data. It takes advantage of the many different molecules to try many different possibilities at one, so it is somewhat similar to parallel computing.

Adleman in his 1994 report [?] demonstrated a proof of concept use of DNA as a form of computation which solved the Hamiltonian path problem. Since then, various Turing machines have been proven to be constructible with DNA [?].

DNA can also be used as a digital data storage with 5.5 petabits per cubic millimeter of DNA (see [?]). The information retrieval is, however, a slow process, as the DNA needs to be sequenced in order to retrieve data, so this method is intended mainly for a long-term archival of large amounts of scientific data.

## 2.3 Biologically inspired computing models

On the other hand, the birth of biologically inspired frameworks started the investigation of mathematical models and their properties and technological requirements for their implementation by biological hardware. Those frameworks are inspired by the nature in the way it “computes”, and has gone through the evolution for billions of years.

Neural networks, genetic algorithms and DNA computing are already well established research fields.

### 2.3.1 Neural networks

Inspired by the human brain, which contains on average 86 billions neurons [?], neurophysiologist Warren McCulloch in 1943 proposed a mathematical model of artificial neural network.

A single perceptron computes a function  $f(x)$ , where  $x$  is an input - a vector of real values.

The perceptron can learn itself by modifying parameters used to compute the function. This learning can be performed in various ways, we will mention only the supervised learning. Imagine a function with

- input: perceptron’s parameters

- output: error of the computed result  $f(x)$

If the perceptron receives a feedback in form of error (from the supervision, e.g. dataset used to train the perceptron), it can modify its parameters such the error will be minimized in the future. This can be done through the gradient descent method, which is used to find a local minimum of a function.

A single perceptron can only compute linear functions, so they are often connected with other perceptron to form a neural network. Often it is practically unusable to say what is the purpose of a single neuron in a more complex neural networks.

Neural networks have broad applicability to real world problems and are best if the modeled system has some tolerance to error such as:

- Image recognition: OCR, web search by image
- Music recognition by voice sample
- Time series forecast: weather, stock
- Diagnosing of illnesses

### 2.3.2 Evolutionary algorithms

Evolutionary algorithms are inspired by Darwin's theory of evolution. Basically, an algorithm starts with a population of random individuals (solutions to the problem). In each generation, the fitness of every individual is evaluated and the more fit individuals are replicated and mutated to form a new generation. The less fit individuals die.

Idea of evolutionary computing was introduced in the 1960's by I. Rechenberg. Nowadays they have been applied to find solution of many optimization problems. They also can be used to design or to train a neural network [?].

### 2.3.3 Membrane systems

Nature computes not only at the neural or genetic level, but also at the cellular level. In general, any non-trivial biological system has a hierarchical structure where objects and information flows between regions, what can be interpreted as a computation process.

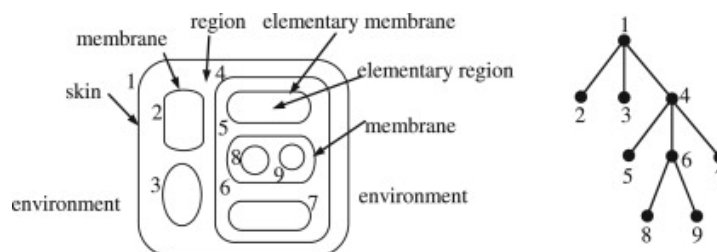


Figure 2.1: The membrane structure of a P system and its associated tree [?]

The regions are typically delimited by various types of membranes at different levels from cell membranes, through skin membrane to virtual membranes which delimits different parts of an ecosystem. This hierarchical system can be seen in other field such as distributed computing, where again well delimited computing units coexist and are hierarchically arranged in complex systems from single processors to the internet.

Membranes keep together certain chemicals or information and selectively determines which of them may pass through.

From these observations, Păun [?] introduces the notion of a membrane structure as a mathematical representation of hierarchical architectures composed of membranes. It is usually represented as a Venn diagram with all the considered sets being subsets of a unique set and not allowed to be intersected. Every two sets are either one the subset of the other, or disjoint. Outermost membrane (also called skin membrane) delimits the finite “inside” and the infinite “outside”.

Hierarchical structures are usually represented by trees, but membrane structures are visually better viewed as Venn diagram as seen in the figure 2.1.

Recently, several computational models based on the membrane structure have been created.

## P systems

P systems were introduced in 1998 as a system with membrane structure that contains multisets of objects and rewrite rules that are executed in a maximally parallel manner. Since then, a huge amount of variants have been created with various computation powers. We have investigated the power of

several such variants. Additionally, we propose a new variant with a vacuum, where the rules can describe what will happen to an object that arrives to an empty membrane.

P systems with existing and newly proposed variants will be discussed in the chapter 3.

Aside from P systems, other models based on the membrane structure have been created such as the Calculus of Looping Sequences (CLS), which was inspired by P systems.

### The Calculus of Looping Sequences

Barbuti in [?] concluded that there is a need for a formalism having a simple notation, having the ability of describing biological systems at different levels of abstractions, having some notions of compositionality and being flexible enough to allow describing new kinds of phenomena as they are discovered, without being specialized to the description of a particular class of systems. The Calculus of Looping Sequences (CLS) was introduced in [?].

The membrane structure in CLS is defined recursively, consisting of terms  $T$  and sequences  $S$ :

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

, where  $a$  is a symbol from the alphabet,  $\varepsilon$  represents the empty sequence,  $\cdot$  is a sequencing operator,  $(S)^L$  is a looping operator,  $\mid$  is a parallel composition operator and  $\rfloor$  is a containment operator.

Membranes are represented by a sequence that is looped around. Several extensions of the CLS have been proposed. In CLS+ the looping operator can be applied to a parallel composition of sequences, which represents a notion of a fluid membrane. CLS+ can be translated to CLS (see [?]). Milazzo in his PhD thesis [?] includes also a simulation of a P system using CLS. The major difficulty is simulating the maximal parallelism of rule application.

# Chapter 3

## P systems

In previous chapter we introduced the notions of membrane and membrane structure.

The next step is to place certain objects in the regions delimited by the membranes. The objects are identified by their names, mathematical symbols from a given alphabet.

Several copies of the same object can appear in a region, so we will work with multisets of objects.

In order to obtain a computing device, we will allow the objects to evolve according to evolution rules. Any object, alone or together with another objects, can be transformed in other objects, can pass through a membrane, and can dissolve the membrane in which it is placed.

All objects evolve at the same time, in parallel manner across all membranes.

The evolution rules are hierarchized by a priority relation, which is a partial order.

These aspects all together forms a P system as introduced in [?]. An example of a P system containing rules for computing the Fibonacci sequence can be seen in the figure 3.1.

In section 3.1 we will provide formal definition of a P system.

### 3.1 Definitions

**P system** is a tuple  $(V, \mu, w_1, w_2, \dots, w_m, R_1, R_2, \dots, R_m)$ , where:

- $V$  is the alphabet of symbols,



- $\mu$  is a membrane structure consisting of  $m$  membranes labeled with numbers  $1, 2, \dots, m$ ,
- $w_1, w_2, \dots, w_m$  are multisets of symbols present in the regions  $1, 2, \dots, m$  of the membrane structure,
- $R_1, R_2, \dots, R_m$  are finite sets of rewriting rules associated with the regions  $1, 2, \dots, m$  of the membrane structure.

Each rewriting rule may specify for each symbol on the right side, whether it stays in the current region, moves through the membrane to the parent region or through membrane to one of the child regions. An example of such rule is the following:  $abb \rightarrow (a, here)(b, in)(c, out)(c, here)$ .

A **configuration** of a P system is represented by its membrane structure and the multisets of objects in the regions.

A **computation step** of P system is a relation  $\Rightarrow$  on the set of configurations such that  $C_1 \Rightarrow C_2$  iff:

For every region in  $C_1$  (suppose it contains a multiset of objects  $w$ ) the multiset in corresponding region in  $C_2$  is the result of applying a multiset of simultaneously applicable multiset rewriting rules to  $w$ .

In the default P system, which works in maximal parallel mode, a maximal multiset of these rules is applied in each step and region.

For example, let's have two regions with multisets  $a|a$  and  $b$ . In the first region there is a rule  $a \rightarrow b$  and in the second membrane there is a rule  $b \rightarrow a|a$ . The only possible result of a computation step is  $b|b, a|a$ . The first rule was applied twice and the second rule once. No more object could be consumed by rewriting rules.

**Computation** of a P system consists of a sequence of steps. The step  $S_i$  is applied to result of previous step  $S_{i-1}$ . So when  $S_i = (C_j, C_{j+1})$ ,  $S_{i-1} = (C_{j-1}, C_j)$ .

There are two possible ways of assigning a result of a computation:

1. By considering the multiplicity of objects present in a designated membrane in a halting configuration. In this case we obtain a vector of natural numbers. We can also represent this vector as a multiset of objects or as Parikh image of a language.
2. By concatenating the symbols which leave the system, in the order they are sent out of the skin membrane (if several symbols are expelled at

the same time, then any ordering of them is accepted). In this case we generate a language.

The result of a computation is clearly only one multiset or a string, but for one initial configuration there can be multiple possible computations. It follows from the fact that there exist more than one maximal multiset of rules that can be applied in each step.

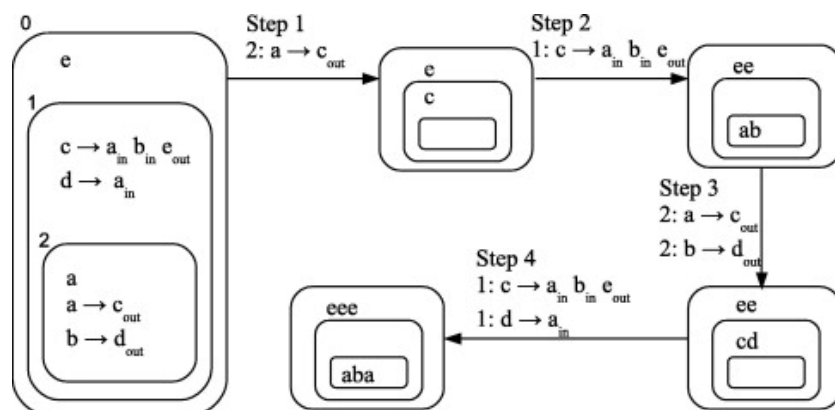


Figure 3.1: P system computing a Fibonacci sequence [?]

## 3.2 P system variants

Besozzi in his PhD thesis (see [?]) formulates three criteria that a good P system variant should satisfy:

1. It should be as much realistic as possible from the biological point of view, in order not to widen the distance between the inspiring cellular reality and the idealized theory.
2. It should result in computational completeness and efficiency, which would mean to obtain universal (and hence, programmable) computing devices, with a powerful and useful intrinsic parallelism;
3. It should present mathematical minimality and elegance, to the aim of proposing an alternative framework for the analysis of computational models.

In membrane computing, many models are equal in power with Turing machines. We should say they are Turing complete (or computationally complete), but because the proofs are always constructive, starting the constructions from these proofs from universal Turing machines or from equivalent devices, we obtain universal P systems (able to simulate any other P system of the given type). That is why we speak about universality results, and not about computational completeness.

### 3.2.1 Accepting vs generating

In the Chomsky hierarchy, there are language acceptors (finite automata, Turing machines) and language generators (formal grammars).

Bordhin in [?] extends grammars to allow for accepting languages by interchanging the left side with the right side of a rule. The mode will apply rewriting rules to an input word and accept it when it reaches the starting nonterminal. However, the input word consists of terminal symbols, which could not be rewritten when using original definition, hence they consider the pure version of various grammar types where they give up the distinction between terminal and nonterminal symbols.

The regular, context-free, context-sensitive and recursively enumerable languages were shown to have equal power in accepting and generating mode. Some other grammars (programmed grammars with appearance checking) are shown to be more powerful in accepting mode than in generating mode. For deterministic Lindenmayer systems, the generating and accepting mode are incomparable.

It can be interesting to investigate accepting and generating mode also in P system variants. Barbuti in [?] shown that in the nondeterministic case, when either promoters or cooperative rules are allowed, acceptor P systems have shown to be universal. The same is known to hold for the corresponding classes of nondeterministic generator P systems. In the deterministic case, acceptor P systems have been shown to be universal only if cooperative rules are allowed. Universality has been shown not to hold for the corresponding classes of generator P systems.

### 3.2.2 Active vs passive membranes

Most of the studied P system variants assumes that the number of membranes can only decrease during a computation, by dissolving membranes as

a result of applying evolution rules to the objects present in the system. A natural possibility is to let the number of membranes also to increase during a computation, for instance, by division, as it is well-known in biology. Actually, the membranes from biochemistry are not at all passive, like those in the models briefly described above. For example, the passing of a chemical compound through a membrane is often done by a direct interaction with the membrane itself (with the so-called protein channels or protein gates present in the membrane); during this interaction, the chemical compound which passes through membrane can be modified, while the membrane itself can in this way be modified (at least locally).

In [?] Păun considers P systems with active membranes where the central role in the computation is played by the membranes: evolution rules are associated both with objects and membranes, while the communication through membranes is performed with the direct participation of the membranes; moreover, the membranes can not only be dissolved, but they also can multiply by division. An elementary membrane can be divided by means of an interaction with an object from that membrane.

Each membrane is supposed to have an electrical polarization (we will say charge), one of the three possible: positive, negative, or neutral. If in a membrane we have two immediately lower membranes of opposite polarizations, one positive and one negative, then that membrane can also divide in such a way that the two membranes of opposite charge are separated; all membranes of neutral charge and all objects are duplicated and a copy of each of them is introduced in each of the two new membranes. The skin is never divided. If at the same time a membrane is divided and there are objects in this membrane which are being rewritten in the same step, then in the new copies of the membrane the result of the evolution is included.

In this way, the number of membranes can grow, even exponentially. As expected, by making use of this increased parallelism we can compute faster. For example, the SAT problem, which is NP complete, can be solved in linear time, when we consider the steps of computation as the time units. Moreover, the model is shown to be computationally universal.

### 3.2.3 Context in rules

Rewriting rules in P systems can be cooperative and non-cooperative, like in Chomsky's context-free and context-sensitive grammars. Non-cooperative rules are restricted to use only one object on the left side and cooperative

rules do not have this restriction. P systems with cooperative rules are universal [?], while P systems with non-cooperative rules only characterize Parikh image of context-free languages (*PsCF*) [?].

Păun [?] also defines P systems with catalysts where catalysts are a specified subset of the alphabet. Rewriting rules can contain catalysts, which are not modified by applying the rule. Surprisingly, P systems with catalytic rules are universal, actually two membranes in the P system are sufficient to achieve universality.

In systems where only catalytic rules (purely catalytic systems [?]), three catalysts are enough [?].

Freund in [?] also shows that two catalysts and one membrane are enough and raised an open problem whether one catalyst is sufficient. He conjectured that for computationally universal P systems the results obtained in this paper are optimal not only with respect to the number of membranes (2), but also with respect to the number of catalysts.

From some point of view, catalysts are way too powerful in restricting the parallelism - they directly participate in the rules, hence the number of catalytic rules that can be applied in one step, is bounded by number of catalysts.

A variant with promoters and inhibitors have been proposed (see [?]).

In the case of promoters, the rules are possible only in the presence of certain symbols. An object  $p$  is a promoter for a rule  $u \rightarrow v$  and we denote this by  $u \rightarrow v|_p$ , if the rule is active only in the presence of object  $p$ . Note that unlike in the case with catalysts, promoters allow the associated rules to be applied as many times as possible.

An object  $i$  is inhibitor for a rule  $u \rightarrow v$  and we denote this by  $u \rightarrow v|_{\neg i}$ , if the rule is active only if inhibitor  $i$  is not present in the region. One of our results (see section 5.1) uses inhibitors as a tool to achieve universality for sequential P systems.

Ionescu in [?] shows that P systems with non-cooperative catalytic rules with only one catalyst and with promoters / inhibitors are universal.

Non-cooperative rules with no catalysts and with inhibitors were studied in [?], the equivalence with Lindenmayer systems (*ETOL* as defined in section 1.6) was proved.

Dang [?] proposes a simple cooperative system (*SCO*) as a P system where the only rules allowed are of the form  $a \rightarrow v$  or of the form  $aa \rightarrow v$ , where  $a$  is a symbol and  $v$  is a (possibly null) string of symbols not containing  $a$ . This variant is investigated with various modes of parallelism, so their

results will be mentioned in the subsection 3.2.7

### 3.2.4 Rules with priorities

In the original definition of a P system [?], a partial order relation over set of rewriting rules have been specified. The rule can be used only if no rule of a higher priority in the region can be applied at the same time.

Sosík in [?] showed that the priorities may be omitted from the model without loss of computational power.

### 3.2.5 Energy in P systems

Various notions of energy has been proposed for use in P systems. Păun in [?] considers a P system where each evolution rule “produces” or “consumes” some quantity of energy, in amounts which are expressed as integer numbers. In each moment and in each membrane the total energy involved in an evolution step should be positive, but if “Too much” energy is present in a membrane, then the membrane will be destroyed (dissolved). This variant was investigated in two cases, both were shown to be universal:

1. when using only two membranes and unbounded amount of energy,
2. when using arbitrarily many membranes and a bounded energy associated with rules

Freund in [?] introduced a new variant where the rules are assigned directly to membranes (every rule consume objects on one side of the membrane and produce objects on the other side) and every membrane carries an energy value that can be changed during a computation by objects passing through the membrane.

This variant is universal even in sequential mode if we allow priorities on the objects. When omitting the priority relation, only the family of Parikh sets generated by context-free matrix grammars (*PsMAT* as defined in section 1.4) is obtained.

### 3.2.6 Symport / antiport rules

Păun in [?] proposes a new way of communicating between membranes.

Symports allow two chemicals to pass together through a membrane in the same direction using symport rules of type  $(ab, in)$  or  $(ab, out)$ . Antiports allow two chemicals to pass simultaneously through a membrane in opposite directions using antiport rules of type  $(a, in; b, out)$ .

Surprisingly, a P system variant, where only the symport / antiport rules are used are computationally complete. Five membranes are enough for this result. If more than two chemicals may collaborate when passing through membranes, two membranes are sufficient for universality. These results are proven in [?].

### 3.2.7 Parallelism options

Original definition of P system (see [?]) uses maximal parallelism when doing a step of computation. There is an obvious biological motivation relying on the assumption that “if we wait long enough, then all reaction which may take place will take place”. This condition is rather powerful, because it decreases the non-determinism of the system’s evolution. For various reasons ranging from looking for more realistic models to just the mathematical challenge, the maximal parallelism was questioned.

Dang in [?] investigates the sequential mode. In each step, from the set of applicable rules across all membrane one is nondeterministically chosen and applied. For purely catalytic systems with 1 membrane, the sequential mode generates only the semilinear sets and thus is strictly weaker than the maximally parallel version. Sequential version of symport / antiport systems are equivalent to vector addition systems making it strictly weaker than the original maximally parallel version.

Investigation of the sequential mode continues in [?]. Sequential P system without priorities with cooperative rules with rules for membrane dissolution are not universal by showing they can be simulated by vector addition systems with states (VASS). This holds even when the membrane creation is allowed for bounded number of created membranes. However, if any number of membranes are allowed to be created, the system becomes universal. This result was shown by simulation of the register machine (see section 1.5).

We have further investigated this variant (sequential P system without priorities with cooperative rules) in section 5.1 by allowing rules with inhibitors, which resulted in universality.

Dang in [?] proposes several restricted versions of parallelism. ***n*-Max-Parallel** version nondeterministically selects a maximal subset of at most

$n$  rules to apply. It is proved that **9-Max-Parallel** SCO (defined in the subsection 3.2.3) is universal.  $\leq n$ -**Parallel** version is similar, but does not require the condition of a maximal subset of rules. It is shown to be weaker than  **$n$ -Max-Parallel** version.  **$n$ -Parallel** version requires the size of the subset of rules to apply to be exactly  $n$ . All three versions are equal to the sequential mode when  $n = 1$ . For non-universality results, Dang used the proof technique by simulation by vector addition systems. Our future research may be inspired by this technique.

Ciobanu in [?] proposes a minimal parallelism: for each region if at least one rule can be applied, then at least one rule will be applied. The symport / antiport rules variant and variant with active membranes were both shown to be universal.

Freund in [?] studied the asynchronous mode of P systems, where in each step, arbitrary many rules can be applied. The application of rules is hence done in parallel way, but are not synchronized or somewhat controlled. In many cases the sequential and asynchronous modes were shown to be equivalent.

## 3.3 Case studies

### 3.3.1 Vultures in Pyrenees

Spanish researchers in [?] presented a model of an ecosystem related to the Bearded Vulture in Pyrenees in Spain by using P systems. They have constructed a simulator to validate the designed P system, which allows them to analyze the evolution under different initial conditions.

### 3.3.2 Cellular Signalling Pathways

Perez in [?] proposed a model for EGFR Signalling Cascade. More than 60 proteins were included and 160 chemical reactions. Membrane structure consists of 3 regions: the environment, the cell surface and the cytoplasm.

### 3.3.3 Solving SAT in linear time

Polynomial time solutions to NP-complete problems by means of P systems are achieved by trading time (number of computation steps) for space (num-



ber of membranes and objects). This is inspired by the capability of cells to produce an exponential number of new membranes in polynomial time.

However, many simulators of P system are inefficient since they cannot handle the parallelism of these devices. Nowadays, we are witnessing the consolidation of the GPUs as a parallel framework to compute general purpose applications such as bitcoin mining. The simulation of P systems with active membranes using GPUs is analysed in [?] and an efficient linear solution to the SAT problem is illustrated.

They compared it to the classical simulator and reported up to 94x of speedup for 256 literals in the formula. The major constraint of this parallel simulation is the GPU memory size, which can be overcome with a data partition on a cluster of GPUs.

### 3.3.4 Implementation of P systems in vitro

“In vitro” are studies in experimental biology that uses components of an organism that have been isolated from their usual biological surroundings in order to provide a more convenient analysis.

There was a planned experiment of computing the Fibonacci sequence using P systems in vitro (see [?]) using test tubes as membranes and DNA molecules as objects, evolving under the control of enzymes. Number of objects in a multiset was represented by a pre-defined “mole” of the substance and synchronization was obtained by “waiting enough”, such that all reactions that can take place in a test tube actually take place. Hence, the variant where reactions don’t cycle is required (Local loop-free P systems). Communication was done by moving all the relevant objects to the next tube in a mechanical way. The final result was read by spectrometry.

The proposed framework has many difficulties. The notable ones are:

- Is there any chance to solve NP-complete problems in this framework?
- Are LL-free P systems universal?

# Part II

## Project proposal

# Chapter 4

## Project proposal

In the Part I we have presented the current state of the research of the P system variants. Especially the parallelism options have been investigated very well, but there are still some gaps that should be filled. For many variants the universality has been proven only when working in the maximally parallel mode. In most cases the sequential mode is strictly weaker, but this is not always true.

We have studied several variants of sequential P systems in order to obtain universality without using maximal parallelism.

### 4.1 Sequential P systems without priorities with cooperative rules

A sequential variant without priorities and with cooperative rules is not universal (see [?]). They have tried modifying the variant to make it universal and showed that with rules for membrane creation with unbounded number of membranes it became universal.

We tried other approach using rules with inhibitors. This variant is universal in both generating and accepting case and we don't even have to use membrane dissolution rules. In generative case the model is able to simulate maximal parallel P system and the accepting model can simulate a register machine.

Future plans include research other more restricted variants such as omitting cooperation in the rules or restrict the power of inhibitors.

## 4.2 P systems with vacuum

We propose a new variant of P system with vacuum in accordance with the criteria formulated in [?] (see beginning of the section 3.2). In the common sense, vacuum represents a state of space with no or a little matter in it. Using vacuum in modelling frameworks can help express certain phenomena more easily. We define a new P system variant, which creates a special vacuum object in a region as soon as the region becomes empty. The vacuum is removed whenever some object interacts with it. After the interaction, there is vacuum no longer. This removal process is realized by allowing the vacuum object to be used only on the left side of rules. If we made the vacuum to be removed automatically when an object enters the region, there would be no difference with the variant without vacuum objects because of no interactions with it.

We are interested in how the variant with the vacuum improves the computation power of a sequential P system in comparison to the variant without using the vacuum. This case have been shown to be universal.

In the future we will research other restriction for this variant such as non-cooperative rules, decaying objects, deterministic steps and will use techniques to show even non-universality results.

# Chapter 5

## Current status of the research

### 5.1 Sequential P systems without priorities with cooperative rules and inhibitors

Original definition of P systems with inhibitors (see [?]) allow to use only one inhibitor per rule, e.g.  $u \rightarrow v|_{\neg i}$ . Alternative definition (see [?]) allow to use whole inhibitor set in the rule like  $u \rightarrow v|_{\neg B}$ , where  $B$  is a set of objects. Such a rule can be applied only if no element of  $B$  is present in the region.

For sequential P systems, the following lemma will show the equivalence of these definitions. However, special condition must be fulfilled, the regions with such rules cannot be empty.

**Lemma 5.1.1** *If there is at least one object present in each region of a P system, rewriting step in P system with inhibitor set can be simulated by multiple consecutive steps of P system with single inhibitor.*

**Proof 5.1.1** Consider P system with alphabet  $V$ . For each rule  $u \rightarrow v|_{\neg B}$ , where  $B = \{b_1, b_2, \dots, b_n\}$  we will have rules:

$$\begin{aligned} & c \rightarrow c|_{GONE_b|_{\neg b}} \text{ for all } c \in V, b \in B \\ & u|_{GONE_{b_1}|_{GONE_{b_2}}| \dots |_{GONE_{b_n}}} \rightarrow v|_{GONE_{b_1}|_{GONE_{b_2}}| \dots |_{GONE_{b_n}}} \end{aligned}$$

Note that symbols  $GONE_b$  are created automatically when some object  $c$  is present in the region.

**Theorem 5.1.2** *The sequential P system with inhibitors defines the same Parikh image of language as P system with maximal parallelism.*

**Proof 5.1.2** We show that we can simulate maximal parallel step of P system with several steps of sequential P system with inhibitors. The proof is quite technical with some workarounds.

It is important to note that in the maximal parallel step the rewriting occurs in all membranes, so we need to synchronize this process. Every membrane will have a state, represented as an object.

The *RUN* state represents that the rewriting still occurs. When there is no more rules to apply, the region has done its maximal parallel step and proceeds to the state *SYNCHRONIZE*. Other states are just technical - we need to implement sending objects between membranes and preparing for the next maximal parallel step by unmarking newly created objects in the current maximal parallel step, which have been marked to prevent double rewriting in one step.

- *RUN*: Rewriting occurs. Objects that are to be sent to parent membrane are directly sent because the parent membrane is already in *RUN* or *SYNCHRONIZE* phase, so the  $a'$  symbols that are sent don't break anything. But objects that are to be sent down, can't be sent immediately because child membranes can be in previous phase waiting to restore symbols from previous step. Current symbols could interfere with them and be rewritten twice in this step. Such objects are only marked as "to be sent down":  $a^\downarrow$
- *SYNCHRONIZE*: Rewriting has ended and membrane is waiting to get signal *SYNCED* from parent membrane to continue to next step.
- *SENDDOWN*: Signal was caught and now all descendant membranes are in *SYNCHRONIZE* phase so  $a^\downarrow$  can be sent down.
- *RESTORE*: All  $a'$  symbols are restored to  $a$ , so the next step of rewriting can take place.
- For every rule  $r_i \in R$  such that

$$r_i = a_1^{M(a_1)} a_2^{M(a_2)} \dots a_n^{M(a_n)} \rightarrow a_1^{N(a_1)} a_2^{N(a_2)} \dots a_n^{N(a_n)}$$

we will have the following rules:

$$\begin{aligned} & a_1^{M(a_1)-m_1} \dot{a}_1^{m_1} a_2^{M(a_2)-m_2} \dot{a}_2^{m_2} \dots a_n^{M(a_n)-m_n} \dot{a}_n^{m_n} | RUN \\ & \rightarrow a_1^{N(a_1)} a_2^{N(a_2)} \dots a_n^{N(a_n)} | RUN \end{aligned}$$

There will be such rule for each  $0 \leq m_i \leq M(a_i)$ . It represents the idea that  $\dot{a}$  can be used in rewriting in the same way as  $a$ . Right side of the rules contains symbols  $a'$ , that prevents the symbols to be rewritten again.

- For every symbol  $a \in V$  we will have the following rules:

$$a|RUN \rightarrow \dot{a}|RUN|_{\neg \dot{a}}$$

There will be max one occurrence of  $\dot{a}$ .

- For every rule  $r_i \in R$  there will be a rule that detects if the rule  $r_i$  is not usable. According to left side of the rule  $r_i$ , symbol  $UNUSABLE_i$  will be created when there is not enough objects to fire the rule  $r_i$ . It means that left side of rule  $r_i$  requires more instances of some object than are present in membrane.

If the left side is of type:

- $a$ : It is a context free rule. The rule can't be used if there is no occurrence of  $a$  nor  $\dot{a}$ .

$$RUN \rightarrow UNUSABLE_i|RUN|_{\neg\{UNUSABLE_i, \dot{a}\}}$$

- $ab$ : It is a cooperative rule with two distinct objects on the left side. The rule can't be used if there is one of them missing.

$$RUN \rightarrow UNUSABLE_i|RUN|_{\neg\{UNUSABLE_i, \dot{a}\}}$$

$$RUN \rightarrow UNUSABLE_i|RUN|_{\neg\{UNUSABLE_i, \dot{b}\}}$$

- $a^2$ : It is a cooperative rule with two same objects. The rule can't be used if there is at most one occurrence of the symbol. That happens if there is no occurrence of  $a$ . There can still be  $\dot{a}$ , but at most one occurrence.

$$RUN \rightarrow UNUSABLE_i|RUN|_{\neg\{UNUSABLE_i, \dot{a}\}}$$

- For every membrane with label  $i$  there will be rule:

$$UNUSABLE_1|UNUSABLE_2|\dots|UNUSABLE_m|RUN \\ \rightarrow SYNCHRONIZE|SYNCTOKEN_i \uparrow$$

If no rule can be used, maximal parallel step in the region is completed so it goes to synchronization phase and sends a synchronization token to the parent membrane.

- For every membrane there will be a rule:

$$\begin{aligned} & \text{SYNCHRONIZE} | \text{SYNCTOKEN}_j \\ & \rightarrow \text{SYNCHRONIZE} | \text{SYNCTOKEN}_j \uparrow \end{aligned}$$

Membrane resends all sync token to parent membrane.

- In skin membrane there is a rule which collects all the synchronization tokens from all membranes  $1 \dots k$  and then sends down signal that synchronization is complete. But before that, there can be some symbols that should be sent down, but they weren't, because the region below could have not started the rewriting phase that time. The result was just marked with  $a^\downarrow$ .

$$\begin{aligned} & \text{SYNCTOKEN}_1 | \dots | \text{SYNCTOKEN}_k | \text{SYNCHRONIZE} \\ & \rightarrow \text{SENDDOWN} \end{aligned}$$

- Every membrane other than skin membrane have to receive the signal to go to senddown phase:

$$\text{SYNCHRONIZE} | \text{SYNCED} \rightarrow \text{SENDDOWN}$$

- Every membrane will have rules for every symbol  $a \in V$  to send down all unsent object that should have been sent down:

$$\text{SENDDOWN} | a^\downarrow \rightarrow \text{SENDDOWN} | a' \downarrow$$

- Every membrane will have rule for detecting when all such objects have been sent and it goes to restore phase:

$$\text{SENDDOWN} \rightarrow \text{RESTORE} | \neg \{a_i^\downarrow | 1 \leq i \leq n\}$$

- In restore phase all symbols  $a'$  will be rewritten to  $a$  in order to be able to be rewritten in next maximal parallel step:

$$\text{RESTORE} | a' \rightarrow \text{RESTORE} | a$$

- When restore phase ends, it sends down signal that all membranes have been synchronized and next phase of rewriting has began in upper membranes:

$$\text{RESTORE} \rightarrow \text{RUN} | \text{SYNCED} \downarrow | \neg \{a'_i | 1 \leq i \leq n\} \cup \{GONE_i | 1 \leq i \leq n\}$$



Phase of membrane is represented by object so the region is never empty and by the Lemma 5.1.1 the rules with set of inhibitors can be simulated by single inhibitors.

We have also reached this result in the accepting case by simulation of register machines.

**Theorem 5.1.3** *Sequential P systems with inhibitors can simulate register machines and thus equal PsRE.*

**Proof 5.1.3** Suppose we have a  $n$ -register machine  $M = (n, P, i, h)$ . In our simulation we will have a membrane structure consisting of one membrane and the contents of register  $j$  will be represented by the multiplicity of the object  $a_j$ .

We will have P system  $(V, \mu, w, R)$ , where:

- $V$  is an alphabet consisting of symbols that represent registers  $a_1, \dots, a_n$  and instruction labels in  $Lab(M)$ ,
- $\mu$  is a membrane structure consisting of only one membrane,
- $w$  is initial contents of the membrane. It contains symbols for the input for the machine  $a_i^{n_i}$  where  $n_i$  is initial state of register with label  $i$  and initial instruction  $e \in Lab(M)$ .
- $R$  is the set of rules in the skin membrane:

For all instructions of type  $(e : add(j), f)$  we will have rule:

$$- e \rightarrow a_j | f.$$

For all instructions of type  $(e : sub(j), f, z)$  we will have rules:

$$\begin{aligned} &- e | a_j \rightarrow f \text{ and} \\ &- e \rightarrow z | \neg a_j. \end{aligned}$$

And finally halting rules:

$$\begin{aligned} &- h | a_j \rightarrow h | \# \text{ for all } a \leq j \leq n, \\ &- \# \rightarrow \#, \end{aligned}$$

When the halting instruction is reached, if there is an object present in the membrane, the hash symbol  $\#$  is created and it will cycle forever. If there is no object present, there is no rule to apply and computation will halt. It corresponds to the condition that all registers should be empty when halting.

## 5.2 Universality results of P systems with vacuum

**Theorem 5.2.1** *The sequential P system with Vacuum is universal.*

**Proof 5.2.1** We can simulate the variant of P system where the only cooperative rule is of type  $a|a \rightarrow b$ . According to [?] the variant, where the only cooperative rule is when both objects are the same, is universal. If there is no rule  $a \rightarrow b$ , we can rewrite  $a$  to  $a'$  so we can mark all present symbols.  $a'$  symbols are kept in special membrane so the Vacuum can be created in main membrane and we can synchronize.

But we won't do this madness again.

Instead, we will try to prove universality by simulating the register machine. We need to detect when the current register is empty. If there was a symbol for every register as in the proof 5.1.3, the Vacuum would be created only if all registers are empty. But the *sub()* instruction need to detect when one concrete register is empty.

We will have a membrane for each register. That membrane will be contained in the skin membrane. The number of objects in membrane  $i$  will correspond to the value of register  $i$ .

The alphabet will consist of instruction labels and register counter  $a$ . The skin membrane will only have an instruction label. It is sent to corresponding membrane where the instruction is executed. Then, the following instruction is sent back to the skin membrane.

We will have following rules in the skin membrane:

- $e \rightarrow e \downarrow_j$  for an instruction of type  $e : add(j), f$  or  $e : sub(j), f, z$  and
- $h \rightarrow h \downarrow$  for a halting instruction  $h$ .

And in non-skin membranes:

- $e \rightarrow a|f \uparrow$  instructions of type  $e : add(j), f$ ,
- $e|a \rightarrow f \uparrow$  for instructions of type  $e : sub(j), f, z$ ,
- $e|VACUUM \rightarrow z \uparrow$  for instructions of type  $e : sub(j), f, z$ , and
- $h|a \rightarrow h|a$

When halting, if there is an nonempty register, it will cycle forever with the last rule. However, if all registers are empty, the halting instruction label will stay in all membranes and the computation will halt.

# Conclusions

We have studied several variants of sequential P systems in order to obtain universality without using maximal parallelism. A variant with rewriting rules that can use inhibitors was shown to be universal in both generating and accepting case. The generating model is able to simulate maximal parallel P system and the accepting model can simulate a register machine.

In addition, we have defined a new notion of vacuum, which is immediately created in the region that becomes empty. A new P system variant was introduced, which allowed the vacuum to be used on the left side of P system rewriting rules. The accepting case of this variant was shown to be universal by direct implementation of a register machine.

A lot of other variants deserve to be combined with the vacuum, so we suggest to research them more thoroughly (non-cooperative rules, rules with priorities, decaying objects, deterministic steps, . . . ).