WMC6

6th International Workshop on Membrane Computing

Vienna, July 18-21, 2005

TU WIEN

Pre-Proceedings of the

6th International Workshop on Membrane Computing

WMC6

Vienna, July 18-21, 2005

edited by

Rudolf Freund
Georg Lojka
Marion Oswald
Gheorghe Paun

Title: Pre-Proceedings of the 6th International Workshop on
Membrane Computing WMC6

Published by: Vienna University of Technology
Faculty of Informatics
Institute of Computer Languages
Favoritenstraße 9-11
1040 Wien, Austria

Published in July 2005

Contents

Preface

The **6th Workshop on Membrane Computing** is held in Vienna from July 18th to July 21st, 2005, under the auspices of the European Molecular Computing Consortium. It continues the tradition of the annual series that started with the **Workshop on Multiset Processing WMP-CdeA 2000** and was followed by the **Workshop on Membrane Computing WMC-CdeA 2001** and **WMC-CdeA 2002** in Curtea de Argeş as well as the **Workshop on Membrane Computing WMC4** held in Tarragona 2003 and **WMC5** held in Milano 2004.

The present volume contains five invited lectures and twenty-eight selected contributions that were reviewed by three referees each. Based on these reviews, the selected papers were accepted as full papers or as extended or short abstracts. The final volume will be published after the workshop in the Springer series *Lecture Notes in Computer Science*. The pre-proceedings of the previous workshops can be found at the P Systems Web Page `http://psystems.disco.unimib.it`.

We thank the members of the program committee as well as the additional referees for their quick reviewing and for their helpful remarks which allowed for considerable improvements of the accepted papers.

Vienna, July 2005

Rudolf Freund
Georg Lojka
Marion Oswald
Gheorghe Păun

Program committee

Erzsébet Csuhaj-Varjú
Rudolf Freund – Co-Chair
Marian Gheorghe
Hendrik Jan Hoogeboom
Oscar H. Ibarra
Natasha Jonoska
Kamala Krithivasan
Vincenzo Manca
Maurice Margenstern
Gheorghe Păun – Co-chair
Mario J. Pérez-Jiménez
Grzegorz Rozenberg
Petr Sosík
Claudio Zandron

Additional Referees

Artiom Alhazov
Daniela Besozzi
Ludek Cienciala
Alica Kelemenová
Dario Pescini
Raghavan Rama
György Vaszil
Hsu-Chun Yen

Organising committee

Aneta Binder
Franziska Freund
Rudolf Freund
Theresia Gschwandtner
Georg Lojka
Marion Oswald

Sponsored by:

Institute of Computer Languages, Vienna University of Technology

BEKO

Part I

Invited Papers

Structural Operational Semantics of P Systems

Oana ANDREI[1], **Gabriel CIOBANU**[2], **Dorel LUCANU**[1]

[1] "A.I.Cuza" University of Iaşi, Faculty of Computer Science
E-mail: {oandrei,dlucanu}@info.uaic.ro

[2]Romanian Academy, Institute of Computer Science, Iaşi
and Research Institute "e-Austria" Timişoara, Romania
E-mail: gabriel@iit.tuiasi.ro

Abstract

The paper formally describes an operational semantics of P systems. We present an abstract syntax of P systems, then the notion of configurations, and we define the sets of inference rules corresponding to the three stages of an evolution step: maximal parallel rewriting, parallel communication, and parallel dissolving. Several results assuring the correctness of each set of inference rules are also presented. Finally, we define simulation and bisimulation relations between P systems.

1 Introduction

Structural operational semantics (SOS) provides a framework of defining a formal description of a computing system. It is intuitive and flexible, and it becomes more attractive during the years by the developments presented by G.Plotkin [14], G.Kahn [7], and R.Milner [9]. In P systems a computation is regarded as a sequence of parallel application of rules in various membranes, followed by a communication step and a dissolving step. A SOS of the P systems emphasizes the deductive nature of the membrane computing by describing the transition steps by using a set of inference rules. Considering a set $\mathcal{R}$ of inference rules of form $\frac{premises}{conclusion}$, we can describe the computation of a P system as a deduction tree. As a consequence, given two configurations C, C of a P system, SOS provides a formal method to show that C is obtained in a transition step from C, i.e., $\mathcal{R} \vdash C \Rightarrow C$.

Structural operational semantics is introduced by Plotkin [14], and it becomes a well-known framework for specifying the semantics of concurrent systems. Configurations are states of transition systems, and computations consists of sequences of transitions between configurations, and terminating (if it terminates) in a final configuration. In the usual style of structural operational semantics, computations proceed by small steps through intermediate configurations. In P systems, the operational semantics is given in a rather big-step style, each step representing the collection of parallel steps due to the maximal parallelism principle.

In this paper we present a structural operational semantics of P systems. First we give an abstract syntax of P systems, and then we define an appropriate notion of configuration. We introduce three sets of inference rules corresponding to distinct phases in the evolution of a P system. We prove the soundness of our inference rules. The (bi)simulation relations between P systems are also defined; they allow to compare the evolution behaviour of two P systems.

The structure of the paper is as follows. Section 2 presents briefly the P systems. Section 3 represents the principal part of the paper; it presents the structural operational semantics of the P systems. Conclusion and references end the paper.

2 Definition of P Systems

P systems represent a new abstract model of parallel and distributed computing inspired by cell compartments and molecular membranes [12]. A cell is divided in various compartments, each compartment with a different task, and all of them working simultaneously to accomplish a more general task of the whole system. P systems provide a nice abstraction for parallel systems, and a suitable framework for distributed and parallel algorithms [3].

A detailed description of the P systems can be found in [12]. A *P system* consists of a hierarchy of membranes that do not intersect, with a distinguishable membrane, called the *skin membrane*, surrounding them all. A membrane without any other membranes inside is *elementary*, while a non-elementary membrane is a *composite* membrane. The membranes produce a demarcation between *regions*. For each membrane there is a unique associated region: the space delimited from above by it and from below by the membranes placed directly inside, if any exists. The space outside the skin membrane is called the *outer region*. Because of this one-to-one correspondence we sometimes use membrane instead of region. Regions contain

multisets of *objects*, *evolution rules* and possibly other membranes. Only rules in a region delimited by a membrane act on the objects in that region. The multisets of objects from a region correspond to the "chemicals swimming in the solution in the cell compartment", while the rules correspond to the "chemical reactions possible in the same compartment". The rules must contain target indications, specifying the membrane where the new objects obtained after applying the rule are sent. The new objects either remain in the same region when they have a *here* target, or they pass through membranes, in two directions: they can be sent *out* of the membrane which delimits a region from outside, or can be sent *in* one of the membranes which delimit a region from inside, precisely identified by its label. In a step, the objects can pass only through one membrane. We say that objects are enclosed in messages together with the target indication. Therefore we have *here* messages of typical form $(w, here)$ with w a possibly empty multiset of objects, *out* messages of typical form (w, out), and *in* messages of typical form (w, in_L), both with w a non-empty multiset of objects.

For the sake of simplicity, we consider that the messages with the same target indication merge into one message, such that

$(w_1, here) \dots (w_n, here) = (w, here)$,

$(w_1, in_L) \dots (w_n, in_L) = (w, in_L)$, and

$(w_1, out) \dots (w_n, out) = (w, out)$,

where $w = w_1 \dots w_n$.

A membrane is *dissolved* by the symbol δ resulted after a rule application; this action is important when discussing about adaptive executions. When such an action takes place, the membrane disappears, its contents (objects and membranes) remain free in the membrane placed immediately outside, and the evolution rules of the dissolved membranes are lost. The skin membrane is never dissolved. The application of evolution rules is done in parallel, and it is eventually regulated by *priority* relationships between rules.

A P system has a certain structure represented by a tree (with skin as its root and elementary membranes as leaves), or by a string of correctly matching parentheses, placed in a unique pair of matching parentheses; each pair of matching parentheses corresponds to a membrane. Graphically, a membrane structure is represented by a Venn diagram in which two sets can be either disjoint, or one is a subset of the other. This representation makes clear that the order of sibling membranes is irrelevant (as they float around), while, on the contrary, the inclusion relationship (or parent-child relationship in the tree-like representation) between membranes is essential. The membranes (and the corresponding regions) are labelled in a one-to-

one manner with labels from a given set, usually ranging from 1 to the total number of membranes.

Formally, a *P system* is a structure $\Pi = (O, \mu, w_1, \ldots, w_m, (R_1, \rho_1), \ldots, (R_m, \rho_m), i_o)$, where:

(i) O is an alphabet of objects;

(ii) μ is a membrane structure;

(iii) w_i are the initial multisets over O associated with the regions defined by μ;

(iv) R_i are finite sets of evolution rules over O associated with the membranes, of typical form $u \rightarrow v$, with u a multiset over O and v consisting of messages and/or the dissolving symbol δ;

(v) ρ_i is a partial order relation over R_i, specifying a *priority* relation among the rules: $(r_1, r_2) \in \rho_i$ iff $r_1 > r_2$ (i.e., r_1 has a higher priority than r_2);

(vi) i_0 is either a number between 1 and m specifying the *output* membrane of Π, or it is equal to 0 indicating that the output is the outer region.

Since the skin is not allowed to be dissolved, we consider that the rules of the skin do not involve δ. These are the *general P systems*, or *transition P systems*; many other variants and classes were introduced [12].

The membranes preserve the initial labelling, evolution rules and priority relation among them in all subsequent configurations. Therefore in order to describe a membrane we consider its label and the current multiset of objects together with its structure. We use the mappings *rules* and *priority* to associate to a membrane label the set of evolution rules and the priority relation : $rules(L_i) = R_i$, $priority(L_i) = \rho_i$, and the projections L and w which return from a membrane its label and its current multiset, respectively.

Notation. If X is a set, then X_c^* denotes the set of the finite multisets defined over X, and X_c^+ denotes X_c^* without the empty multiset. This notations are inspired by the one-to-one correspondence from the set of the finite multisets defined over X onto the free commutative monoid generated by X.

Formally, the set of *membranes for a P system* Π, denoted by $\mathcal{M}(\Pi)$, and *the membrane structure* are inductively defined as follows:

- if L is a label, and w is a multiset over $O \cup (O_c^* \times \{here\}) \cup (O_c^+ \times \{out\}) \cup \{\delta\}$, then $\langle\ L \mid w\ \rangle \in \mathcal{M}(\Pi)$; $\langle\ L \mid w\ \rangle$ is called *simple (or elementary) membrane*, and it has the structure $\langle\rangle$;

- if $M_1, \ldots, M_n \in \mathcal{M}(\Pi)$ with $n \geq 1$, the structure of M_i is μ_i for all $i \in [n]$, L is a label, w is a multiset over $O \cup (O_c^* \times \{here\}) \cup (O_c^+ \times \{out\}) \cup (O_c^+ \times \{in_{L(M_j)} | j \in [n]\}) \cup \{\delta\}$, then $\langle\, L \mid w\,;\, M_1, \ldots, M_n \,\rangle \in \mathcal{M}(\Pi)$; $\langle\, L \mid w\,;\, M_1, \ldots, M_n \,\rangle$ is called *a composite membrane*, and it has the structure $\langle \mu_1, \ldots, \mu_n \rangle$.

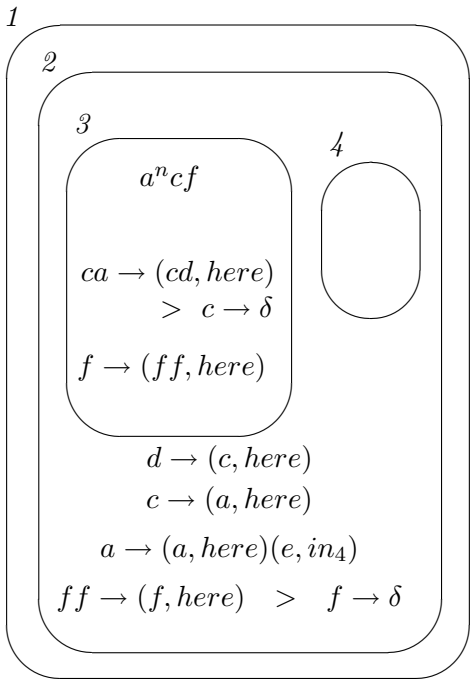
A finite multiset of membranes is usually written as $M_1, \ldots, M_n$. We denote by $\mathcal{M}^+(\Pi)$ the set of non-empty finite multisets of membranes. The union of two multisets of membranes $M_+ = M_1, \ldots, M_m$ and $N_+ = N_1, \ldots, N_n$ is written as $M_+, N_+ = M_1, \ldots, M_m, N_1, \ldots, N_n$. An element from $\mathcal{M}^+(\Pi)$ is either a membrane, or a set of sibling membranes.

A *committed configuration* for a P system Π is a skin membrane which has no messages and no dissolving symbol δ, i.e., the multisets of all regions are elements in O_c^*. We denote by $\mathcal{C}(\Pi)$ the set of committed configurations for Π, and it is a proper subset of $\mathcal{M}^+(\Pi)$. We have $C \in \mathcal{C}(\Pi)$ iff C is a skin membrane of Π and $w(M)$ is a multiset over O for each membrane M in $\mathcal{C}$.

An *intermediate configuration* is a skin membrane in which we have messages or the dissolving symbol δ. The set of intermediate configurations is denoted by $\mathcal{C}^{\#}(\Pi)$. We have $C \in \mathcal{C}^{\#}(\Pi)$ iff C is a skin membrane of Π such that there is a membrane M with $w(M) = w'w''$, $w' \in (Msg(O) \cup \{\delta\})_c^+$, and $w'' \in O_c^*$. By $Msg(O)$ we denote the set $(O^* \times \{here\}) \cup (O^+ \times \{out\}) \cup (O^+ \times \{in_L(M)\})$.

A *configuration* is either a committed configuration or an intermediate configuration. Each P system has an initial committed configuration which is characterized by the initial multiset of objects for each membrane and the initial membrane structure of the system.

Example 1 *We give an example of a deterministic P system computing n^2 for a given n. The initial configuration of such a system is:*



and it is written as $\langle\, 1 \mid empty\,;\, \langle\, 2 \mid empty\,;\, \langle\, 3 \mid a^n cf\,\rangle, \langle\, 4 \mid empty\,\rangle\,\rangle\,\rangle$.

3 Structural Operational Semantics of P Systems

Structural operational semantics descriptions of systems start from abstract syntax. Specifications of abstract syntax introduce symbols for syntactic sets, meta-variables ranging over those sets, and notation for constructor functions. Some of the syntactic sets are usually regarded as basic, and left open or described only informally. The abstract syntax for P systems is given as follows:

Objects:	$o \in O$
Multisets of objects:	$w \in O_c^*$
Labels:	$L \in \{Skin\} \cup \mathcal{L}$
Messages:	$(w, here), (w, in_L), (w, out) \in Msg(O)$
Dissolving symbol:	δ
Membrane contents:	$w \in (O \cup Msg(O) \cup \{\delta\})_c^*$

Membranes:	$M \in \mathcal{M}(\Pi)$
	$M ::= \langle L \mid w \rangle \mid \langle L \mid w; M_+ \rangle$
Sibling membranes:	$M_+ \in \mathcal{M}^+(\Pi) = \mathcal{M}(\Pi)_c^+$
Committed configurations:	$C \in \mathcal{C}(\Pi)$
Intermediate configurations:	$C \in \mathcal{C}^\#(\Pi)$

In structural operational semantics, the evolution of systems is modelled by a transition system specified inductively, by rules. The transition system for a P system Π is intuitively defined as follows. For two committed configurations C_1 and C_2 of Π, we say that there is a *transition* from C_1 to C_2, and write $C_1 \Rightarrow C_2$, if the following *steps* are executed in the following given order:

1. *the maximal parallel rewriting step*, written $C_1 \stackrel{mpr}{\Longrightarrow} C_2'$, is consisting in non-deterministically assigning objects to evolution rules in every membrane, and executing them in a maximal parallel manner;

2. *the parallel communication of objects through membranes*, written $C_2' \stackrel{tar}{\Longrightarrow} C_2''$, is consisting in sending the existing messages;

3. *the parallel membrane dissolving*, written $C_2'' \stackrel{\delta}{\Longrightarrow} C_2$, consisting in dissolving the membranes which contain the δ symbol.

The last two steps are executed only if there are messages or δ symbols resulted from the first step, respectively. If the first step is not possible, consequently neither the other two steps, then we say that the system has reached a *halting configuration*. A halting configuration is always a committed one.

Next we present in the terms of SOS each of the three steps.

3.1 Maximal Parallel Rewriting Step

We can pass from a configuration to another one by using the evolution rules. This is done in parallel: all objects, from all membranes, which can be the subject of local evolution rules, as prescribed by the priority relation, should evolve simultaneously. The rules of a membrane are using its current objects as much as this is possible in a parallel and non-deterministic way. However, an object introduced by a rule cannot evolve at the same step by means of another rule. The use of a rule $u \to v$ in a region with a multiset w means to subtract the multiset identified by u from w, and then adding the objects of v according to the form of the rule.

We denote the maximal parallel rewriting on membranes by $\stackrel{mpr}{\Longrightarrow}$ and by $\stackrel{mpr}{\Longrightarrow}_L$ the maximal parallel rewriting over the multisets of objects of the membrane labelled by L (we omit the label whenever it is clear from the context). SOS definition of $\stackrel{mpr}{\Longrightarrow}$ uses two predicates regarding mpr-irreducibility and (L, w)-consistency.

Definition 1 *The irreducibility property w.r.t. the maximal parallel rewriting relation for multisets of objects, messages, and δ, for membranes, and for sets of sibling membranes is defined as follows:*

- *a multiset w consisting only of objects is* **L-irreducible** *iff there are no rules in $rules(L)$ applicable to w w.r.t. the priority relation $priority(L)$;*
- *a multiset containing at least a message or the dissolving symbol δ is* **L-irreducible**;
- *a simple membrane $\langle\, L \,|\, w \,\rangle$ is* **mpr-irreducible** *iff w is L-irreducible;*
- *a non-empty set of sibling membranes $M_1, \ldots, M_n$ is* **mpr-irreducible** *iff M_i is mpr-irreducible, for every $i \in [n]$;*
- *a composite membrane $\langle\, L \mid w\, ;\, M_1, \ldots, M_n \,\rangle$, with $n \geq 1$, is* **mpr-irreducible** *iff w is L-irreducible, and the set of sibling membranes $M_1, \ldots, M_n$ is mpr-irreducible.*

Definition 2 *Let M be a membrane labelled by L, and w a multiset of objects. A non-empty multiset $R = (u_1 \to v_1, \ldots, u_n \to v_n)$ of evolution rules is* **(L, w)-consistent** *iff:*

- $R \subseteq rules(L)$,
- $w = u_1 \ldots u_n z$, *so each rule $r \in R$ is applicable on w,*
- $(\forall r \in R, \forall r' \in rules(L))$ r' *applicable on w implies* $(r', r) \notin priority(L)$,
- $(\forall r', r'' \in R)$ $(r', r'') \notin priority(L)$,
- *the dissolving symbol δ has at most one occurrence in the multiset $v_1 \ldots v_n$.*

The **maximal parallel rewriting relation** $\stackrel{mpr}{\Longrightarrow}$ is defined by the following inference rules:

For each $w \in O_c^+$ and L-irreducible $z \in O_c^*$ such that $w = u_1 \dots u_n z$, and (L, w)-consistent rules $(u_1 \to v_1, \dots, u_n \to v_n)$,

$$(\mathbf{R_1}) \ \frac{}{w \stackrel{mpr}{\Longrightarrow}_L v_1 \dots v_n z}$$

For each $w \in O_c^+$, $w' \in (O \cup Msg(O) \cup \{\delta\})_c^+$,

$$(\mathbf{R_2}) \ \frac{w \stackrel{mpr}{\Longrightarrow}_L w'}{\langle\, L \mid w \,\rangle \stackrel{mpr}{\Longrightarrow} \langle\, L \mid w' \,\rangle}$$

For each $w \in O_c^+$, $w' \in (O \cup Msg(O) \cup \{\delta\})_c^+$, $M_+, M'_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_3}) \ \frac{w \stackrel{mpr}{\Longrightarrow}_L w', M_+ \stackrel{mpr}{\Longrightarrow} M'_+}{\langle\, L \mid w \,;\, M_+ \,\rangle \stackrel{mpr}{\Longrightarrow} \langle\, L \mid w' \,;\, M'_+ \,\rangle}$$

For each $w \in O_c^+$, $w' \in (O \cup Msg(O) \cup \{\delta\})_c^+$, and mpr-irreducible $M_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_4}) \ \frac{w \stackrel{mpr}{\Longrightarrow}_L w'}{\langle\, L \mid w \,;\, M_+ \,\rangle \stackrel{mpr}{\Longrightarrow} \langle\, L \mid w' \,;\, M_+ \,\rangle}$$

For each L-irreducible $w \in O_c^*$, $M_+, M'_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_5}) \ \frac{M_+ \stackrel{mpr}{\Longrightarrow} M'_+}{\langle\, L \mid w \,;\, M_+ \,\rangle \stackrel{mpr}{\Longrightarrow} \langle\, L \mid w \,;\, M'_+ \,\rangle}$$

For each $M, M' \in \mathcal{M}(\Pi)$, $M_+, M'_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_6}) \ \frac{M \stackrel{mpr}{\Longrightarrow} M', M_+ \stackrel{mpr}{\Longrightarrow} M'_+}{M, M_+ \stackrel{mpr}{\Longrightarrow} M', M'_+}$$

For each $M, M' \in \mathcal{M}(\Pi)$, and mpr-irreducible $M_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{R_7}) \ \frac{M \stackrel{mpr}{\Longrightarrow} M'}{M, M_+ \stackrel{mpr}{\Longrightarrow} M', M_+}$$

Example 2 *Considering the P system of Example 1, the inference tree for* $\langle 1|empty; \langle 2|aaf; \langle 4|ee\rangle\rangle\rangle \stackrel{mpr}{\Longrightarrow} \langle 1|empty; \langle 2|(aa, here)(ee, in_4)\delta; \langle 4|ee\rangle\rangle\rangle$ *is:*

$$(R_5)\ \frac{(R_4)\ \dfrac{(R_1)\ \dfrac{}{aaf \overset{mpr}{\Longrightarrow}_2 (aa, here)(ee, 4)\delta}}{\langle\, 2 \mid aaf\,;\, \langle\, 4 \mid ee\,\rangle\,\rangle \overset{mpr}{\Longrightarrow} \langle\, 2 \mid (aa, here)(ee, in_4)\delta\,;\, \langle\, 4 \mid ee\,\rangle\,\rangle}}{\langle\, 1 \mid empty\,;\, \langle\, 2 \mid aaf\,;\, \langle\, 4 \mid ee\,\rangle\,\rangle\,\rangle \overset{mpr}{\Longrightarrow} \langle\, 1 \mid empty\,;\, \langle\, 2 \mid (aa, here)(ee, in_4)\delta\,;\, \langle\, 4 \mid ee\,\rangle\,\rangle\,\rangle}$$

Lemma 1 *If $w \overset{mpr}{\Longrightarrow}_L w'$, then w' is L-irreducible.*

Proof. We get $w \overset{mpr}{\Longrightarrow}_L w'$ only applying (R_1) using a (L, w)-consistent multiset of rules. Then we have $w' = w''z$ such that $w'' \in (Msg(O) \cup \{\delta\})^+_c$, and $z \in O^*_c$ is L-irreducible. Then w' is L-irreducible by definition because it contains messages or δ. □

Lemma 2 *If $M_+ \overset{mpr}{\Longrightarrow} M'_+$ then M'_+ is mpr-irreducible.*

Proof. Let M_+, M'_+ be two non-empty sets of membranes such that $M_+ \overset{mpr}{\Longrightarrow} M'_+$. We prove that M'_+ is mpr-irreducible by induction on the depth of the associated inference tree. We consider all possible cases for the final step of the inference:

(i) $M_+ \overset{mpr}{\Longrightarrow} M'_+$ is inferred by (R_2). Then $M_+ = \langle L | w \rangle$ and $M'_+ = \langle L | w' \rangle$ with $w \overset{mpr}{\Longrightarrow} w'$. By Lemma 1 w' is L-irreducible, therefore M'_+ is mpr-irreducible by definition.

(ii) $M_+ \overset{mpr}{\Longrightarrow} M'_+$ is inferred by (R_3). Then $M_+ = \langle\, L \mid w\,;\, N_+\,\rangle$ and $M'_+ = \langle\, L \,|\, w'\,;\, N'_+\,\rangle$ with $w \overset{mpr}{\Longrightarrow}_L w'$ inferred by (R_1), and $N_+ \overset{mpr}{\Longrightarrow} N'_+$ inferred by a shorter inference tree. By Lemma 1 w' is L-irreducible, and by inductive hypothesis, N'_+ is mpr-irreducible. Therefore M'_+ is mpr-irreducible by definition.

(iii) $M_+ \overset{mpr}{\Longrightarrow} M'_+$ is inferred by (R_4). Then $M_+ = \langle\, L \mid w\,;\, N_+\,\rangle$ and $M'_+ = \langle\, L \mid w'\,;\, N_+\,\rangle$ with $w \overset{mpr}{\Longrightarrow} w'$ (therefore w' is L-irreducible by Lemma 1) and N_+ mpr-irreducible. By definition we obtain that M'_+ is mpr-irreducible.

(iv) $M_+ \overset{mpr}{\Longrightarrow} M'_+$ is inferred by (R_5). Then $M_+ = \langle\, L \mid w\,;\, N_+\,\rangle$ and $M'_+ = \langle\, L \,|\, w\,;\, N'_+\,\rangle$ with w L-irreducible and $N_+ \overset{mpr}{\Longrightarrow} N'_+$ inferred by a shorter inference tree. By inductive hypothesis N'_+ is mpr-irreducible, therefore M'_+ is mpr-irreducible.

(v) $M_+ \stackrel{mpr}{\Longrightarrow} M'_+$ is inferred by (R_6). Then $M_+ = M, N_+$ and $M'_+ = M', N'_+$ where $M \stackrel{mpr}{\Longrightarrow} M'$ and $N_+ \stackrel{mpr}{\Longrightarrow} N'_+$ are inferred by shorter inference trees. By inductive hypothesis M' and N'_+ are mpr-irreducible, therefore M', N'_+ is mpr-irreducible.

(vi) $M_+ \stackrel{mpr}{\Longrightarrow} M'_+$ is inferred by (R_7). Then $M_+ = M, N_+$ and $M'_+ = M', N_+$ where $M \stackrel{mpr}{\Longrightarrow} M'$ is inferred by a shorter inference tree, and N_+ is mpr-irreducible. By inductive hypothesis M' is mpr-irreducible, therefore M', N_+ is mpr-irreducible by definition.

□

Theorem 1 *Let* Π *be a P system. If* $C \in \mathcal{C}(\Pi)$ *and* $C \stackrel{mpr}{\Longrightarrow} C'$*, then* $C' \in \mathcal{C}^{\#}(\Pi)$ *and* C' *is mpr-irreducible.*

The proof of the theorem follows easily from Lemma 2.

3.2 Parallel Communication of Objects

Communication through two membranes M_1 and M_2 can take place only if one is inside the other.

We say that a multiset w is *here-free/in_L-free/out-free* if it does not contain any *here/in_L/out* messages, respectively. For w a multiset of objects and messages, we introduce the operations *obj*, *here*, *out*, and in_L as follows:

$$
\begin{array}{l}
obj(w) \text{ is obtained from } w \text{ by removing all messages,} \\
here(w) = \left\{ \begin{array}{ll} empty & \text{if } w \text{ is } here\text{-free,} \\ w'' & \text{if } w = w'(w'', here) \wedge w' \text{ is } here\text{-free,} \end{array} \right. \\
out(w) = \left\{ \begin{array}{ll} empty & \text{if } w \text{ is } out\text{-free,} \\ w'' & \text{if } w = w'(w'', out) \wedge w' \text{ is } out\text{-free,} \end{array} \right. \\
in_L(w) = \left\{ \begin{array}{ll} empty & \text{if } w \text{ is } in_L\text{-free,} \\ w'' & \text{if } w = w'(w'', in_L) \wedge w' \text{ is } in_L\text{-free.} \end{array} \right.
\end{array}
$$

We recall that all the messages with the same target merge in one message.

Definition 3 *The* **tar-irreducibility** *property for membranes and for sets of sibling membranes is defined as follows:*

1. *a simple membrane* $\langle\, L \mid w \,\rangle$ *is* **tar-irreducible** *iff* $L \neq Skin \vee (L = Skin \wedge w$ *is out-free);*

2. *a non-empty set of sibling membranes* $M_1, \ldots, M_n$ *is* **tar-irreducible** *iff* M_i *is tar-irreducible, for every* $i \in [n]$*;*

3. *a composite membrane* $\langle L|w; M_1, \ldots, M_n\rangle$, $n \geq 1$, *is* **tar-irreducible** *iff:*

   (a) $L \neq Skin \vee (L = Skin \wedge w$ *is out-free*$)$,

   (b) w *is* $in_{L(M_i)}$*-free, for every* $i \in [n]$,

   (c) *for all* $i \in [n]$, $w(M_i)$ *is out-free,*

   (d) *the set of sibling membranes* $M_1, \ldots, M_n$ *is tar-irreducible.*

Notation. We treat the messages of form $(w', here)$ as a particular communication inside their membranes consisting in substitution of $(w', here)$ by w'. We denote by $\overline{w}$ the multiset obtained by replacing $(here(w), here)$ by $here(w)$ in w. For instance, if $w = a\,(bc, here)\,(d, out)$ then $\overline{w} = abc\,(d, out)$, where $here(w) = bc$. We note that $in_L(\overline{w}) = in_L(w)$, and $out(\overline{w}) = out(w)$.

The **parallel communication relation** $\stackrel{tar}{\Longrightarrow}$ is defined by the following inference rules:

For each tar-irreducible $M_1, \ldots, M_n \in \mathcal{M}^+(\Pi)$ and multiset w such that $here(w) \neq empty$, or $L = Skin \wedge out(w) \neq empty$, or it exists $i \in [n]$ with $in_{L(M_i)}(w)out(w(M_i)) \neq empty$ or $here(w(M_i)) \neq empty$,

$$(\mathbf{C_1})\frac{}{\langle\, L \mid w\,;\, M_1, \ldots, M_n\,\rangle \stackrel{tar}{\Longrightarrow} \langle\, L \mid w'\,;\, M_1', \ldots, M_n'\,\rangle}$$

where

$$w' = \begin{cases} obj(\overline{w})\, out(w(M_1)) \ldots out(w(M_n)) & , \text{ if } L = Skin \\ obj(\overline{w})\, (out(w), out)\, out(w(M_1)) \ldots out(w(M_n)) & , \text{ otherwise} \end{cases}$$

and

$$w(M_i') = obj(\overline{w(M_i')})\, in_{L(M_i)}(w), \text{ for all } i \in [n]$$

For each $M_1, \ldots, M_n, M_1', \ldots, M_n' \in \mathcal{M}^+(\Pi)$, and multiset w,

$$(\mathbf{C_2})\frac{M_1, \ldots, M_n \stackrel{tar}{\Longrightarrow} M_1', \ldots, M_n'}{\langle\, L \mid w\,;\, M_1, \ldots, M_n\,\rangle \stackrel{tar}{\Longrightarrow} \langle\, L \mid w''\,;\, M_1'', \ldots, M_n''\,\rangle}$$

where

$$w'' = \begin{cases} obj(\overline{w})\, out(w(M_1')) \ldots out(w(M_n')) & \text{if } L = Skin, \\ obj(\overline{w})\, (out(w), out)\, out(w(M_1')) \ldots out(w(M_n')) & \text{otherwise,} \end{cases}$$

and each M_i'' is obtained from M_i' by replacing its resources by

$$w(M_i'') = obj(\overline{w(M_i')})\, in_{L(M_i')}(w), \text{ for all } i \in [n]$$

For each multiset w such that $here(w)\,out(w) \neq empty$,

$$(\mathbf{C_3})\frac{}{\langle\, Skin \mid w \,\rangle \stackrel{tar}{\Longrightarrow} \langle\, Skin \mid obj(\overline{w}) \,\rangle}$$

For each $M, M' \in \mathcal{M}(\Pi)$, and tar-irreducible $M_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{C_4})\frac{M \stackrel{tar}{\Longrightarrow} M'}{M, M_+ \stackrel{tar}{\Longrightarrow} M', M_+}$$

For each $M \in \mathcal{M}(\Pi)$, $M_+ \in \mathcal{M}^+(\Pi)$,

$$(\mathbf{C_5})\frac{M \stackrel{tar}{\Longrightarrow} M', M_+ \stackrel{tar}{\Longrightarrow} M'_+}{M, M_+ \stackrel{tar}{\Longrightarrow} M', M'_+}$$

Example 3 *Considering the P system of Example 1, the inference tree for* $\langle 1|empty; \langle 2|(aa, here)(ee, in_4)\delta; \langle 4|ee\rangle\rangle\rangle \stackrel{tar}{\Longrightarrow} \langle 1|empty; \langle 2|aa\delta; \langle 4|eeee\rangle\rangle\rangle$ *is:*

$$(C_2)\,\frac{(C_1)\,\dfrac{}{\langle\, 2 \mid (aa, here)(ee, in_4)\delta\,;\, \langle\, 4 \mid ee \,\rangle \,\rangle \stackrel{tar}{\Longrightarrow} \langle\, 2 \mid aa\delta\,;\, \langle\, 4 \mid eeee \,\rangle \,\rangle}}{\langle\, 1 \mid empty\,;\, \langle\, 2 \mid (aa, here)(ee, in_4)\delta\,;\, \langle\, 4 \mid ee \,\rangle \,\rangle \,\rangle \stackrel{tar}{\Longrightarrow} \langle\, 1 \mid empty\,;\, \langle\, 2 \mid aa\delta\,;\, \langle\, 4 \mid eeee \,\rangle \,\rangle \,\rangle}$$

Lemma 3 *If* $M_+ \stackrel{tar}{\Longrightarrow} M'_+$*, then* M'_+ *is tar-irreducible.*

Proof. Let M_+, M'_+ be two non-empty sets of membranes such that $M_+ \stackrel{tar}{\Longrightarrow} M'_+$. We prove that M'_+ is tar-irreducible by induction on the depth of the associated inference tree. We consider all possible cases for the final step of the inference tree, i.e., each of the five rules for communication:

(i) $M_+ \stackrel{tar}{\Longrightarrow} M'_+$ is inferred by (C_1). Then $M_+ = \langle\, L \mid w\,;\, M_1, \ldots, M_n \,\rangle$, $M'_+ = \langle\, L \mid w'\,;\, M'_1, \ldots, M'_n \,\rangle$ where $M_1, \ldots, M_n$ is a tar-irreducible set of membranes, and ($here(w) \neq empty$, or $L = Skin \wedge out(w) \neq empty$, or it exists $i \in [n]$ with $in_{L(M_i)}(w)out(w(M_i)) \neq empty$ or $here(w(M_i)) \neq empty$). Then M'_+ is tar-irreducible by Definition 3.3.

(ii) $M_+ \stackrel{tar}{\Longrightarrow} M'_+$ is inferred by (C_2). Then $M_+ = \langle\, L \mid w\,;\, M_1, \ldots, M_n \,\rangle$, $M'_+ = \langle\, L \mid w''\,;\, M''_1, \ldots, M''_n \,\rangle$ and the hypothesis $M_1, \ldots, M_n \stackrel{tar}{\Longrightarrow} M'_1, \ldots, M'_n$ is inferred by a shorter inference tree. $M'_1, \ldots, M'_n$ is tar-irreducible by inductive hypothesis. For the membrane M'_+, w'' is *out*-free if $L = Skin$, w'' is $in_{L(M_i)}$-free and $w(M''_i)$ is *out*-free, for all $i \in [n]$. Moreover, the sibling membranes $M''_1, \ldots, M''_n$ are tar-irreducible. Then M'_+ is tar-irreducible by Definition 3.3.

(iii) $M_+ \stackrel{tar}{\Longrightarrow} M'_+$ is inferred by (C_3). Then $M_+ = \langle\, Skin \mid w \,\rangle$, $M'_+ = \langle\, Skin \,|\, obj(\overline{w}) \,\rangle$, where $out(w) \neq empty$, and $w(M'_+)$ is *out*-free. Then M'_+ is tar-irreducible by Definition 3.1.

(iv) $M_+ \stackrel{tar}{\Longrightarrow} M'_+$ is inferred by (C_4). Then $M_+ = M, N_+$, $M'_+ = M', N_+$ where N_+ is tar-irreducible, and $M \stackrel{tar}{\Longrightarrow} M'$ is inferred by a shorter inference tree. It follows that M' is tar-irreducible by inductive hypothesis. Therefore M'_+ is tar-irreducible by Definition 3.2.

(v) $M_+ \stackrel{tar}{\Longrightarrow} M'_+$ is inferred by (C_5). Then $M_+ = M, N_+$, $M'_+ = M', N'_+$ where $M \stackrel{tar}{\Longrightarrow} M'$ and $N_+ \stackrel{tar}{\Longrightarrow} N'_+$ are inferred by shorter inference trees. M' and N'_+ are tar-irreducible by inductive hypothesis. It follows that M', N'_+ is a tar-irreducible by Definition 3.2.

□

Theorem 2 *Let* Π *be a P system. If* $C \in \mathcal{C}^{\#}(\Pi)$ *and* $C \stackrel{tar}{\Longrightarrow} C'$*, then* $C' \in \mathcal{C}(\Pi) \cup \mathcal{C}^{\#}(\Pi)$*, and* C' *is a tar-irreducible.*

Proof follows easily from Lemma 3.

3.3 Parallel Membrane Dissolving

If the special symbol δ occurs in the multiset of objects of a membrane labelled by L, the membrane is dissolved producing the following changes in the system:

- its evolution rules and the associated priority relation are lost, and
- its contents (objects and membranes) are added to the contents of the region which was immediately external to the dissolved membrane.

We consider the extension of the operator w (previously defined over membranes) to non-empty sets of sibling membranes by setting $w(M_1, \ldots, M_n) = w(M_1) \ldots w(M_n)$. We say that a multiset w is *δ-free* if it does not contain the special symbol δ.

Definition 4 *The* **δ-irreducibility** *property for membranes and for sets of sibling membranes is defined as follows:*

1. *a simple membrane is* **δ-irreducible***;*
2. *a non-empty set of sibling membranes $M_1, \ldots, M_n$ is* **δ-irreducible** *iff every membrane M_i is δ-irreducible, for $1 \leq i \leq n$;*
3. *a composite membrane $\langle\, L \mid w\, ;\, M_+ \,\rangle$ is* **δ-irreducible** *iff M_+ is δ-irreducible, and $w(M_+)$ is δ-free.*

The **parallel dissolving relation** $\stackrel{\delta}{\Longrightarrow}$ is defined by the following inference rules:

For each two multisets of objects w_1, w_2, and labels L_1, L_2,

$$(\mathbf{D_1})\frac{}{\langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w_2\delta \,\rangle \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1 w_2 \,\rangle}$$

For each $M_+ \in \mathcal{M}^+(\Pi)$, δ-irreducible $\langle\, L_2 \mid w_2\delta\, ;\, M_+ \,\rangle$, and label L_1,

$$(\mathbf{D_2})\frac{}{\langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w_2\delta\, ;\, M_+ \,\rangle \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1 w_2\, ;\, M_+ \,\rangle}$$

For each $M_+ \in \mathcal{M}^+(\Pi)$, δ-free multiset w_2, and labels L_1, L_2,

$$(\mathbf{D_3})\frac{\langle\, L_2 \mid w_2\, ;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_2 \mid w_2' \,\rangle}{\langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w_2\, ;\, M_+ \,\rangle \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w_2' \,\rangle \,\rangle}$$

For each $M_+, M_+' \in \mathcal{M}^+(\Pi)$, δ-free multiset w_2, multisets w_1, w_2', and labels L_1, L_2

$$(\mathbf{D_4})\frac{\langle\, L_2 \mid w_2\, ;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_2 \mid w_2'\, ;\, M_+' \,\rangle}{\langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w_2\, ;\, M_+ \,\rangle \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1\, ;\, \langle\, L_2 \mid w_2'\, ;\, M_+' \,\rangle \,\rangle}$$

For each $M_+ \in \mathcal{M}^+(\Pi)$, multisets w_1, w_2, w_2', and labels L_1, L_2

$$(\mathbf{D_5})\frac{\langle\, L_2 \mid w_2\delta\,;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_2 \mid w_2'\delta \,\rangle}{\langle\, L_1 \mid w_1\,;\, \langle\, L_2 \mid w_2\delta\,;\, M_+ \,\rangle \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1 w_2' \,\rangle}$$

For each $M_+ \in \mathcal{M}^+(\Pi)$, multisets w_1, w_2, w_2', and labels L_1, L_2

$$(\mathbf{D_6})\frac{\langle\, L_2 \mid w_2\delta\,;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_2 \mid w_2'\delta\,;\, M_+' \,\rangle}{\langle\, L_1 \mid w_1\,;\, \langle\, L_2 \mid w_2\delta\,;\, M_+ \,\rangle \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L_1 \mid w_1 w_2'\,;\, M_+' \,\rangle}$$

For each $M_+, N_+ \in \mathcal{M}^+(\Pi)$, δ-irreducible $\langle\, L \mid w\,;\, N_+ \,\rangle$, and multisets w',

$$(\mathbf{D_7})\frac{\langle\, L \mid w\,;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid w' \,\rangle}{\langle\, L \mid w\,;\, M_+, N_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid w'\,;\, N_+ \,\rangle}$$

For each $M_+, M_+', N_+' \in \mathcal{M}^+(\Pi)$, δ-irreducible $\langle\, L \mid w\,;\, N_+ \,\rangle$, and multisets w', w'',

$$(\mathbf{D_8})\frac{\langle\, L \mid w\,;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid w'\,;\, M_+' \,\rangle}{\langle\, L \mid w\,;\, M_+, N_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid w'\,;\, M_+', N_+ \,\rangle}$$

$$(\mathbf{D_9})\frac{\langle\, L \mid w\,;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww' \,\rangle \qquad \langle\, L \mid w\,;\, N_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww'' \,\rangle}{\langle\, L \mid w\,;\, M_+, N_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww'w'' \,\rangle}$$

$$(\mathbf{D_{10}})\frac{\langle\, L \mid w\,;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww' \,\rangle \qquad \langle\, L \mid w\,;\, N_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww''\,;\, N_+' \,\rangle}{\langle\, L \mid w\,;\, M_+, N_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww'w''\,;\, N_+' \,\rangle}$$

$$(\mathbf{D_{11}})\frac{\langle\, L \mid w\,;\, M_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww'\,;\, M_+' \,\rangle \qquad \langle\, L \mid w\,;\, N_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww''\,;\, N_+' \,\rangle}{\langle\, L \mid w\,;\, M_+, N_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww'w''\,;\, M_+', N_+' \,\rangle}$$

Example 4 *Considering the P system of Example 1, the inference tree for*

$\langle 1|empty; \langle 2|empty; \langle 3|dfffff\delta\rangle, \langle 4|empty\rangle\rangle\rangle \stackrel{\delta}{\Longrightarrow} \langle 1|empty; \langle 2|dfffff; \langle 4|empty\rangle\rangle\rangle$

is:

$$\dfrac{(D_4)\ \dfrac{(D_7)\ \dfrac{(D_1)\ \dfrac{}{\langle\, 2 \mid \mathit{empty}\,;\, \langle\, 3 \mid \mathit{dfffff}\delta\,\rangle\,\rangle \overset{\delta}{\Longrightarrow} \langle\, 2 \mid \mathit{dfffff}\,\rangle}}{\langle\, 2 \mid \mathit{empty}\,;\, \langle\, 3 \mid \mathit{dfffff}\delta\,\rangle, \langle\, 4 \mid \mathit{empty}\,\rangle\,\rangle \overset{\delta}{\Longrightarrow} \langle\, 2 \mid \mathit{dfffff}\,;\, \langle\, 4 \mid \mathit{empty}\,\rangle\,\rangle}}{\langle\, 1 \mid \mathit{empty}\,;\, \langle\, 2 \mid \mathit{empty}\,;\, \langle\, 3 \mid \mathit{dfffff}\delta\,\rangle, \langle\, 4 \mid \mathit{empty}\,\rangle\,\rangle\,\rangle \overset{\delta}{\Longrightarrow} \langle\, 1 \mid \mathit{empty}\,;\, \langle\, 2 \mid \mathit{dfffff}\,;\, \langle\, 4 \mid \mathit{empty}\,\rangle\,\rangle\,\rangle}$$

Lemma 4 *If $M_+ \overset{\delta}{\Longrightarrow} M'_+$, then M'_+ is δ-irreducible.*

Proof. Let M_+, M'_+ be two non-empty sets of membranes such that $M_+ \overset{\delta}{\Longrightarrow} M'_+$. We prove that M'_+ is δ-irreducible by induction on the depth of the associated inference tree. We consider all possible cases for the final step of the inference:

(i) $M_+ \overset{\delta}{\Longrightarrow} M'_+$ is inferred by (D_1). Since M'_+ is a simple membrane, M'_+ is δ-irreducible by Definition 4.1.

(ii) $M_+ \overset{\delta}{\Longrightarrow} M'_+$ is inferred by (D_2). Then $M_+ = \langle L_1 \,|\, w_1 \,;\, \langle L_2 \,|\, w_2\delta \,;\, N_+ \rangle\rangle$ and $M'_+ = \langle\, L_1 \mid w_1 w_2 \,;\, N_+ \,\rangle$, where $\langle\, L_2 \mid w_2\delta \,;\, N_+ \,\rangle$ is δ-irreducible. It follows that $w(N_+)$ is δ-free, and M'_+ is δ-irreducible by Definition 4.3.

(iii) $M_+ \overset{\delta}{\Longrightarrow} M'_+$ is inferred by (D_3). Then $M_+ = \langle L_1 \,|\, w_1 \,;\, \langle L_2 \,|\, w_2 \,;\, N_+ \rangle\rangle$ and $M'_+ = \langle\, L_1 \mid w_1 \,;\, \langle\, L_2 \mid w'_2 \,\rangle\,\rangle$, where $\langle\, L_2 \mid w_2 \,;\, N_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L_2 \mid w'_2 \,\rangle$ is inferred by a shorter inference tree, and w_2 is δ-free. $\langle\, L_2 \mid w'_2 \,\rangle$ is δ-irreducible by inductive hypothesis. Since w_2 is δ-free and no dissolving rule preserves δ, it follows w'_2 is δ-free. Then M'_+ is δ-irreducible by Definition 4.3.

(iv) $M_+ \overset{\delta}{\Longrightarrow} M'_+$ is inferred by (D_4). We proceed in a similar way to that used for (D_3).

(v) $M_+ \overset{\delta}{\Longrightarrow} M'_+$ is inferred by (D_5). M'_+ is δ-irreducible by Definition 4.1.

(vi) $M_+ \overset{\delta}{\Longrightarrow} M'_+$ is inferred by (D_6). Then $M_+ = \langle L_1 \,|\, w_1 \,;\, \langle L_2 \,|\, w_2\delta \,;\, N_+ \rangle\rangle$ and $M'_+ = \langle L_1 \,|\, w_1 w'_2 \,;\, N'_+ \rangle$, where $\langle L_2 \,|\, w_2\delta \,;\, N_+ \rangle \overset{\delta}{\Longrightarrow} \langle L_2 \,|\, w'_2\delta \,;\, N'_+ \rangle$ is inferred by a shorter inference tree. $\langle L_2 \,|\, w'_2\delta \,;\, N'_+ \rangle$ is δ-irreducible by inductive hypothesis, and hence N'_+ is δ-irreducible, and $w(N'_+)$ is δ-free. M'_+ is also δ-irreducible by Definition 4.3.

(vii) $M_+ \overset{\delta}{\Longrightarrow} M'_+$ is inferred by (D_7). Then $M_+ = \langle\, L \mid w \,;\, N'_+, N''_+ \,\rangle$ and $M'_+ = \langle\, L \mid w' \,;\, N''_+ \,\rangle$, where $\langle\, L \mid w \,;\, N'_+ \,\rangle \overset{\delta}{\Longrightarrow} \langle\, L \mid w' \,\rangle$ is inferred by

a shorter inference tree, and $\langle\, L \mid w\,;\, N''_+ \,\rangle$ is δ-irreducible. It follows that $w(N''_+)$ is δ-free, and M'_+ is δ-irreducible by Definition 4.3.

(viii) $M_+ \stackrel{\delta}{\Longrightarrow} M'_+$ is inferred by (D_8). Then $M_+ = \langle\, L \mid w\,;\, N^1_+, N^2_+ \,\rangle$ and $M'_+ = \langle\, L \mid w'\,;\, N^3_+, N^2_+ \,\rangle$, where $\langle\, L \mid w\,;\, N^1_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid w'\,;\, N^3_+ \,\rangle$ is inferred by a shorter inference tree, and $\langle\, L \mid w\,;\, N^2_+ \,\rangle$ is δ-irreducible. N^3_+ is δ-irreducible and $w(N^3_+)$ is δ-free by inductive hypothesis and Definition 4.3. It follows that N^3_+, N^2_+ is δ-irreducible by Definition 4.2. Since $w(N^3_+, N^2_+)$ is δ-free, we get that M'_+ is δ-irreducible by Definition 4.3.

(ix) $M_+ \stackrel{\delta}{\Longrightarrow} M'_+$ is inferred by (D_9). M'_+ is δ-irreducible by Definition 4.1.

(x) $M_+ \stackrel{\delta}{\Longrightarrow} M'_+$ is inferred by (D_{10}). Then $M_+ = \langle\, L \mid w\,;\, N^1_+, N^2_+ \,\rangle$ and $M'_+ = \langle\, L \mid ww'w''\,;\, N^3_+ \,\rangle$, where $\langle\, L \mid w\,;\, N^1_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww' \,\rangle$ and $\langle\, L \mid w\,;\, N^2_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww''\,;\, N^3_+ \,\rangle$ are inferred by shorter inference trees. $\langle\, L \mid ww''\,;\, N^3_+ \,\rangle$ is δ-irreducible by inductive hypothesis, and hence N^3_+ is δ-irreducible, and $w(N^3_+)$ is δ-free. Then M'_+ is δ-irreducible by Definition 4.3.

(xi) $M_+ \stackrel{\delta}{\Longrightarrow} M'_+$ is inferred by (D_{11}). Then $M_+ = \langle\, L \mid w\,;\, N^1_+, N^2_+ \,\rangle$ and $M'_+ = \langle L \mid ww'w''\,;\, N^3_+, N^4_+ \rangle$, where $\langle L \mid w\,;\, N^1_+ \rangle \stackrel{\delta}{\Longrightarrow} \langle L \mid ww'\,;\, N^3_+ \rangle$ and $\langle\, L \mid w\,;\, N^2_+ \,\rangle \stackrel{\delta}{\Longrightarrow} \langle\, L \mid ww''\,;\, N^4_+ \,\rangle$ are inferred by shorter inference trees. Both N^3_+ and N^4_+ are δ-irreducible, and both $w(N^3_+)$ and $w(N^4_+)$ are δ-free by inductive hypothesis and Definition 4.3. Therefore N^3_+, N^4_+ is also δ-irreducible, and $w(N^3_+, N^4_+)$ is δ-free. It follows that M'_+ is δ-irreducible by Definition 4.3.

□

Theorem 3 *Let* Π *be a P system. If* $C \in \mathcal{C}^{\#}(\Pi)$ *is mpr- and tar-irreducible and* $C \stackrel{\delta}{\Longrightarrow} C'$*, then* $C' \in \mathcal{C}(\Pi)$*.*

The proof follows easily from Lemma 4.

Proposition 1 *Let* Π *be a P system. If* $C \stackrel{mpr}{\Longrightarrow} C'$ *and* $C' \stackrel{tar}{\Longrightarrow} C''$ *such that* $C \in \mathcal{C}(\Pi)$*,* $C' \in \mathcal{C}^{\#}(\Pi)$*, and* C'' *is* δ*-irreducible, then* $C'' \in \mathcal{C}(\Pi)$*.*

Proof. C' is mpr-irreducible by Theorem 1, and C'' is tar-irreducible by Theorem 2. Therefore C'' does not contain both messages and δ, and hence it is a committed configuration, i.e., $C'' \in \mathcal{C}(\Pi)$. □

Proposition 2 *Let* Π *be a P system. If* $C \stackrel{mpr}{\Longrightarrow} C'$ *and* $C' \stackrel{\delta}{\Longrightarrow} C''$ *such that* $C \in \mathcal{C}(\Pi)$, $C' \in \mathcal{C}^{\#}(\Pi)$, *and* C' *is tar-irreducible, then* $C'' \in \mathcal{C}(\Pi)$.

Proof. C' is mpr-irreducible by Theorem 1, and C'' is δ-irreducible by Theorem 3. Therefore C'' does not contain both messages and δ, and hence it is a committed configuration, i.e., $C'' \in \mathcal{C}(\Pi)$. □

Proposition 3 *Let* Π *be a P system. If* $C \stackrel{mpr}{\Longrightarrow} C'$, $C' \stackrel{tar}{\Longrightarrow} C''$, *and* $C'' \stackrel{\delta}{\Longrightarrow} C'''$ *such that* $C \in \mathcal{C}(\Pi)$, *and* $C', C'' \in \mathcal{C}^{\#}(\Pi)$, *then* $C''' \in \mathcal{C}(\Pi)$.

Proof. C' is mpr-irreducible by Theorem 1, C'' is tar-irreducible by Theorem 2, and hence C'' does not contain messages. C''' is δ-irreducible by Theorem 3, and hence it does not contain both messages and δ. Therefore C''' is a committed configuration, i.e., $C''' \in \mathcal{C}(\Pi)$. □

Definition 5 *Let* Π *be a P system. A* transition step *in* Π *is defined by the following inference rules:*

For each $C, C'' \in \mathcal{C}(\Pi)$, *and* δ*-irreducible* $C'' \in \mathcal{C}^{\#}(\Pi)$,

$$\frac{C \stackrel{mpr}{\Longrightarrow} C', C' \stackrel{tar}{\Longrightarrow} C''}{C \Rightarrow C''}$$

For each $C, C'' \in \mathcal{C}(\Pi)$, *and tar-irreducible* $C' \in \mathcal{C}^{\#}(\Pi)$,

$$\frac{C \stackrel{mpr}{\Longrightarrow} C', C' \stackrel{\delta}{\Longrightarrow} C''}{C \Rightarrow C''}$$

For each $C, C''' \in \mathcal{C}(\Pi)$, *and* $C', C'' \in \mathcal{C}^{\#}(\Pi)$,

$$\frac{C \stackrel{mpr}{\Longrightarrow} C', C' \stackrel{tar}{\Longrightarrow} C'', C'' \stackrel{\delta}{\Longrightarrow} C'''}{C \Rightarrow C'''}$$

The consistency of this definition follows from the previous three propositions.

A sequence of transition steps represents a *computation*. A computation is successful if this sequence is finite, namely there is no rule applicable to the objects present in the last committed configuration. In a halting committed configuration, the result of a successful computation is the total number of objects present either in the membrane considered as the output membrane, or in the outer region.

3.4 Bisimulation

Operational semantics provides us with a formal and mechanizable way to find out which transitions are possible for the current configurations of a P system. It provides an abstract interpreter for P systems, as well as the basis for the definition of certain equivalences and congruences between P systems. Moreover, given an operational semantics, we can reason about the rules defining the semantics.

Operational semantics allows a formal analysis of membrane computing, permitting the study of relations between systems. Important relations include simulation preorders and bisimulation. These are especially useful in our context of P systems, allowing to compare two P systems.

A simulation preorder is a relation between transition systems associated to P systems expressing that the second one can match the transitions of the first one. We present a simulation as a relation over the states in a single transition system rather than between the configurations of two systems. Often a transition system consists intuitively of two or more distinct systems, but we also need our notion of simulation over the same transition system. Therefore our definitions relate configurations within one transition system, and this is easily adapted to relate two separate transition systems by building a single transition system consisting of their disjoint union.

Definition 6 *Let* Π *be a P system.*

1. *A* simulation *relation is a binary relation* R *over* $\mathcal{C}(\Pi)$ *such that for every pair of configurations* $C_1, C_2 \in \mathcal{C}(\Pi)$*, if* $(C_1, C_2) \in R$*, then for all* $C_1' \in \mathcal{C}(\Pi), C_1 \Rightarrow C_1'$ *implies that there is a* $C_2' \in \mathcal{C}(\Pi)$ *such that* $C_2 \Rightarrow C_2'$ *and* $(C_1', C_2') \in R$*.*

2. *Given two configurations* $C, C' \in \mathcal{C}(\Pi)$*,* C *simulates* C'*, written* $C' \leq C$*, iff there is a simulation* R *such that* $(C', C) \in R$*. In this case,* C *and* C' *are said to be similar, and* $\leq$ *is called the* similarity relation*.*

The similarity relation is a preorder. Furthermore, it is the largest simulation relation over a given transition system.

A bisimulation is an equivalence relation between transition systems associated to systems which behave in the same way, in the sense that one system simulates the other and vice-versa. Intuitively two systems are bisimilar if they match each other's transitions, and their evolutions cannot be distinguished.

Definition 7 *Let* Π *be a P system.*

1. *A* bisimulation *relation is a binary relation R over $\mathcal{C}(\Pi)$ such that both R and R^{-1} are simulation preorders.*

2. *Given two configurations $C, C' \in \mathcal{C}(\Pi)$, C is* bisimilar *to C', written $C \sim C'$, iff there is a bisimulation R such that $(C, C') \in R$. In this case, C and C' are said to be bisimilar, and $\sim$ is called the bisimilarity relation.*

The bisimilarity relation $\sim$ is an equivalence relation. Furthermore, it is the largest bisimulation relation over a given transition system.

4 Conclusion and Related Work

Structural operational semantics is an approach originally introduced by Plotkin [14] in which the operational semantics of a programming language or a computational model is specified in a logical way, independent of a machine architecture or implementation details, by means of rules that provide an inductive definition based on the elementary structures of the language or model. Within "structural operational semantics", two main approaches coexist:

- *Big-step semantics* is also called *natural semantics* by Kahn [7], Gunter [5], and Nielson and Nielson [11], and *evaluation semantics* by Hennessy [6]. In this approach, the main inductive predicate describes the overall result or value of executing a computation, ignoring the intermediate steps.

- *Small-step semantics* is also called *structural operational semantics* by Plotkin [14], and Nielson and Nielson [11], *computational semantics* by Hennessy [6], and *transition semantics* by Gunter [5]. In this approach, the main inductive predicate describes in more detail the execution of individual steps in a computation, with the overall computation roughly corresponding to the transitive closure of such small steps.

In general, the small-step style tends to require a greater number of rules that the big-step style, but this is outweighed by the fact that the small-step rules also tend to be simpler. The small-step style facilitates the description of interleaving [10].

In this paper we present an abstract syntax of the membrane systems, and we define a structural operational semantics of P systems by means of three sets of inference rules corresponding to maximal parallel rewriting,

parallel communication, and parallel dissolving. The inference rules come together with correctness results. The simulation and bisimulation relations between P systems are also defined.

The inference rules provide a big-step operational semantics due to the parallel nature of the model. As a continuation of this work, we translate this big-step operational semantics of P systems into rewriting logic [8], and so we get a small-step operational description. Moreover, by using an efficient implementation of rewriting logic as Maude [4], we obtain an interpreter for membrane systems, and we can verify various properties of these systems by means of a `search` command (a semi-decision procedure for finding failures of safety properties), and a LTL model checker. These achievements are presented in [2].

References

[1] O. Andrei, G. Ciobanu, D. Lucanu: Executable Specifications of the P Systems. In: Membrane Computing WMC5. *Lecture Notes in Computer Science* **3365**, Springer (2005), 127–146.

[2] O. Andrei, G. Ciobanu, D. Lucanu: Operational Semantics and Rewriting Logic in Membrane Computing. *Proceedings SOS Workshop ICALP* (2005). To appear in *ENTCS*.

[3] G. Ciobanu: Distributed Algorithms over Communicating Membrane Systems. *Biosystems* **70** (2003), Elsevier, 123–133.

[4] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, J.F. Quesada: Maude: Specification and Programming in Rewriting Logic. *Theoretical Computer Science* **285** (2002), 187–243.

[5] C. Gunter: Forms of semantic specification. *Bulletin of the EATCS* **45** (1991), 98–113.

[6] M. Hennessy: *The Semantics of Programming Languages: An Elementary Introduction Using Structural Operational Semantics.* Wiley (1990).

[7] G. Kahn: Natural semantics. *Technical Report 601*, INRIA Sophia Antipolis (1987).

[8] N. Marti-Oliet, J. Meseguer: Rewriting logic as a logical and semantical framework. In: *Handbook of Philosophical Logic.* 2nd Ed., Kluwer Academic (2002), 1–87.

[9] R. Milner: Operational and algebraic semantics of concurrent processes. In: J. van Leeuwen (Ed.): *Handbook of Theoretical Computer Science*. Vol.B, Elsevier Science (1990), 1201–1242.

[10] P. Mosses: Modular Structural Operational Semantics. *BRICS RS* 05-7 (2005).

[11] H.R. Nielson, F. Nielson: *Semantics with Applications: A Formal Introduction*. Wiley (1992).

[12] Gh. Păun: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin (2002).

[13] A. Pitts: *Semantics of programming languages*. Lecture Notes, University of Cambridge (1989).

[14] G. Plotkin: Structural operational semantics. *Journal of Logic and Algebraic Programming* **60** (2004), 17–139.

Some Recent Results Concerning Deterministic P Systems*

Oscar H. IBARRA

Department of Computer Science
University of California
Santa Barbara, California 93106, USA
E-mail: ibarra@cs.ucsb.edu

We consider P systems that are used as acceptors. In the standard semantics of P systems, each evolution step is a result of applying the rules in a maximally parallel manner – at each step, a maximal multiset of rules are nondeterministically selected and applied in parallel to the current configuration to derive the next configuration (thus, the next configuration is not unique, in general). The system is *deterministic* if at each step, there is a *unique* maximally parallel multiset of rules applicable. The question of whether or not the deterministic version is weaker than the nondeterministic version for various models of P systems is an interesting and fundamental research issue in membrane computing. Here, we look at three popular models of P systems: catalytic systems, symport/antiport systems, and communicating P systems. We report on recent results that answer some open problems in the field. These include the following:

- The membership problem for deterministic multi-membrane catalytic systems is decidable. (These systems have rules of the forms $Ca \rightarrow Cv$ or $a \rightarrow v$, where C is a catalyst, a is a noncatalyst, and v is a possibly null string of noncatalysts.) This is in contrast to the known result that nondeterministic 1-membrane catalytic systems are universal, even when the rules are restricted to the form $Ca \rightarrow Cv$. This result also gives the first example of a P system where the nondeterministic version is universal but the deterministic version is not.

*This research was supported in part by NSF Grants CCR-0208595 and CCF-0430945.

- For deterministic catalytic systems that allow rules to be prioritized, the answer to the question of whether or not they are universal depends on how priority in the application of the rules is interpreted: some are universal, others are not.

- For symport/antiport systems whose rules are of the form $(u, out; v, in)$, where u, v are multisets of objects with the restriction that $|u| \geq |v|$, the deterministic and nondeterministic versions are equivalent if and only if deterministic and nondeterministic linear-bounded automata are equivalent, the latter being a long-standing open problem in complexity theory This is in contrast to the fact that deterministic and nondeterministic 1-membrane unrestricted symport/antiport systems are equivalent and are universal.

- For a restricted model of a 1-membrane symport/antiport system that has an attached one-way input tape (containing the input string), the deterministic version is strictly weaker than the nondeterministic version. For example, the language $L = \{x\#y \mid x, y \in \{0,1\}^*, x \neq y\}$ can be accepted by a nondeterministic system but not by any deterministic system.

- The last two items above hold for similarly restricted classes of communicating P systems.

Membrane Algorithm: An Approximate Algorithm for NP-Complete Optimization Problems Exploiting P-Systems

Taishin Y. NISHIDA

Faculty of Engineering
Toyama Prefectural University
Kosugi-machi, 939-0398 Toyama, Japan
E-mail: nishida@pu-toyama.ac.jp

Abstract

A new approximate algorithm for optimization problems, called membrane algorithm, is proposed, which is an application of G. Păun's membrane computing or P-system. Membrane algorithm consists of several membrane separated regions and a subalgorithm and a few tentative solutions of the optimization problem to be solved in every region. Subalgorithms improve tentative solutions simultaneously. Then the best and worst solutions in a region are sent to adjacent inner and outer regions, respectively. By repeating this process, a good solution will appear in the innermost region. The algorithm terminates if a terminate condition is satisfied. A simple terminate condition is the number of iterations, while a little sophisticated condition becomes true if the good solution is not changed during a predetermined period. Computer experiments show that the algorithm solves the traveling salesman problem very well.

1 Introduction

Studies on approximate algorithms for NP-complete problems [1, 2, 10] are a very important issue in computer science because:

- There are thousands of NP-complete problems.
- Almost all NP-complete problems have practical importance.

- There are very few (I think no) expectations for P = NP, or strictly solving NP-complete problems in deterministic polynomial time.

We have suggested a new approximate algorithm for solving NP complete optimization problems [5, 6, 7]. The algorithm uses P-system paradigm [8]. Then it is called *membrane algorithm*. Membrane algorithm borrows nested membrane structures, rules in membrane separated regions, transporting mechanisms through membranes, and dynamic structures in rules and membranes from P-systems. Membrane algorithm remakes these components to solve NP-complete optimization problems approximately.

In the next section, basic idea and framework of membrane algorithm are explained. Then two types of membrane algorithms, which solve the traveling salesman problems approximately, are defined and their computer experiments are shown in Section 3. In Section 4, we propose an adaptive membrane algorithm which may overcome "No Free Lunch Theorem" [11].

2 Basic idea and framework of membrane algorithm

Here we explain the new algorithm, called *membrane algorithm* or MA for short.

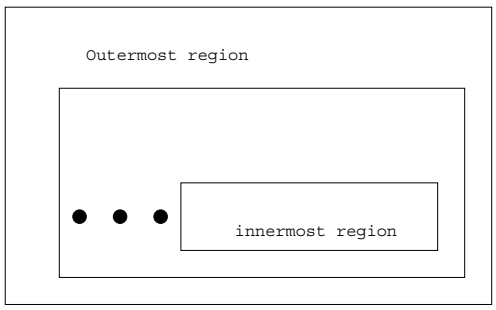


Figure 1: Membrane structure of membrane algorithm.

An MA consists of three different kinds of components:

1. A number of regions which are separated by nested membranes (Figure. 1).

2. For every region, a subalgorithm and a few tentative solutions of the optimization problem to be solved.

3. Solution transporting mechanisms between adjacent regions.

After initial settings, an MA works as follows:

1. For every region, the solutions are updated by the subalgorithm at the region, simultaneously.

2. In every region, the best and worst solutions, with respect to the optimization, are sent to the adjacent inner and outer regions, respectively.

3. MA repeats updating and transporting solutions until a terminate condition is satisfied. A simple terminate condition is the number of iterations, while a little sophisticated condition becomes true if the good solution is not changed during a predetermined period.

The best solution in the innermost region is the output of the algorithm.

An MA can have a number of subalgorithms which are any approximate algorithm for optimization problems, for example, genetic algorithm, tabu search, simulated annealing, local search, and so forth. An MA is expected to be able to escape from local minima by using a subalgorithm which likes random search at outer regions. On the other hand, it can improve good solutions in the inner regions by a subalgorithm which likes local search. So, assigning appropriate subalgorithms for a given problem, performance of MA will be excellent.

Because the subalgorithms are separated by membranes and communications occur only between adjacent regions, MA will be easily implemented in parallel, distributed, or grid computing systems. This is the second superior point of the algorithm.

3 Computer Experiments of Membrane Algorithm Solving Traveling Salesman Problem

In this section we fix components of membrane algorithm to solve the traveling salesman problem (TSP for short). Then we implement and experiment the algorithm on a computer.

3.1 Traveling Salesman Problem

An instance of TSP with n nodes contains n pairs of real numbers (x_i, y_i) $(i = 0, 1, \ldots, n-1)$ which correspond to points in the two dimensional space. The distance between two nodes $v_i = (x_i, y_i)$ and $v_j = (x_j, y_j)$ is the geometrical distance $d(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. A solution (or a tour) is a list of nodes $(v_0, v_1, \ldots, v_{n-1})$ in which no nodes appear twice, i.e., $\forall i, j\ i \neq j$ implies $v_i \neq v_j$. The *value* of a solution $v = (v_0, v_1, \ldots, v_{n-1})$, denoted by $W(v)$, is given by

$$W(v) = \sum_{i=0}^{n-2} d(v_i, v_{i+1}) + d(v_{n-1}, v_0).$$

For two solutions u and v, v is better than u if $W(v) < W(u)$. The solution which has the minimum value in all possible solutions is said to be the *strict solution* of the instance. A solution which has a value close to the strict solution is called an approximate solution.

3.2 Simple Membrane Algorithm

First we examine a simple realization of MA, called simple MA.

Let m be the number of membranes and let region 0 be the innermost and region $m-1$ be the outermost regions, respectively.

Simple MA has one tentative solution in region 0 and two solutions in regions 1 to $m-1$.

We use a tabu search as the subalgorithm in the innermost region, region 0. Tabu search searches a neighbour of the tentative solution by exchanging two nodes in the solution. In order for the same solution to avoid appearing twice, tabu search has a tabulist which consists of nodes already exchanged. Nodes in the tabulist are not exchanged again. Tabu search resets the tentative solution and the tabulist if one of the three conditions occurs:

1. The value of the neighbouring solution is less than that of the tentative solution. The neighbouring solution becomes the tentative solution.

2. The value of the best solution in the region 1 is less than that of the tentative solution. The best solution in the region 1 becomes the new tentative solution.

3. Neighbour search exceeds a predetermined turns (in this case $\frac{n}{5}$). The tentative solution remains. Only tabulist is reset.

In case 3, no improvement occurs. But tabu search tries to search other neighbours, since there are many unsearched neighbours.

The tentative solutions in regions 1 to $m-1$ (there are two solutions in each region) are improved by a subalgorithm summarized below:

1. If the two solutions have the same value, then a part of one solution (which is selected probabilisticly) is reversed.
2. Recombinates the two solutions and makes two new solutions.
3. Modifies the two new solutions by point mutations. In the i-th region, a mutation occurs under probability $\frac{i}{m}$.

Obviously the subalgorithm described above resembles genetic algorithms. But the subalgorithm always recombinates the two solutions in a region while genetic algorithms randomly select solutions to be recombined. If the two solutions in a region are identical, recombination makes no new solutions. The step 1 avoids this case and introduces a new solution using reverse operation, which is a kind of mutation.

The overall algorithm looks like:

1. Given an instance of TSP.
2. Randomly makes one tentative solution for region 0 and two tentative solutions for every region 1 to $m-1$.
3. Repeats 3.1 to 3.3 for d times (d is given as a parameter).
   - 3.1 Modify tentative solutions simultaneously in every region using the subalgorithm at the region.
   - 3.2 For every region i ($1 \leq i \leq m-2$), sends the best solution of the solutions in the region (old solutions and modified solutions) to region $i-1$ and the worst solution to region $i+1$. (In region 0, sends the worst solution to region 1 and in region $m-1$, sends the best solution to region $m-2$.)
   - 3.3 For every region 1 to $m-1$ erases solutions but the best two.
4. Outputs the tentative solution in region 0 as the output of the algorithm.

In the above algorithm, steps 3.2 and 3.3 correspond to solution transporting mechanisms between adjacent regions.

3.3 Computer Experiments

We have implemented the algorithm using Java programming language (see Appendix A). By using Java, modifications of the algorithm have been easily tested on a computer. For example, we have implemented several recombination methods and have found that edge exchange recombination (EXX, [4] and Appendix B) exhibits the best performance.

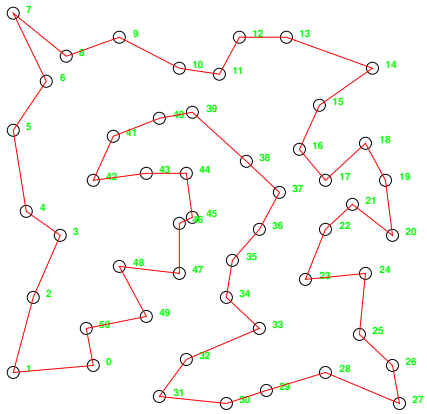


Figure 2: An example tour (solution) obtained by simple MA. The instance is eil51.

Tables 1 and 2 show results of the program for TSP benchmark problems eil51 and kroA100 from TSPLIB [9][1]. Results of simulated annealing from [12] are also shown in the tables. Table 3 shows results of the program for various benchmark problems from TSPLIB.

[1]The optimum values in TSPLIB are obtained with integer distances between nodes. The MA's in this paper are implemented using real valued distances. So we cannot directly compare the results to the optimum values in TSPLIB. To solve the problems using real valued distances is, however, as hard as using integer distances.

Table 1: Results of membrane algorithm and a simulated annealing (SA) for the benchmark problem eil51. Membrane algorithm repeats step 3 40,000 times. The number of trials of membrane algorithm is 20. Membrane 2, 10, 30, 50, and 70 stand for the algorithms with 2, 10, 30, 50, and 70 regions, respectively.

Algorithm	Membrane					SA
	2	10	30	50	70	
Best	440	437	432	429	429	430
Average	522	449	441	435	434	438
Worst	786	466	451	444	443	445

Table 2: Results for benchmark problem kroA100. 100,000 iterations and 20 trials.

Algorithm	Membrane						SA
	2	10	30	50	70	100	
Best	23564	21776	21770	21651	21544	21299	21369
Average	34601	23195	22878	22590	22275	21941	21763
Worst	82756	24862	23940	24531	23569	22954	22564

Table 3: Results of simple MA with 50 membranes and 100,000 iterations for verous benchmark problems.

Problem	nodes	Results			optimum
		best	average	worst	value
ulysses22	22	75.31	75.31	75.31	—
eil51	51	429	434	444	426
eil76	76	556	564	575	538
eil101	101	669	684	693	629
kroA100	100	21651	22590	24531	21282
ch150	150	7073	7320	7633	6528
gr202	202	509.7	520.1	528.4	—
tsp225	225	4073.1	4153.6	4238.9	—

Note: The — in the optimum column represents that optimum values in TSPLIB are calculated other metric and cannot be compared with geometrical distance used here.

3.4 Shrink Membrane Algorithm

Now we make *shrink membrane algorithm* or shrink MA by incorporating dynamic structures of P-system.

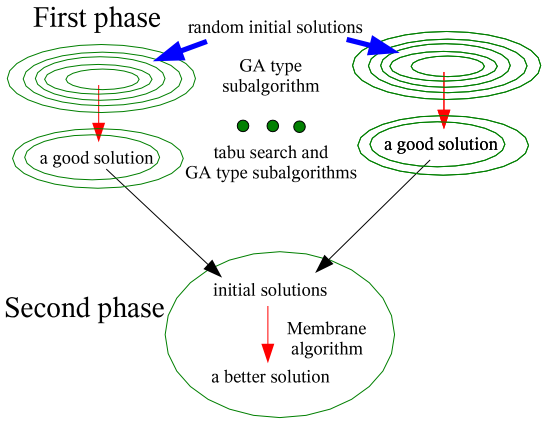


Figure 3: Shrink membrane algorithm.

Table 4: Parameters of shrink membrane algorithm used computer experiment.

Phase	number of		subalgorithms	terminate conditions
	structures	membranes		unchange during
1–1	$t = 100$	$m = 5$	GA type only	$100n$
1–2	$t = 100$	2	GA type and tabu search	$300n$
2	1	$\frac{t}{2} = 50$	GA type and tabu search	$100n$

Shrink MA consists of two phases. Its first phase starts with m membranes and GA type subalgorithms in all regions, where m is a parameter. If the best solution in region 0 does not change during $100n$ iterations (where n is the size of the instance, i.e., number of nodes), then the number of membranes becomes 2 with tabu search in region 0 and GA type subalgorithm

in region 1. The two regions have the same initial solutions which are the best solution obtained so far. Then the algorithm improves solutions until the solution in region 0 does not change during $300n$ iterations.

The first phase has a number, say t, of membrane structures. All structures do the same computation independently. They get different solutions because the subalgorithms use probabilistic choice[3].

The second phase of shrink MA has one membrane structure with $\frac{t}{2}$ regions. The t solutions obtained in the first phase are sorted and put in $\frac{t}{2}$ regions. A region gets two solutions and the inner region gets better solutions. In other words, the results of the first phase become the initial solutions of the second phase. The subalgorithms of the second phase are identical to those of simple MA but shrink MA terminates if the best solution does not change during $100n$ iterations.

Figure 3 and Table 4 illustrate flow and parameters of shrink MA. The parameters shown in Table 4 are selected by several preliminary experiments of solving eil51 and kroA100 with various combinations of parameters.

Table 5: Results of 10 trials of shrink MA for various benchmark problems. The columns "steps" stand for total counts of iterations, including both phases 1 and 2, to obtain the value. The "average steps" represents the average of steps of all 10 trials.

Problem	best		average		worst	
	value	steps	value	steps	value	steps
eil51	429	3,171,593	431	3,159,977.5	436	3,145,823
eil76	547	5,980,046	556	5,812,737.6	561	5,896,362
eil101	655	9,002,312	667	9,042,661.4	677	8,782,816
kroA100	21299	9,776,654	21504	9,673,806.1	21750	9,846,136
ch150	6751	18,365,598	6889	18,656,657.4	6961	18,821,244
gr202	506.8	30,473,165	510.4	29,910,766.6	515.4	29,963,798
tsp225	4031.4	35,420,300	4112.3	35,583,424.0	4171.6	35,912,701

Table 5 shows results of shrink MA for benchmark problems in TSPLIB. We can see that shrink MA always yields a good approximate solution while it takes quite many steps. By changing parameters in Table 4, shrink MA

[3]Of course the results of the first phase may be unique if the strict solution is obtained.

may get results in shorter steps for large TSP instances. The computations of the first phase of shrink MA are completely independent each other. So shrink MA will be easily implemented on a parallel processing system and computation time willl become much shorter.

4 An Adaptive Membrane Algorithm

There are several optimization algorithms, e.g., simulated annealing, ant colony, etc., which may give MA good results if they are used as the subalgorithms of MA. But we must remind "No Free Lunch Theorem" [3, 11], which says that every optimization algorithm has equal performance in average if the average is computed over *all* optimization problems. So we propose *adaptive membrane algorithm* or adaptive MA (Figure 4).

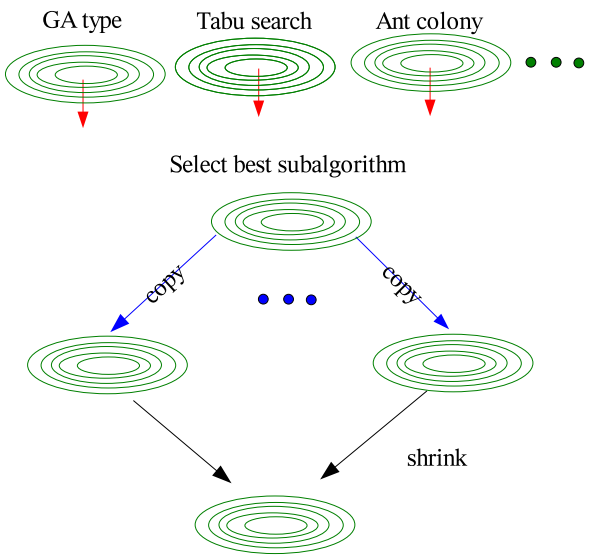


Figure 4: Outline of adaptive MA.

An adaptive MA starts with several membrane structures which have different subalgorithms each other. After a predetermined steps or real computation time, a structure which has the best solution is selected. Then the structure is copied with its subalgorithm. Finally, a shrink type process

gets the result of the algorithm. An adaptive MA will be able to overcome "No Free Lunch Theorem" and to obtain good results for any, not only TSP, NP complete optimization problems.

5 Conclusion

In this paper, notions of membrane algorithms (MA) are presented. MA is a new approximate algorithm for solving NP-complete optimization problems. Here we have defined two types of MA solving the traveling salesman problem, simple MA and shrink MA, and examined their behaviours on a computer. We have observed that simple MA gets as good approximate solutions as simulated annealing and that simple MA converges fast. On the other hand, shrink MA always obtains quite good approximate solutions.

We have proposed adaptive membrane algorithm which may solve any optimization problems well. That is, adaptive membrane algorithm has a possibility of defeating "No Free Lunch Theorem".

References

[1] C. A. Floudas, P. M. Pardalos (Eds.): *Encyclopedia of Optimization.* Kluwer, Dordrecht (2001).

[2] M. R. Garey, D. S. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness.*Freeman (1979).

[3] C. Igel, M. Toussaint: On classes of functions for which No Free Lunch results hold. *Information Processing Letters*, **86** (2003), 317–321.

[4] K. Maekawa et al.: A solution of traveling salesman problem by genetic algorithm (in Japanese). *SICE* **31** (1995), 598–605.

[5] T. Y. Nishida: An application of P-system. A new algorithm for NP-complete optimization problems. In: N. Callaos et. al.(Eds.): *Proceedings of The 8th World Multi-Conference on Systems, Cybernetics and Informatics* **Vol V** (2004), 109–112.

[6] T. Y. Nishida: An approximate algorithm for NP-complete optimization problems exploiting P-systems. In *Proceedings of Brainstorming Workshop on Uncertainty in Membrane Computing*(2004), 185–192.

[7] T. Y. Nishida: An approximate algorithm for NP-complete optimization problems exploiting P-systems. In G. Ciobanu, Gh. Păun, M. Prez-Jimnez (Eds.): *Applications of Membrane Computing.* Springer, to appear.

[8] Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (2000), 108–143.

[9] G. Reinelt: TSPLIB: `http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`

[10] A. Salomaa: *Computation and Automata.* Cambridge University Press, Cambridge (1985).

[11] D. H. Wolpert, W. G. Macready: No Free Lunch Theorem for optimization. *IEEE Transactions on Evolutionary Computation* **1** (1997), 67-82.

[12] M. Yoneda: `http://www.mikilab.doshisha.ac.jp/dia/research/person/yoneda/research/2002_7_10/SA/07-sareslut.html`

Appendices

A The Simulation Program

Although Java is slightly slower than C or other languages which are compiled into native machine codes, Java has many merits [1]:

- Because Java is an object oriented language, various types of subalgorithms are easily incorporated in a program without affecting the remaining parts of the program.

- Because Java is a multi-platform language, a program in Java runs almost all computers and operating systems including Solaris on SPARC, Solaris on x86, Linux, Windows, MacOS, and so forth.
- Developing a program in Java is, in general, more efficient than developing in any other languages. Rigorous syntax checks in compiling and detailed exception reports at run time help us to find and fix most bugs.

Table 6: Classes and their methods of the simulation program.

class	super class	descriptions
TSP	(Object)	TSP contains methods which are commonly used, e.g., read a TSP instance, write results of simulations, etc.
TSPsimple	TSP	TSPsimple has the main method of simple MA.
TSPshrink	TSP	TSPshrink has the main method of shrink MA.
Membrane	(Object)	Membrane represents a nested membrane structure. Membrane has an array of OneRegion as a field and method oneRound() which proceeds simulation one step.
OneRegion	(Object)	OneRegion is an abstract class which defines one region of membrane structure. OneRegion has two abstract methods update() and select().
RegionEXX_HC	OneRegion	RegionEXX_HC realizes edge exchange crossover subalgorithm.
RegionTabuSearch	OneRegion	RegionTabuSearch does tabu search subalgorithm.

Table 6 and Figure 5 show essential classes and methods of the simulation program. The program will be opened as a free software [2].

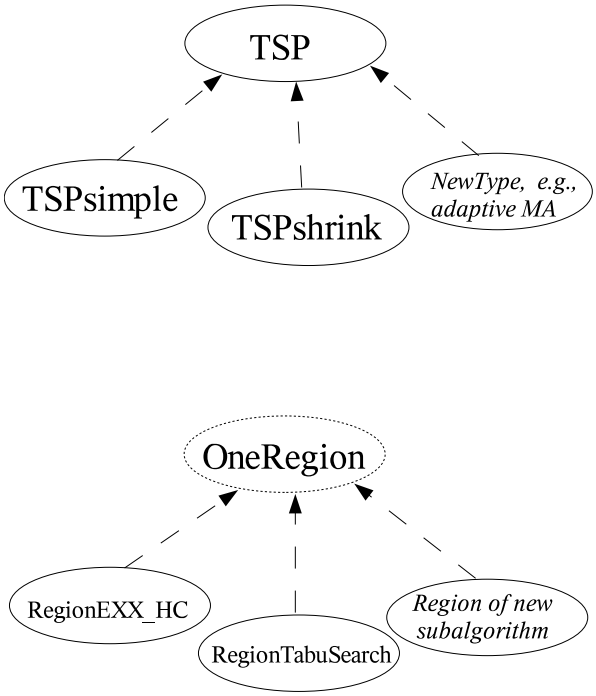


Figure 5: Class hierarchies of TSP, TSPsimple, TSPshrink, and others (above); and OneRegion, RegionEXX_HC, RegionTabuSearch, and others (below). An abstract class is represented by a doted circle.

B Edge Exchange crossover

We describe edge exchange crossover in this appendix.

For a positive integer n, let $[n]$ denote the set $\{0, 1, \ldots, n-1\}$. An instance of TSP with n nodes is a weighted complete graph of n nodes[1]. A (pseudo) tour of a TSP instance of n nodes is a function $X : [n] \to [n]$. If X is a bijection, then X is a tour of the instance; otherwise, X is a pseudo

[1]The definition here corresponds to all types of weights not only geometrical distance in the two dimensional space.

tour. For every $i \in [n]$, $X(i)$ stands for the i-th node of the (pseudo) tour X.

For a (pseudo) tour X and an integer $i \in [n]$, $E(X,i)$ stands for the i-th edge of the (pseudo) tour, i.e., $E(X,i) = (X(i), X((i+1) \bmod n))$. For an edge e of a (pseudo) tour X, $s(e)$ and $t(e)$ denote the beginning and the ending nodes of e, respectively. The following equations are obvious:

$$\begin{aligned} s(E(X,i)) &= X(i) \\ t(E(X,i)) &= X((i+1) \bmod n). \end{aligned}$$

In the sequel, all additions and subtractions are those of the modulo-n residue ring.

The edge exchange crossover (EXX for short) is an algorithm described below.

Algorithm EXX:
Input: A TSP instance and two tours A, B
Output: recombined tours A', B'

1. Select $i \in [n]$ randomly.
2. Let j be an integer such that $s(E(B,j)) = s(E(A,i))$.
3. Make new (pseudo) tours A', B' by

$$A'(k) = \begin{cases} A(k) & 0 \le k \le i, i+2 \le k \le n-1 \\ t(E(B,j)) & k = i+1 \end{cases}$$

$$B'(k) = \begin{cases} B(k) & 0 \le k \le j, j+2 \le k \le n-1 \\ t(E(A,i)) & k = j+1 \end{cases}$$

4. While $t(E(A,i)) \neq t(E(B,j))$ do
   /* If $t(E(A,i)) = t(E(B,j))$, then A' and B' are tours and the algorithm terminates. See lemma 1. */
5. Let i' and j' be integers such that $s(E(A,i')) = t(E(B,j))$ and $s(E(B,j')) = t(E(A,i))$.
6. Make (pseudo) tours A'' and B'' by
   case $i < i'$

$$A''(k) = \begin{cases} A'(k) & 0 \le k \le i+1 \text{ or } i' \le k \le n-1 \\ t(E(B,j))(= A'(i+1)) & k = i+1 \\ A(i+i'+1-k) & i+2 \le k \le i'-1 \end{cases}$$

case $i' < i$

$$A''(k) = \begin{cases} A'(k) & i' \le k \le i+1 \\ t(E(B,j))(= A'(i+1)) & k = i+1 \\ A(i+i'+1-k) & 0 \le k \le i'-1 \text{ or } i+2 \le k \le n-1 \end{cases}$$

case $j < j'$

$$B''(k) = \begin{cases} B'(k) & 0 \le k \le j+1 \text{ or } j' \le k \le n-1 \\ t(E(A,i))(= B'(j+1)) & k = j+1 \\ B(j+j'+1-k) & j+2 \le k \le j'-1 \end{cases}$$

case $j' < j$

$$B''(k) = \begin{cases} B'(k) & j' \le k \le j+1 \\ t(E(B,j))(= A'(i+1)) & k = i+1 \\ B(j+j'+1-k) & 0 \le k \le j'-1 \text{ or } j+2 \le k \le n-1 \end{cases}$$

7. Let $i \leftarrow i'$, $j \leftarrow j'$, $A \leftarrow A''$, $B \leftarrow B''$
8. Make new (pseudo) tours A', B' by

$$A'(k) = \begin{cases} A(k) & 0 \le k \le i-1 \text{ or } i+2 \le k \le n-1 \\ B(j) & k = i \\ t(E(B,j)) & k = i+1 \end{cases}$$

$$B'(k) = \begin{cases} B(k) & 0 \le k \le j-1 \text{ or } j+2 \le k \le n-1 \\ A(i) & k = j \\ t(E(A,i)) & k = j+1 \end{cases}$$

/* the end of "while" loop */

Example 1 *If tours A and B are given by the following table*

x	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
$A(x)$	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
$B(x)$	*1*	*4*	*3*	*0*	*5*	*6*	*2*	*7*

and i is selected to 1, then the algorithm EXX calculates $j = 0$, $i' = 4$, and $j' = 6$. The pseudo tours A'' and B'' are given by

x	0	1	2	3	4	5	6	7
$A''(x)$	0	1	4	3	4	5	6	7
$B''(x)$	1	2	6	5	0	3	2	7

.

Now, $i = 4$, $j = 6$, $i' = 7$, *and* $j' = 3$. *The next* A'' *and* B'' *become*

x	0	1	2	3	4	5	6	7
$A''(x)$	0	1	4	3	2	7	6	7
$B''(x)$	6	2	1	5	0	3	4	5

.

Then $i = 7$ *and* $j = 3$ *and the terminate condition of the algorithm becomes true obtaining the tours*

x	0	1	2	3	4	5	6	7
$A'(x)$	0	1	4	3	2	7	6	5
$B'(x)$	6	2	1	7	0	3	4	5

.

Lemma 1 *Let* A, B, A', B', A'', B'', i, j, i', *and* j' *be the symbols defined in the algorithm EXX. The following conditions are equivalent.*

1. A' *and* B' *are not tours.*
2. $t(E(A,i)) \neq t(E(B,j))$, $A''(i') = A''(i+1)$, $B''(j') = B''(j+1)$, $\forall x, y \in [n] - \{i', i+1\}\, x \neq y$ *implies* $A''(x) \neq A''(y)$, *and* $\forall x, y \in [n] - \{j', j+1\}\, x \neq y$ *implies* $B''(x) \neq B''(y)$.

Proof. The proof of $2 \rightarrow 1$ is obvious.

We prove the $1 \rightarrow 2$ by induction on the "while" loop.

At the first execution of the loop, A' has $t(E(B,j))(= B(j+1))$ at $A'(i+1)$ and B' has $t(E(A,i))(= A(i+1))$ at $B'(j+1)$. If $A(i+1) \neq B(j+1)$, then there exist $i', j' \in [n]$ such that $i \neq i'$, $j \neq j'$, $A(i') = A'(i') = A'(i+1)$, and $B(j') = B'(j') = B'(j+1)$ because A and B are tours. Then all assertions of 2 hold.

Next let 1 and 2 be equivalent until the previous execution of the loop. In this case we have $A'(i) = B(j)$, $A'(i+1) = t(E(B,j))$, $B'(j) = A(i)$, and $B'(j+1) = t(E(A,i))$. If $A(i+1) \neq B(j+1)$, then there exist $i', j' \in [n]$ such that $i \neq i'$, $j \neq j'$, $A(i') = A'(i') = A'(i+1)$, and $B(j') = B'(j') = B'(j+1)$ because of the hypothesis of induction. Then assertions of 2 hold. □

Lemma 2 *The algorithm EXX always terminates.*

Proof. It is easily seen that the algorithm EXX is reversible, that is, given A'', B'', i, j, i', and j', A and B are uniquely determined. The observation implies that the algorithm EXX is an injection on the direct product of the sets of pseudo tours. Then the lemma follows from Lemma 1 and the fact that the domain of the algorithm is finite. □

Lemmas 1 and 2 prove the next theorem.

Theorem 3 *The algorithm EXX always terminates and outputs recombined tours.*

References

[1] K. Arnold, J. Gosling, D. Holmes: *The Java Programming Language. Third Edition.*Addison-Wesley, Boston (2000).

[2] T. Y. Nishida: `http://www.comp.pu-toyama.ac.jp/nishida/`

Computational Power of Symport/Antiport: History, Advances and Open Problems

Artiom ALHAZOV[1,2], Rudolf FREUND[3], Yurii ROGOZHIN[2]

[1]Research Group on Mathematical Linguistics
Rovira i Virgili University, Tarragona, Spain
E-mail: artiome.alhazov@estudiants.urv.es

[2]Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
E-mail: {artiom,rogozhin}@math.md

[3]Faculty of Informatics
Vienna University of Technology, Austria
E-mail: rudi@emcc.at

Abstract

We first give a historical overview of the most important results obtained in the area of P systems and tissue P systems with *symport / antiport* rules, especially with respect to the development of computational completeness results improving descriptional complexity parameters as the number of membranes and cells, respectively, and the weight of the rules as well as the number of objects. Then we establish our newest results: P systems with only one membrane and symport rules of weight three can generate any recursively enumerable language with only seven additional objects remaining in the skin membrane at the end of a halting computation; P systems with minimal cooperation, i.e., P systems with symport / antiport rules of size one and P systems with symport rules of weight two, are computationally complete with only two membranes with only three and six, respectively, superfluous objects remaining in the output membrane at the end of a halting computation.

1 Introduction

P systems with *symport / antiport* rules, i.e., P systems with *pure communication rules assigned to membranes*, first were introduced in [27]; symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions. These operations are very powerful, i.e., P systems with symport / antiport rules have universal computational power with only one membrane, e.g., see [12], [20], [15].

After establishing the necessary definitions, we first give a historical overview of the most important results obtained in the area of P systems and tissue P systems with *symport / antiport* rules and review the development of computational completeness results improving descriptional complexity parameters, especially concerning the number of membranes and cells, respectively, and the weight of the rules as well as the number of objects. Moreover, we establish our newest results: first we prove that P systems with only one membrane and symport rules of weight three can generate any recursively enumerable language with only seven additional symbols remaining in the skin membrane at the end of a halting computation, which improves the result of [19] where thirteen superfluous symbols remained. Then we show that P systems with minimal cooperation, i.e., P systems with symport / antiport rules of weight one and P systems with symport rules of weight two, are computationally complete with only two membranes modulo some initial segment: In P systems with symport / antiport rules of weight one, only three superfluous objects remain in the output membrane at the end of a halting computation, whereas in P systems with symport rules of weight two six additional objects remain. For both variants, in [4] it has been shown that two membrane are enough to obtain computational completeness modulo a terminal alphabet; in this paper, we now show that the use of a terminal alphabet can be avoided for the price of superfluous objects remaining in the output membrane at the end of a halting computation. So far we were not able to completely avoid these additional objects, hence, it remains as an interesting question how to reduce their number.

2 Basic Notions and Definitions

For the basic elements of formal language theory needed in the following, we refer to [30]. We just list a few notions and notations: $\mathbb{N}$ denotes the set of natural numbers (i.e., of non-negative integers). V^* is the free monoid

generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; by $\mathbb{N}RE$, $\mathbb{N}REG$, and $\mathbb{N}FIN$ we denote the family of recursively enumerable sets, regular sets, and finite sets of natural numbers, respectively. For $k \geq 1$, by $\mathbb{N}_kRE$ we denote the family of recursively enumerable sets of natural numbers excluding the initial segment 0 to $k-1$. Equivalently, $\mathbb{N}_kRE = \{k + L \mid L \in \mathbb{N}RE\}$, where $k + L = \{k + n \mid n \in L\}$.

Let $\{a_1, ..., a_n\}$ be an arbitrary alphabet; the number of occurrences of a symbol a_i in x is denoted by $|x|_{a_i}$; the *Parikh vector* associated with x with respect to $a_1, ..., a_n$ is $\left(|x|_{a_1}, ..., |x|_{a_n}\right)$. The *Parikh image* of a language L over $\{a_1, ..., a_n\}$ is the set of all Parikh vectors of strings in L. A (finite) multiset $\langle m_1, a_1\rangle ... \langle m_n, a_n\rangle$ with $m_i \in \mathbb{N}$, $1 \leq i \leq n$, can be represented by any string x the Parikh vector of which with respect to $a_1, ..., a_n$ is $(m_1, ..., m_n)$.

The families of recursively enumerable sets of vectors of natural numbers are denoted by $PsRE$.

2.1 Register Machines and Counter Automata

The proofs of the main results discussed in this paper are based on the simulation of register machines or counter automata, respectively; with respect to register machines, we refer to [25] for original definitions, and to [11] for definitions like those we use in this paper:

A (non-deterministic) *register machine* is a construct

$$M = (d, Q, q_0, q_f, P)$$

where

- d is the number of registers,
- Q is a finite set of label for the instructions of M,
- q_0 is the initial label,
- q_f is the final label, and
- P is a finite set of instructions injectively labelled with elements from Q.

  The instructions are of the following forms:

  1. $q_1 : (A(r), q_2, q_3)$;
     add 1 to the contents of register r and proceed to one of the instructions (labelled with) q_2 and q_3 ("ADD"-instruction).

2. $q_1 : (S(r), q_2, q_3)$;
   if register r is not empty, then subtract 1 from its contents and go to instruction q_2, otherwise proceed to instruction q_3 ("SUBTRACT"-instruction).
3. $q_f : halt$;
   stop the machine; the final label q_f is only assigned to this instruction.

A (non-deterministic) register machine M is said to generate a vector $(s_1, \ldots, s_k)$ of natural numbers if, starting with the instruction with label q_0 and all registers containing the number 0, the machine stops (it reaches the instruction $q_f : halt$) with the first k registers containing the numbers $s_1, \ldots, s_k$ (and all other registers being empty).

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate exactly the sets of vectors of natural numbers which can be generated by Turing machines, i.e., the family $PsRE$. More precisely, from the main result in [25] that the actions of a Turing machine can be simulated by a register machine with two registers (using a prime number encoding of the configuration of the Turing machine) we know that any recursively enumerable set of k-vectors of natural numbers can be generated by a register machine with $k+2$ registers where only "ADD"-instructions are needed for the first k registers.

A non-deterministic *counter automaton* is a construct

$$M = (d, Q, q_0, q_f, P)$$

where

- d is the number of counters, and we denote $D = \{1, ..., d\}$;
- Q is a finite set of states, and without loss of generality, we use the notation $Q = \{q_i \mid 0 \leq i \leq f\}$ and $F = \{0, 1, ..., f\}$,
- $q_0 \in Q$ is the initial state,
- $q_f \in Q$ is the final state, and
- P is a finite set of instructions of the following form:

  1. $(q_i \rightarrow q_l, k+)$, with $i, l \in F$, $i \neq f$, $k \in D$ ("increment"-instruction). This instruction increments counter k by one and changes the state of the system from q_i to q_l.

2. $(q_i \rightarrow q_l, k-)$, with $i, l \in F,\ i \neq f,\ k \in D$ ("decrement"-instruction). If the value of counter k is greater than zero, then this instruction decrements it by 1 and changes the state of the system from q_i to q_l. Otherwise (when the value of register k is zero) the computation is blocked in state q_i.
3. $(q_i \rightarrow q_l, k = 0)$, with $i, l \in F,\ i \neq f,\ k \in D$ ("test for zero"-instruction). If the value of counter k is zero, then this instruction changes the state of the system from q_i to q_l. Otherwise (the value stored in counter k is greater than zero) the computation is blocked in state q_i.
4. *halt*. This instruction stops the computation of the counter automaton, and it can only be assigned to the final state q_f.

A transition of the counter automaton consists in updating/checking the value of a counter according to an instruction of one of the types described above and by changing the current state to another one. The computation starts in state q_0 with all counters being equal to zero. The result of the computation of a counter automaton is the value of the first k counters when the automaton halts in state $q_f \in Q$ (without loss of generality we may assume that in this case all other counters are empty). A counter automaton thus (by means of all computations) generates a set of k-vectors of natural numbers. As for register machines, we know that any set of k-vectors of natural numbers from $PsRE$ can be generated by a counter automaton with $k+2$ counters where only "increment"-instructions are needed for the first k counters.

A special variant of counter automata uses a set C of pairs $\{i, j\}$ with $i, j \in Q$ and $i \neq j$. As a part of the semantics of the *counter automaton with conflicting counters* $M = (d, Q, q_0, q_f, P, C)$, the automaton stops without yielding a result whenever it reaches a configuration where, for any pair of conflicting counters, both are non-empty.

Given an arbitrary counter automaton, we can easily construct an equivalent counter automaton with conflicting counters: For every counter i which shall also be tested for zero, we add a conflicting counter $\bar{\imath}$; then we replace all "test for zero"-instructions $(l \rightarrow l', i = 0)$ by the sequence of instructions $(l \rightarrow l'', \bar{\imath}+)$, $(l'' \rightarrow l', \bar{\imath}-)$. Thus, in counter automata with conflicting counters we only use "increment"-instructions and "decrement"-instructions, whereas the "test for zero"-instructions are replaced by the special conflicting counters semantics.

2.2 P Systems with Symport / Antiport Rules

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [28]; comprehensive information can be found on the P systems web page `http://psystems.disco.unimib.it`.

A *P system with symport / antiport rules* is a construct

$$\Pi = (O, \mu, w_1, \ldots, w_k, E, R_1, \ldots, R_k, i_0)$$

where:

1. O is a finite alphabet of symbols called *objects*;
2. μ is a *membrane structure* consisting of k membranes that are labelled in a one-to-one manner by $1, 2, \ldots, k$;
3. $w_i \in O^*$, for each $1 \leq i \leq k$, is a finite multiset of objects associated with the region i (delimited by membrane i);
4. $E \subseteq O$ is the set of objects that appear in the environment in an infinite number of copies;
5. R_i, for each $1 \leq i \leq k$, is a finite set of symport / antiport rules associated with membrane i; these rules are of the forms (x, in) and (y, out) *(symport rules)* and $(y, out; x, in)$ *(antiport rules)*, respectively, where $x, y \in O^+$;
6. i_0 is the label of an elementary membrane of μ that identifies the corresponding output region.

A P system with symport / antiport rules is defined as a computational device consisting of a set of k hierarchically nested membranes that identify k distinct regions (the membrane structure μ), where to each membrane i there are assigned a multiset of objects w_i and a finite set of symport / antiport rules R_i, $1 \leq i \leq k$. A rule $(x, in) \in R_i$ permits the objects specified by x to be moved into region i from the immediately outer region. Notice that for P systems with symport rules the rules in the skin membrane of the form (x, in), where $x \in E^*$, are forbidden. A rule $(x, out) \in R_i$ permits the multiset x to be moved from region i into the outer region. A rule $(y, out; x, in)$ permits the multisets y and x, which are situated in region i and the outer region of i, respectively, to be exchanged. It is clear that a rule can be applied if and only if the multisets involved by this rule are

present in the corresponding regions. The weight of a symport rule (x, in) or (x, out) is given by $|x|$, while the weight of an antiport rule $(y, out; x, in)$ is given by $max\{|x|, |y|\}$.

As usual, a computation in a P system with symport / antiport rules is obtained by applying the rules in a non-deterministic maximally parallel manner. Specifically, in this variant, a computation is restricted to moving objects through membranes, since symport / antiport rules do not allow the system to modify the objects placed inside the regions. Initially, each region i contains the corresponding finite multiset w_i, whereas the environment contains only objects from E that appear in infinitely many copies.

A computation is successful if starting from the initial configuration, the P system reaches a configuration where no rule can be applied anymore. The result of a successful computation is a natural number that is obtained by counting all objects (only the terminal objects as it done in [4], if in addition we specify a subset of O as the set of terminal symbols) present in region i_0. Given a P system Π, the set of natural numbers computed in this way by Π is denoted by $N(\Pi)$. If the multiplicity of each (terminal) object is counted separately, then a vector of natural numbers is obtained, denoted by $Ps(\Pi)$, see [28]. For short, we shall also speak of a *P system* only when dealing with a *tissue P system with symport / antiport rules* as defined above.

By

$$\mathbb{N}O_nP_m(sym_s, anti_t)$$

we denote the family of sets of natural numbers (non-negative integers) that are generated by a P system with symport / antiport rules having at most $n > 0$ objects in O, at least $m > 0$ membranes, symport rules of size at most $s \geq 0$, and antiport rules of size at most $t \geq 0$. By

$$\mathbb{N}_kO_nP_m(sym_s, anti_t)$$

we denote the corresponding families of recursively enumerable sets of natural numbers without initial segment $\{0, 1, \ldots, k-1\}$. If we replace numbers by vectors, then in the notations above $\mathbb{N}$ is replaced by Ps. When any of the parameters m, n, s, t is not bounded, it is replaced by $*$; if the number of objects n is unbounded, we also may just omit n. If $s = 0$, then we may even omit sym_s; if $t = 0$, then we may even omit $anti_t$.

It may happen that P system with symport / antiport (symport) rules can simulate deterministic register machines (i.e., register machines where in each ADD-instruction $q_1 : (A(r), q_2, q_3)$ the labels q_2 and q_3 are equal)

in a deterministic way, i.e., from each configuration of the P system we can derive at most one other configuration. Then, when considering these P systems as accepting devices (the input from a set in $PsRE$ is put as an additional multiset into some specified membrane of the P system), we can get deterministic accepting P systems; the corresponding families of recursively enumerable sets of natural numbers then are denoted in the same way as before, but with the prefix aD; e.g., from the results proved in [17] and [13] we immediately obtain

$$PsRE = aDPsOP_1(anti_2).$$

Sometimes, the results we recall use the intersection with a terminal alphabet, in that way avoiding superfluous symbols to be counted as a result of a halting computation. In that case, we add the suffix $_T$ at the end of the corresponding notation.

2.3 Tissue P Systems with Symport / Antiport Rules

Tissue P systems were introduced in [24], and tissue-like P systems with channel states were investigated in [16]. Here we deal with the following type of systems (omitting the channel states):

A tissue P system (of degree $m \geq 1$) with symport / antiport rules is a construct

$$\Pi = \left(m, O, w_1, \ldots, w_m, ch, \left(R_{(i,j)}\right)_{(i,j)\in ch}\right)$$

where

- m is the number of cells,
- O is the alphabet of *objects*,
- $w_1, \ldots, w_m$ are strings over O representing the *initial* multiset of *objects* present in the cells of the system (it is assumed that the m cells are labelled with $1, 2, \ldots, m$) and, moreover, we assume that all objects from O appear in an unbounded number in the environment,
- $ch \subseteq \{(i,j) \mid i,j \in \{0,1,2,\ldots,m\}, (i,j) \neq (0,0)\}$ is the set of links (*channels*) between cells (these were called *synapses* in [16]; 0 indicates the environment), $R_{(i,j)}$ is a finite set of symport / antiport rules associated with the channel $(i,j) \in ch$.

A *symport / antiport rule* of the form y/λ, λ/x or y/x, respectively, $x, y \in O^+$, from $R_{(i,j)}$ for the ordered pair (i, j) of cells means moving the objects specified by y from cell i (from the environment, if $i = 0$) to cell j, at the same time moving the objects specified by x in the opposite direction. For short, we shall also speak of a *tissue P system* only when dealing with a *tissue P system with symport / antiport rules* as defined above.

The computation starts with the multisets specified by $w_1, \ldots, w_m$ in the m cells; in each time unit, a rule is used on each channel for which a rule can be used (if no rule is applicable for a channel, then no object passes over it). Therefore, the use of rules is sequential at the level of each channel, but it is parallel at the level of the system: all channels which can use a rule must do it (the system is synchronously evolving). The computation is successful if and only if it halts.

The result of a halting computation is the number described by the multiplicity of objects present in cell 1 (or in the first k cells) in the halting configuration. The set of all (vectors of) natural numbers computed in this way by the system Π is denoted by $N(\Pi)$ ($Ps(\Pi)$). The family of sets $N(\Pi)$ ($Ps(\Pi)$) of (vectors of) natural numbers computed as above by systems with at most $n > 0$ symbols and $m > 0$ cells as well as with symport rules of weight $s \geq 0$ and antiport rules of weight $t \geq 0$ is denoted by

$$\mathbb{N}O_n t' P_m(sym_s, anti_t)\ (PsO_n t' P_m(sym_s, anti_t)).$$

When any of the parameters m, n, s, t is not bounded, it is replaced by $*$.

In [16], only channels (i, j) with $i \neq j$ are allowed, and, moreover, for any i, j only one channel out of $\{(i, j), (j, i)\}$ is allowed, i.e., between two cells (or one cell and the environment) only one channel is allowed (this technical detail may influence considerably the computational power). The family of sets $N(\Pi)$ ($Ps(\Pi)$) of (vectors of) natural numbers computed as above by systems with at most $n > 0$ symbols and $m > 0$ cells as well as with symport rules of weight $s \geq 0$ and antiport rules of weight $t \geq 0$ is denoted by

$$\mathbb{N}O_n t P_m(sym_s, anti_t)\ (PsO_n t P_m(sym_s, anti_t)).$$

3 Descriptional Complexity - a Historic Overview

In this section we review the development of computational completeness results with respect to descriptional complexity parameters, especially concerning the number of membranes and cells, respectively, and the weight of the rules as well as the number of objects.

3.1 Rules Involving More Than Two Objects

We first recall results where rules involving more than two objects are used. As it was shown in [27], two membranes are enough for getting computational completeness when rules involving at most four objects, moving up to two objects in each direction, are used, i.e.,

$$\mathbb{N}RE = \mathbb{N}OP_2(sym_2, anti_2).$$

Using antiport. The result stated above was independently improved in [12], [20], and [15] - one membrane is enough:

$$\mathbb{N}RE = \mathbb{N}OP_1(sym_1, anti_2).$$

In fact, only one symport rule is needed; this can be avoided for the price of one additional object in the output region:

$$\mathbb{N}_1RE = \mathbb{N}_1OP_1(anti_2).$$

It is worth mentioning that the only antiport rules used are those exchanging one object by two objects.

Using symport. The history of P systems with symport only is longer. In [23] the results

$$\mathbb{N}RE = \mathbb{N}OP_2(sym_5) = \mathbb{N}OP_3(sym_4) = \mathbb{N}OP_5(sym_3)$$

are proved, whereas in [19]

$$\mathbb{N}_{13}RE = \mathbb{N}_{13}OP_1(sym_3)$$

is shown; the additional symbols can be avoided if a second membrane is used:

$$\mathbb{N}RE = \mathbb{N}OP_2(sym_3).$$

In this paper we now will show that we can bound the number of additional symbols by 7:

$$\mathbb{N}_7RE = \mathbb{N}_7OP_1(sym_3).$$

Determinism. It is known that deterministic P systems with one membrane using only antiport rules of weight at most 2 (actually, only the rules exchanging one object for two objects are needed, see [17], [10]) or using only symport rules of weight at most 3 (see [17]) can accept all sets of vectors of natural numbers (in fact, this is only proved for sets of numbers, but the extension to sets of vectors is straightforward), i.e.,

$$PsRE = aDPsOP_1(anti_2) = aDPsOP_1(sym_3).$$

3.2 Minimal Cooperation

Already in [27] it was shown that

$$\mathbb{N}RE = \mathbb{N}OP_5(sym_2, anti_1),$$

i.e., five membranes are already enough when only rules involving two objects are used. However, both types of rules involving two objects are used: symport rules moving up to two objects in the same direction, and antiport rules moving two objects in different directions.

Minimal cooperation by antiport. We now consider P systems where symport rules move only one object and antiport rules move only two objects across the a membrane in different directions. The first proof of the computational completeness of such P systems can be found in [8]:

$$\mathbb{N}RE = \mathbb{N}OP_9(sym_1, anti_1),$$

i.e., these P systems have nine membranes. This first result was improved by reducing the number of membranes to six [21], five [9], and four [18, 22], and finally in [31] it was shown that

$$\mathbb{N}_5RE = \mathbb{N}_5OP_3(sym_1, anti_1),$$

i.e., three membranes are sufficient to generate all recursively enumerable sets of numbers (with five additional objects in the output membrane).

In [5], a stronger result was shown where the output membrane did not contain superfluous symbols:

$$PsRE = PsOP_3(sym_1, anti_1),$$

In [4] it was shown that even two membranes are enough to obtain computational completeness, yet only modulo a terminal alphabet:

$$PsRE = PsOP_2(sym_1, anti_1)_T,$$

In this paper we now will show that we can bound the number of additional symbols by 3:

$$\mathbb{N}_3RE = \mathbb{N}_3OP_2(sym_1, anti_1).$$

Minimal cooperation by symport. We now consider P systems moving only one or two objects by a symport rule; these systems were shown to be computationally complete with four membranes in [20]:

$$\mathbb{N}RE = \mathbb{N}OP_4(sym_2).$$

In [5], this result was improved down to three membranes even for vectors of natural numbers:

$$PsRE = PsOP_3(sym_2).$$

Moreover, in [5] it was also shown that even two membranes are enough to obtain computational completeness (modulo a terminal alphabet):

$$PsRE = PsOP_2(sym_2)_T$$

In this paper we will show that the number of additional objects in the output region can be bound by six:

$$\mathbb{N}_6RE = \mathbb{N}_6OP_2(sym_2)$$

The tissue case. If we do not restrict the graph of communication to be a tree, certain advantages appear. It was shown in [33] that

$$\mathbb{N}RE = \mathbb{N}OtP_3(sym_1, anti_1),$$

i.e., three cells are enough when using symport / antiport rules of weight one. This result was improved in [7] to two cells, again without additional objects in the output cell, and an equivalent result holds if antiport rules of weight one are replaced by symport rules of weight two:

$$PsRE = PsOtP_2(sym_1, anti_1) = PsOtP_2(sym_2).$$

Moreover, it was shown in the same article that accepting can be done deterministically:

$$PsRE = aDPsOtP_2(sym_1, anti_1) = aDPsOtP_2(sym_2).$$

A nice aspect of the proof is that it not only holds true for P systems with channels operating sequentially (as it is usually defined for tissue P systems), but also for P systems with channels operating in a maximally parallel way (like in standard P systems, generalizing the region communication structure of P systems to the arbitrary graph structure of tissue P systems).

Below computational completeness. In [7], it was also shown that

$$\mathbb{N}OP_1(sym_1, anti_1) \cup \mathbb{N}OtP_1(sym_1, anti_1) \subseteq NFIN.$$

Together with the counterpart results for symport systems,

$$\mathbb{N}OP_1(sym_2) \cup \mathbb{N}OtP_1(sym_2) \subseteq NFIN$$

obtained in [19], this is enough to state the optimality of the computational completeness results for the two-membrane/ two-cell systems.

The most interesting open questions remaining in the cases considered so far concern the possibility to reduce the number of extra objects in the output region in some of the results stated above.

3.3 Small Number of Objects

In the preceding subsections, a survey of computational completeness results depending on the number of *membranes* or *cells* and the *weights* of the rules has been given. We now follow another direction of descriptional complexity: we try to keep the number of *membranes* or *cells* and especially the number of *objects* small, yet on the other hand allow rules of unbounded weight.

P systems. A quite surprising result was presented in [29]: using symport / antiport rules of unbounded weight, P systems with four membranes are computationally complete even when the alphabet contains only three symbols:

$$\mathbb{N}RE = \mathbb{N}O_3P_4(sym_*, anti_*).$$

Then it has been shown in [1] that

$$\mathbb{N}RE = \mathbb{N}O_5P_1(sym_*, anti_*),$$

i.e., for P systems with one membrane, even five objects are enough for getting computational completeness.

The original result was improved in [3]; in sum, the actual results for P systems can be found there:

$$\begin{aligned} &\mathbb{N}RE = \mathbb{N}O_nP_m(sym_*, anti_*) \\ &\quad \text{for } (n,m) \in \{(5,1), (4,2), (3,3), (2,4)\}. \end{aligned}$$

The same article presents undecidability results for the families $\mathbb{N}O_2P_3(sym_*, anti_*)$ and $\mathbb{N}O_3P_2(sym_*, anti_*)$; moreover, it is shown that

$$\mathbb{N}O_1P_2(sym_*, anti_*) \cap \mathbb{N}O_2P_1(sym_*, anti_*) \supseteq \mathbb{N}REG.$$

The results mentioned above are presented as part of a general picture ("complexity carpet"), including results for generating/ accepting/ computing functions on vectors of specified dimensions.

Several questions are still open; the most interesting one is to determine the computational power of P systems with one symbol (we conjecture that they are not computationally complete, even if we can use an unbounded number of membranes and symport / antiport rules of unbounded weight).

Tissue P Systems. The question concerning systems with only one object has been answered in a positive way in [14] for tissue P systems:

$$\mathbb{N}RE = \mathbb{N}O_1tP_7(sym_*, anti_*) = \mathbb{N}O_1t'P_6(sym_*, anti_*).$$

In [2] the "complexity carpet" for tissue P systems was completed:

$$\mathbb{N}RE = \mathbb{N}O_ntP_m(sym_*, anti_*)$$
$$\text{for } (n,m) \in \{(4,2),(2,3),(1,7)\},$$

but

$$\mathbb{N}REG = \mathbb{N}O_*tP_1(sym_*, anti_*) = \mathbb{N}O_2tP_1(sym_*, anti_*)$$

and

$$\mathbb{N}FIN = \mathbb{N}O_1tP_1(sym_*, anti_*) = \mathbb{N}O_1t'P_1(sym_*, anti_*).$$

Using two channels between a cell and the environment, one cell can sometimes be saved, and one-cell systems become computationally complete:

$$\mathbb{N}RE = \mathbb{N}O_nt'P_m(sym_*, anti_*)$$
$$\text{for } (n,m) \in \{(5,1),(3,2),(2,3),(1,6)\}.$$

3.4 Computational Completeness - Summary

We now finish our historical review with repeating (some of) the best known results of computational completeness:

One membrane

$aDPsOP_1(anti_2) = aDPsOP_1(sym_3) = PsRE,$
$\mathbb{N}_1RE = \mathbb{N}_1OP_1(anti_2),$
$\mathbb{N}_7RE = \mathbb{N}_7OP_1(sym_3).$

P systems - minimal cooperation

$$PsRE = PsOP_2(sym_1, anti_1)_T = PsOP_2(sym_2)_T,$$
$$\mathbb{N}_3RE = \mathbb{N}_3OP_2(sym_1, anti_1),$$
$$\mathbb{N}_6RE = \mathbb{N}_6OP_2(sym_2).$$

Tissue P systems - minimal cooperation

$$PsRE = aDPsOtP_2(sym_1, anti_1) = aDPsOtP_2(sym_2),$$
$$PsRE = PsOtP_2(sym_1, anti_1) = PsOtP_2(sym_2).$$

P systems - small number of objects

$$\mathbb{N}RE = \mathbb{N}O_nP_m(sym_*, anti_*)$$
$$\text{for } (n,m) \in \{(5,1),(4,2),(3,3),(2,4)\}.$$

Tissue P systems - small number of objects

$$\mathbb{N}RE = \mathbb{N}O_ntP_m(sym_*, anti_*)$$
$$\text{for } (n,m) \in \{(4,2),(2,3),(1,7)\}.$$
$$\mathbb{N}RE = \mathbb{N}O_nt'P_m(sym_*, anti_*)$$
$$\text{for } (n,m) \in \{(5,1),(3,2),(2,3),(1,6)\}.$$

4 New Results

We first improve the result $\mathbb{N}_{13}OP_1(sym_3) = \mathbb{N}_{13}RE$ from [19]. For the proof, we use the variant of counter machines with conflicting counters and implement the semantics that if two conflicting counters are non-empty at the same time, then the computation is blocked without producing a result.

Theorem 1. $\mathbb{N}_7OP_1(sym_3) = \mathbb{N}_7RE$.

Proof. Let L be an arbitrary set from $\mathbb{N}_7RE$ and consider a counter automaton $M = (d, Q, q_0, q_f, P, C)$ with conflicting counters generating $L - 7$ $(= \{n-7 \mid n \in L\})$; C is a finite set of pair sets of conflicting counters $\{i, \bar{i}\}$. We construct a P system simulating M:

$$\begin{aligned}
\Pi &= (O, E, [_1\ \ ]_1, w_1, R_1, 1),\\
O &= \{x_i \mid 1 \le i \le 6\} \cup Q \cup \{(p,j) \mid p \in P, 1 \le j \le 6\}\\
&\cup \{a_i, A_i \mid i \in C\} \cup \{\#, b, d\},\\
E &= \{a_i, A_i \mid i \in C\} \cup \{x_2, x_3, \#\}\\
&\cup Q \cup \{(p,j) \mid p \in P, j \in \{2,4,5,6\}\}\\
w_1 &= l_0 d x_1 x_4 x_5 x_6 \textstyle\prod_{p \in P} (p,1)\,(p,3)\, b.
\end{aligned}$$

The following rules allow us to simulate the counter automaton M:

- The rules $(da_i a_{\bar{i}}, out)$ implement the special semantics of conflicting counters $\{i, \bar{i}\}$ with leading to an infinite computation by applying the rules $(d\#, out)$ and $(d\#, in)$.

- The simulation of the instructions of M is initiated by also sending out x_1 in the first step; the rules $(x_1 x_2 x_3, in)$ as well as $(x_2 x_4 x_5, out)$ and $(x_3 x_6, out)$ then allow us to send out the specific signal variables x_4, x_5, and x_6 which are needed to guide the sequence of rules to be applied.

- The instruction $p : (l \to l', i-)$ is simulated by the sequence of rules

  $(l(p,1)x_1, out)$,
  $((p,1)x_4(p,2), in)$,
  $((p,2)(p,3)a_i, out)$, $((p,2)(p,3)d, out)$,
  $((p,3)x_5(p,4), in)$,
  $((p,4)(p,5), out)$,
  $((p,5)x_6 l', in)$.

  In case that no symbol a_i is present (which corresponds to the fact that counter i is empty), the rule $((p,2)(p,3)d, out)$ leads to an infinite computation by applying the rules $(d\#, out)$ and $(d\#, in)$. Otherwise, decrementing is successfully accomplished by applying the rule $((p,2)(p,3)a_i, out)$.

- The instruction $p : (l \to l', i+)$ is simulated by the sequence of rules

  $(l(p,1)x_1, out)$,
  $((p,1)x_4(p,2), in)$,
  $((p,2)(p,3)A_i, out)$,
  $((p,3)x_5 l', in)$,
  $(A_i x_6 a_i, in)$.

  The symbol A_i is sent out to take exactly one symbol a_i in.

- A simulation of M by Π terminates with sending out the symbols from $\{(p,1), (p,3) \mid p \in P\} \cup \{A_i \mid i \in C\}$ which were used during the simulation of the instructions of M as soon as the halting label l_h of M appears:

  $(l_h b x, out)$,
  $x \in \{(p,1), (p,3) \mid p \in P\} \cup \{A_i \mid i \in C\}$,
  $(l_h b, in)$.

If the system halts, the objects inside correspond with the contents of the output registers, and the extra symbols are $l_h, d, b, x_1, x_4, x_5, x_6$, i.e., seven in total. □

We now show that two membranes are enough to obtain computational completeness with symport / antiport rules of minimal size 1 with only three additional objects remaining in halting computations.

Theorem 2. $\mathbb{N}_3 OP_2(sym_1, anti_1) = \mathbb{N}_3 RE$.

Proof. We simulate a counter automaton $M = (d, Q, q_0, q_f, P)$ which starts with empty counters. We also suppose that all instructions from P are labelled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$; I is the disjoint union of $\{n\}$ as well as I_+, I_-, and $I_{=0}$ where by I_+, I_-, and $I_{=0}$ we denote the set of labels for the "increment"-, "decrement"-, and "test for zero"-instructions, respectively. Additionally we suppose, without loss of generality, that on the first counter of the counter automaton M only "increment" instructions - of the form $(q_i \to q_l, c_1+)$ - are operating.

We construct the P system Π_1 as follows:

$$\begin{aligned}
\Pi_1 &= (O, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, E, R_1, R_2, 2),\\
O &= E \cup \{I_c, q_0', F_1, F_2, F_3, F_4, F_5, \#_1, \#_2, b_j, b_j' \mid j \in I\},\\
E &= Q \cup \{a_j, a_j', a_j'' \mid j \in I\} \cup C \cup \{F_2, F_3, F_4, F_5\},\\
w_1 &= q_0' I_c \#_1 \#_1 \#_2 \#_2,\\
w_2 &= F_1 F_1 F_1 \prod_{j \in I} b_j \prod_{j \in I} b_j',\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,f}, \quad i = 1, 2.
\end{aligned}$$

The functioning of this system may be split into two stages:

1. simulating the instructions of the counter automaton.
2. terminating the computation.

We code the counter automaton as follows:

Region 1 will hold the current state of the automaton, represented by a symbol $q_i \in Q$; region 2 will hold the value of all counters, represented by the number of occurrences of symbols $c_k \in C$, $k \in D$, where $D = \{1, ..., d\}$.

We also use the following idea realized by the phase "START" below: from the environment, we bring symbols c_k into region 1 all the time during the computation. This process may only be stopped if all stages finish correctly; otherwise, the computation will never stop.

We split our proof into several parts that depend on the logical separation of the behavior of the system. We will present the rules and the initial symbols for each part, but we remark that the system we present is the union of all these parts. The rules R_i are given by three phases:

1. START (stage 1);
2. RUN (stage 1);
3. END (stage 2).

The parts of the computations illustrated in the following describe different stages of the evolution of the P system given in the corresponding theorem. For simplicity, we focus on explaining a particular stage and omit the objects that do not participate in the evolution at that time. Each rectangle represents a membrane, each variable represents a copy of an object in a corresponding membrane (symbols outside of the outermost rectangle are found in the environment). In each step, the symbols that will evolve (will be moved) are written in boldface. The labels of the applied rules are written above the symbol $\Rightarrow$.

1. START.

$$\begin{aligned} R_{1,s} &= \{\texttt{1s1} : (I_c, in), \texttt{1s2} : (I_c, out; c_k, in), \texttt{1s3} : (c_k, out) \mid c_k \in C\} \\ &\cup \{\texttt{1s4} : (q_0', out; q_0, in)\}, \\ R_{2,s} &= \emptyset \end{aligned}$$

Symbol I_c brings one symbol c_k from the environment into region 1 (rules `1s1, 1s2`), where it may be used immediately during the simulation of the "increment" instruction and then moved to region 2. Otherwise symbol c_k returns to the environment (rule `1s3`). Rule `1s4` is used for synchronizing the appearance of the symbols c_k and q_i in region 1.

We illustrate the beginning of the computation as follows:

$$\mathbf{c_{k_1}}\,\mathbf{q_0}\,a_j\,c_{k_2}\,\boxed{\mathbf{q_0'}\,\mathbf{I_c}\,\boxed{b_j}} \Rightarrow^{\texttt{1s2,1s4}} \mathbf{I_c}\,q_0'\,\mathbf{a_j}\,c_{k_2}\,\boxed{\mathbf{q_0}\,\mathbf{c_{k_1}}\,\boxed{b_j}} \Rightarrow^{\texttt{1s1,1s3,1r1}}$$

$$q_0'\,q_0\,c_{k_1}\,\mathbf{c_{k_2}}\,\boxed{\mathbf{a_j}\,\mathbf{I_c}\,\boxed{\mathbf{b_j}}} \Rightarrow^{\texttt{1s2,2r1}} q_0'\,q_0\,c_{k_1}\,\mathbf{I_c}\,\boxed{c_{k_2}\,\mathbf{b_j}\,\boxed{\mathbf{a_j}}} \cdots$$

2. RUN.

$$\begin{aligned}
R_{1,r} &= \{\texttt{1r1} : (q_i, out; a_j, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, = 0\}\} \\
&\cup \{\texttt{1r2} : (b_j, out; a_j', in), \texttt{1r3} : (a_j, out; b_j, in), \\
&\quad \texttt{1r4} : (\#_1, out; b_j, in) \mid j \in I\} \\
&\cup \{\texttt{1r5} : (a_j', out; a_j'', in) \mid j \in I_+ \cup I_-\} \cup \{\texttt{1r6} : (\#_1, out; \#_1, in)\} \\
&\cup \{\texttt{1r7} : (b_j', out; a_j'', in), \texttt{1r8} : (a_j', out; b_j', in), \\
&\quad \texttt{1r9} : (\#_1, out; b_j', in) \mid j \in I_{=0}\} \\
&\cup \{\texttt{1r10} : (a_j'', out, q_l, in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{+, -, = 0\}\} \\
&\cup \{\texttt{1r11} : (b_j, out), \texttt{1r12} : (b_j', out) \mid j \in I\}, \\
R_{2,r} &= \{\texttt{2r1} : (b_j, out; a_j, in) \mid j \in I\} \\
&\cup \{\texttt{2r2} : (a_j, out; c_k, in) \mid (j : q_i \to q_l, c_k+) \in P\} \\
&\cup \{\texttt{2r3} : (a_j', in) \mid j \in I_+\} \\
&\cup \{\texttt{2r4} : (a_j', out; b_j, in) \mid j \in I_+ \cup I_-\} \\
&\cup \{\texttt{2r5} : (a_j, out) \mid j \in I_- \cup I_{=0}\} \\
&\cup \{\texttt{2r6} : (c_k, out; a_j', in) \mid (j : q_i \to q_l, c_k\gamma) \in P, \gamma \in \{-, = 0\}\} \\
&\cup \{\texttt{2r7} : (b_j', out; b_j, in), \texttt{2r8} : (b_j', in) \mid j \in I_{=0}\} \\
&\cup \{\texttt{2r9} : (a_j, out; \#_2, in) \mid j \in I_+\} \cup \{\texttt{2r10} : (\#_2, out; \#_2, in)\}.
\end{aligned}$$

"Increment"-instruction:

$$\mathbf{a_j}\,a_j'\,a_j''\,q_l\,\boxed{\mathbf{q_i}\,c_k\#_1\#_1\,\boxed{b_j}} \Rightarrow^{\texttt{1r1}} a_j'\,a_j''\,q_i\,q_l\,\boxed{\mathbf{a_j}\,c_k\#_1\#_1\,\boxed{\mathbf{b_j}}} \Rightarrow^{\texttt{2r1}}$$

$$\mathbf{a_j'}\,a_j''\,q_i\,q_l\,\boxed{\mathbf{b_j}\,\mathbf{c_k}\#_1\#_1\,\boxed{\mathbf{a_j}}} \Rightarrow^{\texttt{1r2,2r2}} \mathbf{b_j}\,a_j''\,q_i\,q_l\,\boxed{\mathbf{a_j}\,\mathbf{a_j'}\#_1\#_1\,\boxed{c_k}}$$

Now there are two possibilities: we may either apply
a) rule 1r5 or
b) rule 2r3.

It is easy to see that **case a)** leads to an infinite computation:

$$\mathbf{b_j}\mathbf{a_j''}q_iq_l\boxed{\mathbf{a_j}\mathbf{a_j'}\#_1\#_1\boxed{c_k}} \Rightarrow^{\texttt{1r5,1r3}}$$

$$a_j\mathbf{a_j'}q_i\mathbf{q_l}\boxed{\mathbf{b_j}\mathbf{a_j''}\#_1\#_1\boxed{c_k}} \Rightarrow^{\texttt{1r2,1r10}} a_j\mathbf{b_j}q_i\mathbf{a_j''}\boxed{\mathbf{a_j'}\mathbf{q_l}\#_\mathbf{1}\#_1\boxed{c_k}}$$

After that rule `1r4` will eventually be applied, object $\#_1$ will be moved to the environment and then applying rule `1r6` leads to an infinite computation.

Now let us consider **case b)**:

$$\mathbf{b_j}a_j''q_iq_l\boxed{\mathbf{a_j}\mathbf{a_j'}\#_1\#_1\boxed{c_k}} \Rightarrow^{\texttt{1r3,2r3}} a_ja_j''q_iq_l\boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j'}c_k}}$$

We cannot apply rule `1r2` as this leads to an infinite computation (see above). Hence, rule `2r4` has to be applied:

$$a_ja_j''q_iq_l\boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j'}c_k}} \Rightarrow^{\texttt{2r4}} a_j\mathbf{a_j''}q_iq_l\boxed{\mathbf{a_j'}\#_1\#_1\boxed{b_jc_k}} \Rightarrow^{\texttt{1r5}}$$

$$a_ja_j'q_i\mathbf{q_l}\boxed{\mathbf{a_j''}\#_1\#_1\boxed{b_jc_k}} \Rightarrow^{\texttt{1r10}} a_ja_j'a_j''q_i\boxed{\mathbf{q_l}\#_1\#_1\boxed{b_jc_k}}$$

In that way, q_i is replaced by q_l and c_k is moved from region 1 into region 2.

"Decrement"-instruction:

$$\mathbf{a_j}a_j'a_j''q_l\boxed{\mathbf{q_i}\#_1\#_1\boxed{b_jc_k}} \Rightarrow^{\texttt{1r1}} a_j'a_j''q_iq_l\boxed{\mathbf{a_j}\#_1\#_1\boxed{\mathbf{b_j}c_k}} \Rightarrow^{\texttt{2r1}}$$

$$\mathbf{a_j'}a_j''q_iq_l\boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j}c_k}} \Rightarrow^{\texttt{1r2,2r5}} \mathbf{b_j}a_j''q_iq_l\boxed{\mathbf{a_j}\mathbf{a_j'}\#_1\#_1\boxed{c_k}} \Rightarrow^{\texttt{1r3,2r6}}$$

$$a_ja_j''q_iq_l\boxed{\mathbf{b_j}c_k\#_1\#_1\boxed{\mathbf{a_j'}}} \Rightarrow^{\texttt{2r4}} a_j\mathbf{a_j''}q_iq_l\boxed{\mathbf{a_j'}c_k\#_1\#_1\boxed{b_j}} \Rightarrow^{\texttt{1r5}}$$

$$a_ja_j'q_i\mathbf{q_l}\boxed{\mathbf{a_j''}c_k\#_1\#_1\boxed{b_j}} \Rightarrow^{\texttt{1r10}} a_ja_j'a_j''q_i\boxed{\mathbf{q_l}c_k\#_1\#_1\boxed{b_j}}$$

In the way described above, q_i is replaced by q_l and c_k is removed from region 2 to region 1.

"Test for zero"-instruction:

q_i is replaced by q_l if there is no c_k in region 2, otherwise a_j' in region 1 exchanges with c_k in region 2 and the computation will never stop.

(i) *There is no c_k in region 2:*

$$\mathbf{a_j}a'_ja''_jq_l\boxed{\mathbf{q_i}\#_1\#_1\boxed{b_jb'_j}} \Rightarrow^{\texttt{1r1}} a'_ja''_jq_iq_l\boxed{\mathbf{a_j}\#_1\#_1\boxed{\mathbf{b_j}b'_j}} \Rightarrow^{\texttt{2r1}}$$
$$a'_ja''_jq_iq_l\boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j}b'_j}}$$

Now there are two possibilities: we apply either
a) rule `2r7` or
b) rule `1r2`.

It is easy to see that **case a)** leads to an infinite computation:

$$a'_ja''_jq_iq_l\boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_jb'_j}}} \Rightarrow^{\texttt{2r7,2r5}} a'_ja''_jq_iq_l\boxed{\mathbf{a_jb'_j}\#_1\#_1\boxed{\mathbf{b_j}}} \Rightarrow^{\texttt{2r1,2r8}}$$
$$a'_ja''_jq_iq_l\boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_jb'_j}}} \Rightarrow^{\texttt{2r7,2r5}} \ldots \Rightarrow^{\texttt{2r1,2r8}}$$
$$\mathbf{a'_j}a''_jq_iq_l\boxed{\mathbf{b_j}\#_1\#_1\boxed{\mathbf{a_j}b'_j}} \Rightarrow^{\texttt{1r2,2r5}}$$
$$\mathbf{b_j}a''_jq_iq_l\boxed{\mathbf{a_j}a'_j\#_1\#_1\boxed{b'_j}} \Rightarrow^{\texttt{1r3}} a_ja''_jq_iq_l\boxed{\mathbf{b_j}a'_j\#_1\#_1\boxed{b'_j}}$$

Again there are two possibilities: we can apply either
c) rule `1r2` or
d) rule `2r7`.

Case c) leads to an infinite computation (rules `1r4` and `1r6`).

Now let us consider **case d)**:

$$a_ja''_jq_iq_l\boxed{\mathbf{b_j}a'_j\#_1\#_1\boxed{\mathbf{b'_j}}} \Rightarrow^{\texttt{2r7}} a_j\mathbf{a''_j}q_iq_l\boxed{\mathbf{b'_j}a'_j\#_1\#_1\boxed{b_j}} \Rightarrow^{\texttt{1r7}}$$
$$a_j\mathbf{b'_j}q_i\mathbf{q_l}\boxed{\mathbf{a''_ja'_j}\#_1\#_1\boxed{b_j}} \Rightarrow^{\texttt{1r8,1r10}} a_ja'_ja''_jq_i\boxed{\mathbf{q_lb'_j}\#_1\#_1\boxed{b_j}}$$

There are two possibilities: we can apply either
e) rule `1r7` or
f) rule `2r8`.

Case e) leads to infinite computation (rules `1r9` and `1r6`).

In **case f)**, the object b'_j comes back to region 2.

(b) *There is some c_k in region 2:*

Consider again **case d)**:

$a_j a''_j q_i q_l \boxed{\mathbf{b_j} \mathbf{a'_j} \#_1 \#_1 \boxed{\mathbf{b'_j} \mathbf{c_k}}} \quad \Rightarrow^{\texttt{2r7,2r6}} \quad a_j \mathbf{a''_j} q_i q_l \boxed{\mathbf{b'_j} c_k \#_1 \#_1 \boxed{a'_j b_j}}$

$\Rightarrow^{\texttt{1r7}}$

$a_j \mathbf{b'_j} q_i \mathbf{q_l} \boxed{\mathbf{a''_j} c_k \#_1 \#_1 \boxed{a'_j b_j}} \Rightarrow^{\texttt{1r9,1r10}} a_j a''_j \#_1 q_i \boxed{\mathbf{q_l} \mathbf{b'_j} c_k \#_1 \boxed{a'_j b_j}}$

Now the application of rule `1r6` leads to an infinite computation.

Finally, let us notice that applying the rules `1r11` and `1r12` during the phase RUN leads to infinite computation. Hence, we model correctly the "test for zero" instruction.

3. END.

$$
\begin{aligned}
R_{1,f} &= \{\texttt{1f1} : (F_1, out; F_2, in), \texttt{1f2} : (F_2, out; F_3, in), \\
&\quad \texttt{1f3} : (F_3, out; F_4, in), \texttt{1f4} : (F_4, out; F_5, in), \\
R_{2,f} &= \{\texttt{2f1} : (F_1, out; q_f, in), \texttt{2f2} : (q_f, out; I_c, in), \\
&\quad \texttt{2f3} : (q_f, out; \#_1, in), \texttt{2f4} : (q_f, out; \#_2, in), \texttt{2f5} : (F_5, out), \\
&\quad \texttt{2f6} : (b_j, out; F_5, in), \texttt{2f7} : (b'_j, out; F_5, in)\}.
\end{aligned}
$$

We illustrate the end of computations as follows:

$F_2 F_3 F_4 F_5 I_c c_{k_1} c_{k_2} \boxed{q_f \#_1 \#_1 \#_2 \#_2 \boxed{F_1 F_1 F_1 b_{j_1} b'_{j_2}}}$

$\Rightarrow^{\texttt{2f1,1s1}}$

$F_2 F_3 F_4 F_5 c_{k_1} c_{k_2} \boxed{I_c \#_1 \#_1 \#_2 \#_2 F_1 \boxed{q_f F_1 F_1 b_{j_1} b'_{j_2}}}$

$\Rightarrow^{\texttt{2f3,1s2,1f1}}$

$F_2 F_3 F_4 F_5 I_c c_{k_2} F_1 \boxed{F_2 c_{k_1} \#_1 \#_2 \#_2 q_f \boxed{\#_1 F_1 F_1 b_{j_1} b'_{j_2}}}$

$\Rightarrow^{\texttt{1s1,1s4,1f2,2f1}}$

$F_2 F_3 F_4 F_5 c_{k_1} c_{k_2} F_1 \boxed{F_3 I_c \#_1 \#_2 \#_2 F_1 \boxed{q_f \#_1 F_1 b_{j_1} b'_{j_2}}}$

$\Rightarrow^{\texttt{1s2,1f1,1f3,2f3}}$

$F_2 F_3 F_4 F_5 c_{k_1} I_c F_1 F_1 \boxed{F_2 F_4 c_{k_2} \#_2 \#_2 q_f \boxed{\#_1 \#_1 F_1 b_{j_1} b'_{j_2}}}$

$\Rightarrow^{\texttt{1s1,1s4,1f2,1f4,2f1}}$

$F_2F_3F_4F_5c_{k_1}c_{k_2}F_1F_1\boxed{F_3F_5I_c\#_2\#_2F_1\boxed{q_f\#_1\#_1b_{j_1}b'_{j_2}}}$

Notice that now rule `2f2` will be applied eventually, as otherwise the application of rule `2f4` will lead to an infinite computation (rule `2r10`). Hence, we continue as follows:

$F_2F_3F_4F_5c_{k_1}c_{k_2}F_1F_1\boxed{F_3F_5I_c\#_2\#_2F_1\boxed{q_f\#_1\#_1b_{j_1}b'_{j_2}}}$

$\Rightarrow^{\texttt{1f1,1f3,2f2,2f6}}$

$F_2F_3F_4F_5c_{k_1}c_{k_2}F_1F_1F_1\boxed{F_2F_4\#_2\#_2b_{j_1}q_f\boxed{I_c\#_1\#_1F_5b'_{j_2}}}$

$\Rightarrow^{\texttt{1f2,1f4,1r11,2f5}}$

$F_2F_3F_4F_5c_{k_1}c_{k_2}F_1F_1F_1b_{j_1}\boxed{F_3F_5F_5\#_2\#_2q_f\boxed{I_c\#_1\#_1b'_{j_2}}}$

We continue in this manner until all objects $b_j, b'_j,\ j \in I$ from the elementary membrane 2 have been moved to the environment. Notice that the result in the elementary membrane 2 (multiset c_1^t) cannot be changed during phase END, as object I_c now is situated in the elementary membrane and cannot bring symbols c_1 from the environment. Recall that the counter automaton can only increment the first counter c_1, so all other computations of P system Π_1 cannot change the number of symbols c_1 in the elementary membrane. Thus, at the end of a terminating computation, in the elementary membrane there are the result (multiset c_1^t) and only the three additional objects $I_c, \#_1, \#_1$. □

A "dual" class of systems with minimal cooperation is the class where two objects are moved across the membrane in the same direction rather than in the opposite ones. We now prove a similar result for this class using six additional symbols.

Theorem 3. $\mathbb{N}_6OP_2(sym_2) = \mathbb{N}_6RE$.

Proof. As in the proof of Theorem 1 we simulate a counter automaton $M = (d, Q, q_0, q_f, P)$ that starts with empty counters. Again we suppose that all instructions from P are labelled in a one-to-one manner with elements of $\{1, \ldots, n\} = I$ and that I is the disjoint union of $\{n\}$ as well as I_+, I_-, and $I_{=0}$ where by I_+, I_-, and $I_{=0}$ we denote the set of labels for the

"increment"-, "decrement"-, and "test for zero"-instructions, respectively. Moreover, we define $I' = \{1, 2, \ldots, n+4\}$, $Q^k = \{q_i^k\}$, $1 \leq k \leq 5$, $i \in K$, $K = \{0, 1, \ldots, f\}$, and $C = \{c_i \mid 1 \leq i \leq d\}$.

We construct the P system Π_2 as follows:

$$
\begin{aligned}
\Pi_2 &= (O, [_1\, [_2\ \ ]_2\,]_1, w_1, w_2, E, R_1, R_2, 2),\\
O &= \{\#_0, \#_1, \#_2, \$^1, \$^2, \$^3, \hat{a}, \hat{b}, I_c\} \cup \{a^k \mid 1 \leq k \leq 5\} \cup Q \bigcup_{1 \leq k \leq 5} Q^k \cup\\
&\quad C \cup \{a_j, a'_j, \check{a}_j, \hat{a}_j, b_j, d_j, d'_j, d''_j \mid j \in I\} \cup \{e_t, h_t \mid t \in I'\}\\
E &= \{a^1, a^3, a^5, \#_0\} \cup \{a_j, a'_j \mid j \in I\} \cup \{h_t \mid t \in I'\} \cup Q \cup Q^2 \cup Q^4 \cup C,\\
w_1 &= \#_1 \hat{a}\hat{b}a^2a^4\$^3 \prod_{j \in I} \check{a}_j \prod_{j \in I} d'_j \prod_{j \in I} d''_j \prod_{t \in I'} e_t \prod_{i \in K} \hat{q}_i, \prod_{i \in K} q_i^1, \prod_{i \in K} q_i^3, \prod_{i \in K} q_i^5\\
w_2 &= \#_2 \underbrace{\$^1\$^1 \ldots \$^1}_{n+1} \$^2 \prod_{j \in I} \hat{a}_j \prod_{j \in I} b_j \prod_{j \in I} d_j,\\
R_i &= R_{i,s} \cup R_{i,r} \cup R_{i,f}, i \in \{1, 2\}.
\end{aligned}
$$

The functioning of this system again may be split into two stages:

1. simulating the instructions of the counter automaton;
2. terminating the computation.

We code the counter automaton as in Theorem 1 above: region 1 will hold the current state of the automaton, represented by a symbol $q_i \in Q$; region 2 will hold the value of all counters, represented by the number of occurrences of symbols $c_k \in C$, $k \in D$, where $D = \{1, ..., d\}$. We also use the following idea (called "*Circle*") realized by phase "START" below: from the environment, we bring symbols c_k into region 1 all the time during the computation. This process may only be stopped if all stages finish correctly; otherwise, the computation will never stop.

We split our proof into several parts that depend on the logical separation of the behavior of the system. We will present the rules and the initial symbols for each part, but we remark that the system that we present is the union of all these parts.

The rules R_i again are given by three phases:

1. START (stage 1);
2. RUN (stage 1);

3. END (stage 2).

1. START.

$$\begin{aligned} R_{1,s} &= \{\texttt{1s1} : (I_c, out), \texttt{1s2} : (I_c c_k, in), \texttt{1s3} : (c_k, out) \mid k \in D\}, \\ R_{2,s} &= \emptyset. \end{aligned}$$

Symbol I_c brings one symbol $c \in C$ from the environment into region 1 (rules `1s1, 1s2`) where it may be used immediately during the simulation of an "increment"-instruction and moved to region 2. Otherwise symbol c returns to the environment (rule `1s3`).

2. RUN.

$$\begin{aligned} R_{1,r} &= \{\texttt{1r1} : (q_i \hat{q}_i, out) \mid i \in K\} \\ &\cup \{\texttt{1r2} : (a_j \hat{q}_i, in) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{+, -, = 0\}, k \in D\} \\ &\cup \{\texttt{1r3} : (a_j \hat{a}, out) \mid j \in I_+ \cup I_-\} \cup \{\texttt{1r4} : (a_j \hat{b}, out) \mid j \in I_{=0}\} \\ &\cup \{\texttt{1r5} : (\#_2, out), \texttt{1r6} : (\#_2, in)\} \cup \{\texttt{1r7} : (b_j \check{a}_j, out) \mid j \in I\} \\ &\cup \{\texttt{1r8} : (b_j \#_1, out) \mid j \in I\} \cup \{\texttt{1r9} : (\hat{a}_j \#_1, out) \mid j \in I\} \\ &\cup \{\texttt{1r10} : (\#_0 \#_1, in), \texttt{1r11} : (\#_0 \hat{b}, in)\} \cup \{\texttt{1r12} : (a'_j b_j, in) \mid j \in I\} \\ &\cup \{\texttt{1r13} : (\hat{a} a^1, in), \texttt{1r14} : (a^1 a^2, out), \texttt{1r15} : (a^2 a^3, in)\} \\ &\cup \{\texttt{1r16} : (a^3 a^4, out), \texttt{1r17} : (a^4 a^5, in), \texttt{1r18} : (a^5, out)\} \\ &\cup \{\texttt{1r19} : (a'_j q_l^1, out) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{+, -, = 0\}, k \in D\} \\ &\cup \{\texttt{1r20} : (q_i^1 q_i^2, in), \texttt{1r21} : (q_i^2 q_i^3, out), \texttt{1r22} : (q_i^3 q_i^4, in) \mid i \in K\} \\ &\cup \{\texttt{1r23} : (q_i^4 q_i^5, out), \texttt{1r24} : (q_i^5 q_i, in) \mid i \in K\} \\ &\cup \{\texttt{1r25} : (d_j \hat{a}, out), \texttt{1r26} : (d_j \#_0, in) \mid j \in I_+ \cup I_-\} \\ &\cup \{\texttt{1r27} : (d_j \check{a}_j, in) \mid j \in I\} \cup \{\texttt{1r28} : (d_j \#_1, out) \mid j \in I_+ \cup I_-\} \\ &\cup \{\texttt{1r29} : (d_j d'_j, out) \mid j \in I_{=0}\} \cup \{\texttt{1r30} : (d'_j \hat{b}, in) \mid j \in I_{=0}\}, \end{aligned}$$

$$\begin{aligned} R_{2,r} &= \{\texttt{2r1} : (a_j \check{a}_j, in) \mid j \in I\} \cup \{\texttt{2r2} : (b_j \check{a}_j, out) \mid j \in I\} \\ &\cup \{\texttt{2r3} : (a_j c_k, out) \mid (j : q_i \to q_l, k\gamma) \in P, \gamma \in \{-, = 0\}, k \in D\} \\ &\cup \{\texttt{2r4} : (a_j \#_2, out) \mid j \in I_-\} \cup \{\texttt{2r5} : (a_j \hat{a}_j, out) \mid j \in I_+\} \\ &\cup \{\texttt{2r6} : (\#_0, in), \texttt{2r7} : (\#_0, out)\} \\ &\cup \{\texttt{2r8} : (c_k \hat{a}_j, in) \mid (j : q_i \to q_l, k+) \in P, k \in D\} \\ &\cup \{\texttt{2r9} : (a'_j b_j, in) \mid j \in I\} \cup \{\texttt{2r10} : (a'_j d_j, out) \mid j \in I\} \\ &\cup \{\texttt{2r11} : (d_j a^5, in) \mid j \in I_+ \cup I_-\} \cup \{\texttt{2r12} : (a^5, out)\} \\ &\cup \{\texttt{2r13} : (d_j d''_j, in) \mid j \in I_{=0}\} \cup \{\texttt{2r14} : (a_j d''_j, out) \mid j \in I_{=0}\}. \end{aligned}$$

"Increment"-instruction:

$$a_j c \boxed{\mathbf{I_c q_i \hat{q}_i} \check{a}_j \hat{a} \boxed{b_j \hat{a}_j}} \Rightarrow^{\texttt{1r1,1s1}} q_i \mathbf{\hat{q}_i a_j I_c c} \boxed{\check{a}_j \hat{a} \boxed{b_j \hat{a}_j}} \Rightarrow^{\texttt{1r2,1s2}}$$

$$q_i \boxed{I_c c \hat{q}_i a_j \check{a}_j \hat{a} \boxed{b_j \hat{a}_j}}, \text{ where } c \in C$$

Now there are two variants of computations (depending on the application of rule `2r1` or rule `1r3`). It is easy to see that the application of rule `1r3` leads to an infinite computation (by *"Circle"*). Consider applying rule `2r1`:

$$q_i c_k \boxed{\mathbf{I_c c} \hat{q}_i \mathbf{a_j} \check{\mathbf{a}}_\mathbf{j} \hat{a} \boxed{b_j \hat{a}_j}} \Rightarrow^{\texttt{2r1,1s1,1s3}}$$

$$q_i \mathbf{I_c c_k} c \boxed{\hat{q}_i \hat{a} \boxed{\mathbf{b_j} \check{\mathbf{a}}_\mathbf{j} \mathbf{a_j} \hat{\mathbf{a}}_\mathbf{j}}} \Rightarrow^{\texttt{2r2,2r5,1s2}}$$

$$q_i c \boxed{I_c c_k \hat{q}_i \hat{a} b_j \check{a}_j a_j \hat{a}_j \boxed{}}$$

Notice that object $\hat{a}_j$ cannot be idle, as the application of the rules `1r9`, `1r10`, `2r6`, `2r7` leads to an infinite computation. Hence, rule `2r8` will be applied and object c_k will be moved to region 2 (thus, we increase the number of objects c_k in region 2 by one and model the increment-instruction of the counter automaton). In an analogous way, object b_j cannot be idle, as applying rules `1r8`, `1r10`, `2r6`, `2r7` leads to an infinite computation. Thus, rule `2r1` cannot be applied and rule `1r7` will eventually be applied.

$$c a'_j a^1 a^3 a^5 \boxed{\mathbf{I_c c_k} \hat{q}_i \hat{a} \mathbf{b_j} \check{\mathbf{a}}_\mathbf{j} \mathbf{a_j} \hat{\mathbf{a}}_\mathbf{j} a^2 a^4 q_l^1 \boxed{}}$$

$$\mathbf{I_c c a'_j b_j} \check{a}_j a_j \mathbf{\hat{a} a^1} a^3 a^5 \boxed{\hat{q}_i a^2 a^4 q_l^1 \boxed{\hat{a}_j c_k}} \Rightarrow^{\texttt{1r12,1r13,1s2}}$$

$$\check{a}_j a_j a^3 a^5 \boxed{I_c c \hat{q}_i \hat{a} a^1 a^2 a^4 q_l^1 a'_j b_j \boxed{\hat{a}_j c_k}}$$

Notice that applying rule `1r19` leads to an infinite computation, as object b_j cannot be idle. Thus, rule `2r9` will eventually be applied.

$$\check{a}_j a_j a^3 a^5 q_l^2 q_l^4 \boxed{\mathbf{I_c c} \hat{q}_i \hat{a} \mathbf{a^1 a^2} a^4 q_l^1 \mathbf{a'_j b_j} q_l^3 q_l^5 \boxed{d_j \hat{a}_j c_k}}$$

$$\Rightarrow^{\texttt{2r9,1r14,1s1,1s3}}$$

$\mathbf{I_c c}\check{a}_j a_j a^1 \mathbf{a^2 a^3} a^5 q_l^2 q_l^4 \boxed{\hat{q}_i \hat{a} a^4 q_l^1 q_l^3 q_l^5 \boxed{\mathbf{d_j a'_j} b_j \hat{a}_j c_k}}$

$\Rightarrow^{\texttt{2r10,1r15,1s2}}$

$\check{a}_j a_j a^1 a^5 q_l^2 q_l^4 \boxed{\mathbf{I_c c} \hat{q}_i a^2 \mathbf{a^3 a^4 \hat{a} d_j a'_j q_l^1} q_l^3 q_l^5 \boxed{b_j \hat{a}_j c_k}}$

$\Rightarrow^{\texttt{1r19,1r25,1r16,1s1,1s3}}$

$\mathbf{I_c c} a_j \mathbf{\check{a}_j d_j \hat{a} a^1} a^3 \mathbf{a^4 a^5} a'_j \mathbf{q_l^1 q_l^2} q_l^4 \boxed{\hat{q}_i a^2 q_l^3 q_l^5 \boxed{b_j \hat{a}_j c_k}}$

$\Rightarrow^{\texttt{1r27,1r13,1r17,1r20,1s2}}$

$a_j a^3 a'_j q_l^4 \boxed{I_c c \hat{q}_i \hat{a} a^1 a^2 a^4 \check{a}_j d_j a^5 q_l^1 q_l^2 q_l^3 q_l^5 \boxed{b_j \hat{a}_j c_k}}$

Now we can apply the rules `1r25`, `1r18` or `2r11`. It is easy to see that applying rule `1r25` leads to an infinite computation (rules `1r26`, `2r6`, `2r7`), which is true for rule `1r18`, too (rules `1r28`, `1r10`, `2r6`, `2r7`). Hence, now consider applying rule `2r11`.

$a_j a^3 a'_j q_l^4 q_l \boxed{\mathbf{I_c c} \hat{q}_l \hat{q}_i \hat{a} \mathbf{a^1 a^2} a^4 \check{a}_j \mathbf{d_j a^5} q_l^1 \mathbf{q_l^2 q_l^3} q_l^5 \boxed{b_j \hat{a}_j c_k}}$

$\Rightarrow^{\texttt{2r11,1r21,1r14,1s1,1s3}}$

$\mathbf{I_c c} a_j a^1 \mathbf{a^2 a^3} a'_j q_l^2 \mathbf{q_l^3 q_l^4} q_l \boxed{\hat{q}_l \hat{q}_i \hat{a} a^4 \check{a}_j q_l^1 q_l^5 \boxed{d_j \mathbf{a^5} b_j \hat{a}_j c_k}}$

$\Rightarrow^{\texttt{2r12,1r15,1r22,1s2}}$

$a_j a^1 a'_j q_l^2 q_l \boxed{\mathbf{I_c c} \hat{q}_l \hat{q}_i \hat{a} a^2 \mathbf{a^3 a^4 a^5} \check{a}_j q_l^1 q_l^3 \mathbf{q_l^4 q_l^5} \boxed{d_j b_j \hat{a}_j c_k}}$

$\Rightarrow^{\texttt{1r16,1r18,1r23,1s1,1s3}}$

$\mathbf{I_c c} a_j a^1 a^3 \mathbf{a^4 a^5} a'_j q_l^2 q_l^4 \mathbf{q_l^5 q_l} \boxed{\hat{q}_l \hat{q}_i \hat{a} a^2 \check{a}_j q_l^1 q_l^3 \boxed{d_j b_j \hat{a}_j c_k}}$

$\Rightarrow^{\texttt{1r17,1r24,1s2}}$

$a_j a^1 a^3 a'_j q_l^2 q_l^4 \boxed{\mathbf{I_c c q_l \hat{q}_l} \hat{q}_i \hat{a} a^2 a^4 \mathbf{a^5} \check{a}_j q_l^1 q_l^3 q_l^5 \boxed{d_j b_j \hat{a}_j c_k}}$

$\Rightarrow^{\texttt{1r1,1r18,1s1,1s3}}$

$I_c c a_j a^1 a^3 a^5 a'_j q_l^2 q_l^4 q_l \hat{q}_l \boxed{\hat{q}_i \hat{a} a^2 a^4 \check{a}_j q_l^1 q_l^3 q_l^5 \boxed{d_j b_j \hat{a}_j c_k}}$

Thus, we begin a new circle of modelling.

"Decrement"-instruction.

If there is an object c_k in region 2, we obtain the following computation:

$$a_j \boxed{\mathbf{q_i}\hat{\mathbf{q}}_\mathbf{i}\check{a}_j\hat{a}\boxed{b_jc_k\#_2}} \Rightarrow^{\texttt{1r1}} q_i\hat{\mathbf{q}}_\mathbf{i}\mathbf{a_j}\boxed{\check{a}_j\hat{a}\boxed{b_jc_k\#_2}} \Rightarrow^{\texttt{1r2}}$$
$$q_i\boxed{\hat{q}_ia_j\check{a}_j\hat{a}\boxed{b_jc_k\#_2}}$$

Now there are two variants of computations (depending on the application of rule `2r1` or rule `1r3`). It is easy to see that the application of rule `1r3` leads to an infinite computation (by *"Circle"*). Now consider applying rule `2r1`:

$$q_i\boxed{\hat{q}_i\mathbf{a_j}\check{\mathbf{a}}_\mathbf{j}\hat{a}\boxed{b_jc_k\#_2}} \Rightarrow^{\texttt{2r1}} q_i\boxed{\hat{q}_i\hat{a}\boxed{\mathbf{b_j}\check{\mathbf{a}}_\mathbf{j}\mathbf{a_j}\mathbf{c_k}\#_2}} \Rightarrow^{\texttt{2r2,2r3}}$$
$$q_i\boxed{\hat{q}_ib_j\check{a}_j\hat{a}a_jc_k\boxed{\#_2}}$$

Thus, object c_k is moved from region 2 to region 1 (thus, we decrease the number of objects c_k in region 2 by one and model the "decrement"-instruction of the counter automaton).

The case when there is no object c_k in region 2 leads to an infinite computation (rules `2r4`, `1r5`, `1r6`), hence, again we correctly model the "decrement"-instruction. The further behavior of the system is the same as in the case of modelling the "increment"-instruction.

"Test for zero"-instruction:

q_i is replaced by q_l if there is no c_k in region 2 (case a)), otherwise the computation will never stop (case b)).

Case a):

$$a_j \boxed{\mathbf{q_i}\hat{\mathbf{q}}_\mathbf{i}\check{a}_j\hat{b}d'_jd''_j\boxed{b_jd_j\#_2}} \Rightarrow^{\texttt{1r1}} q_i\hat{\mathbf{q}}_\mathbf{i}\mathbf{a_j}\boxed{\check{a}_j\hat{b}d'_jd''_j\boxed{b_jd_j\#_2}} \Rightarrow^{\texttt{1r2}}$$
$$q_i\boxed{\hat{q}_ia_j\check{a}_j\hat{b}d'_jd''_j\boxed{b_jd_j\#_2}}$$

Now there are two variants of computations (depending on the application of rule `2r1` or rule `1r4`). It is easy to see that the application of rule `1r4` leads to an infinite computation (by *"Circle"*). Consider the application of rule `2r1`:

$$q_iq_l^2q_l^4q_la'_j\boxed{\hat{q}_i\mathbf{a_j}\check{\mathbf{a}}_\mathbf{j}q_l^1q_l^3q_l^5\hat{b}d'_jd''_j\boxed{b_jd_j\#_2}} \Rightarrow^{\texttt{2r1}}$$

$$q_i q_l^2 q_l^4 q_l a'_j \boxed{\hat{q}_i q_l^1 q_l^3 q_l^5 \hat{b} d'_j d''_j \boxed{a_j \check{\mathbf{a}}_\mathbf{j} \mathbf{b_j} d_j \#_2}} \Rightarrow^{\texttt{2r2}}$$

$$q_i q_l^2 q_l^4 q_l a'_j \boxed{\hat{q}_i \check{\mathbf{a}}_\mathbf{j} \mathbf{b_j} q_l^1 q_l^3 q_l^5 \hat{b} d'_j d''_j \boxed{a_j d_j \#_2}} \Rightarrow^{\texttt{1r7}}$$

$$q_i q_l^2 q_l^4 q_l \check{a}_j \mathbf{b_j a'_j} \boxed{\hat{q}_i q_l^1 q_l^3 q_l^5 \hat{b} d'_j d''_j \boxed{a_j d_j \#_2}} \Rightarrow^{\texttt{1r12}}$$

$$q_i q_l^2 q_l^4 q_l \check{a}_j \boxed{\hat{q}_i b_j a'_j q_l^1 q_l^3 q_l^5 \hat{b} d'_j d''_j \boxed{a_j d_j \#_2}}$$

Again there are two variants of computations, depending on the application of rule `1r19` or rule `2r9`. Notice that applying rule `1r19` leads to an infinite computation, as object b_j cannot be idle (rules `1r8`, `1r10`, `2r6`, `2r7`). Hence, we only consider the case of applying rule `2r9`:

$$q_i q_l^2 q_l^4 q_l \check{a}_j \boxed{\hat{q}_i \mathbf{b_j a'_j} q_l^1 q_l^3 q_l^5 \hat{b} d'_j d''_j \boxed{a_j d_j \#_2}} \Rightarrow^{\texttt{2r9}}$$

$$q_i q_l^2 q_l^4 q_l \check{a}_j \boxed{\hat{q}_i q_l^1 q_l^3 q_l^5 \hat{b} d'_j d''_j \boxed{a_j b_j \mathbf{a'_j d_j} \#_2}} \Rightarrow^{\texttt{2r10}}$$

$$q_i q_l^2 q_l^4 q_l \check{a}_j \boxed{\hat{q}_i a'_j q_l^1 q_l^3 q_l^5 \hat{b} d_j d'_j d''_j \boxed{a_j b_j \#_2}}$$

Now there are two variants of computations, depending on the application of rule `2r13` and `1r29`. It is easy to see that applying rule `2r14` leads to an infinite computation (rules `2r14`, `1r4`, `1r11`, `2r6`, `2r7`). Hence, consider applying rule `1r29`:

$$q_i q_l^2 q_l^4 q_l \check{a}_j \boxed{\hat{q}_i \mathbf{a'_j q_l^1} q_l^3 q_l^5 \hat{b} \mathbf{d_j d'_j} d''_j \boxed{a_j b_j \#_2}} \Rightarrow^{\texttt{1r29,1r19}}$$

$$q_i a'_j \mathbf{q_l^1 q_l^2} q_l^4 q_l \check{\mathbf{a}}_\mathbf{j} \mathbf{d_j} d'_j \boxed{\hat{q}_i q_l^3 q_l^5 \hat{b} d''_j \boxed{a_j b_j \#_2}} \Rightarrow^{\texttt{1r20,1r27}}$$

$$q_i a'_j q_l^4 q_l d'_j \boxed{\hat{q}_i q_l^1 \mathbf{q_l^2 q_l^3} q_l^5 \hat{b} \check{a}_j \mathbf{d_j d''_j} \boxed{a_j b_j \#_2}} \Rightarrow^{\texttt{1r21,2r13}}$$

$$q_i a'_j q_l^2 \mathbf{q_l^3 q_l^4} q_l d'_j \boxed{\hat{q}_i q_l^1 q_l^5 \hat{b} \check{a}_j \boxed{d_j \mathbf{d''_j a_j} b_j \#_2}} \Rightarrow^{\texttt{1r22,2r14}}$$

$$q_i a'_j q_l^2 q_l d'_j \boxed{\hat{q}_i q_l^1 q_l^3 \mathbf{q_l^4 q_l^5} d''_j \mathbf{a_j} \hat{\mathbf{b}} \check{a}_j \boxed{d_j b_j \#_2}} \Rightarrow^{\texttt{1r4,1r23}}$$

$$q_i a'_j q_l^2 q_l^4 \mathbf{q_l^5 q_l} a_j \hat{\mathbf{b}} \mathbf{d'_j} \boxed{\hat{q}_i q_l^1 q_l^3 d''_j \check{a}_j \boxed{d_j b_j \#_2}} \Rightarrow^{\texttt{1r24,1r30}}$$

$$q_i a'_j q_l^2 q_l^4 a_j \boxed{\hat{q}_i q_l^1 q_l^3 q_l^5 q_l \hat{b} d'_j d''_j \check{a}_j \boxed{d_j b_j \#_2}}$$

Thus, q_i is replaced by q_l in region 1.

Case b):

$$a_j\boxed{\mathbf{q_i}\hat{\mathbf{q}}_\mathbf{i}\check{a}_j\hat{b}\boxed{c_kb_jd_j\#_2}} \Rightarrow^{\texttt{1r1}} q_i\hat{\mathbf{q}}_\mathbf{i}\mathbf{a_j}\boxed{\check{a}_j\hat{b}\boxed{c_kb_jd_j\#_2}} \Rightarrow^{\texttt{1r2}}$$
$$q_i\boxed{\hat{q}_ia_j\check{a}_j\hat{b}\boxed{c_kb_jd_j\#_2}}$$

Again there are two variants of computations (depending on the application of rule `2r1` or rule `1r4`). It is easy to see that the application of rule `1r4` leads to infinite computation (by *"Circle"*). Consider the applying of rule `2r1`:

$$q_i\boxed{\hat{q}_i\mathbf{a_j}\check{\mathbf{a}}_\mathbf{j}\hat{b}\boxed{c_kb_jd_j\#_2}} \Rightarrow^{\texttt{2r1}} q_i\boxed{\hat{q}_i\hat{b}\boxed{\mathbf{c_k}\mathbf{a_j}\check{\mathbf{a}}_\mathbf{j}\mathbf{b_j}d_j\#_2}} \Rightarrow^{\texttt{2r2,2r3}}$$
$$q_i\boxed{\hat{q}_i\check{a}_jb_jc_ka_j\hat{b}\boxed{d_j\#_2}}$$

There are two variants of computations, depending on the application of rule `2r1` or rule `1r4`. Notice that they both lead to infinite computations. Indeed, if rule `2r1` will be applied, then rules `1r8`, `1r10`, `2r6`, `2r7` will be applied (applying rules `2r6`, `2r7` leads to an infinite computation). If rule `1r4` will be applied, it again leads to an infinite computation (rules `1r11`, `2r6`, `2r7`). Thus, we correctly model a "test for zero"-instruction.

3. END.

$$\begin{aligned}
R_{1,f} &= \{\texttt{1f1} : (\$^1\check{a}_j, out) \mid j \in I\} \\
&\cup \{\texttt{1f2} : (\$^2e_1, out), \texttt{1f3} : (\$^1\$^3, out)\} \\
&\cup \{\texttt{1f4} : (e_th_t, in) \mid t \in I'\} \\
&\cup \{\texttt{1f5} : (h_te_{t+1}, out) \mid 1 \le t \le n+3\} \\
R_{2,f} &= \{\texttt{2f1} : (q_f, in), \texttt{2f2} : (q_f\$^1, out), \texttt{2f3} : (q_f\$_2, out)\} \\
&\cup \{\texttt{2f4} : (\$^1\hat{a}, in), \texttt{2f5} : (\$^1\#_1, in), \texttt{2f6} : (\$^1I_c, in)\} \\
&\cup \{\texttt{2f7} : (h_{n+4}, in)\} \\
&\cup \{\texttt{2f8} : (h_{n+4}\hat{a}_j, out) \mid j \in I\} \\
&\cup \{\texttt{2f9} : (h_{n+4}b_j, out) \mid j \in I\} \\
&\cup \{\texttt{2f10} : (h_{n+4}d_j, out) \mid j \in I\}
\end{aligned}$$

At first, all objects $\check{a}_j$ will be moved to the environment and the objects $\hat{a}, \#_1, I_c$ to region 2 (thus, we stop without continuing the loop) and after that all objects $\hat{a}_j, b_j, d_j$ will be moved from region 2 to region 1. Hence, in region 2 now there are only the objects c_1 (representing the result of the computation) and the six additional objects $\#_1$, $\#_2$, $\hat{a}$, I_c, q_f, h_{n+4}. □

Both constructions from Theorem 2 and Theorem 3 can easily be modified to show that

$$PsOP_2(sym_1, anti_1)_T = PsRE \text{ and}$$
$$PsOP_2(sym_2)_T = PsRE,$$

i.e., the results proved in Theorem 2 and Theorem 3 can be extended from sets of natural numbers to sets of vectors of natural numbers.

5 Final Remarks

In this paper we have proved the new results that P systems with minimal cooperation, i.e., P systems with symport / antiport rules of size one, are computationally complete with only two membranes: they generate all recursively enumerable sets of vectors of nonnegative integers excluding (at most) the initial segment $\{0, 1, 2\}$. In an analogous manner, P systems with symport rules of size two are computationally complete with only two membranes: they generate all recursively enumerable sets of vectors of nonnegative integers excluding (at most) the initial segment $\{0, 1, 2, 3, 4, 5\}$. On the other hand it is known that systems with such rules in only one membrane cannot be universal, see [19, 32, 6]. Hence, the results we have proved in this paper are optimal with respect to the number of membranes. Notice that for *tissue* P systems with minimal cooperation this problem has already been solved successfully ([7]), i.e., it was proved that two cells are enough to generate all recursively enumerable sets of natural numbers.

Moreover, for P systems with symport rules of weight three we already obtain computational completeness with only one membrane modulo the initial segment $\{0, 1, 2, 3, 4, 5, 6\}$, which improves the result of [19], where thirteen objects remained in the skin membrane at the end of a halting computation.

As so far we have not been able to completely avoid additional symbols that remain after a computation has halted, the interesting open question

remains to find the minimal numbers of these additional objects that permit to obtain computationally completeness in the cases described above.

Acknowledgements

The first author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona. The first and the third authors acknowledge the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034 for providing a challenging and fruitful framework for cooperation. This article was written during the first author's stay at the Vienna University of Technology.

References

[1] A. Alhazov, R. Freund: P systems with one membrane and symport/antiport rules of five symbols are computationally complete. M.A. Gutierrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan (Eds.): *Proceedings of the Third Brainstorming Week on Membrane Computing*, Sevilla (Spain), January 31 – February 4 (2005), 19–28.

[2] A. Alhazov, R. Freund, M. Oswald: Tissue P Systems with Antiport Rules and a Small Number of Symbols and Cells. In: C. De Felice, A. Restivo (Eds.): Developments in Language Theory, 9th International Conference, DLT 2005, Palermo, Italy, July 4 – 8, 2005. *Lecture Notes in Computer Science* **3572** (2005), 100–111.

[3] A. Alhazov, R. Freund, M. Oswald: Symbol / Membrane Complexity of P Systems with Symport / Antiport Rules, *this volume*.

[4] A. Alhazov, R. Freund, Yu. Rogozhin: Some Optimal Results on Communicative P Systems with Minimal Cooperation. In: *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects)*, Sevilla, Spain, January 31 – February 2, 2005, 23–36.

[5] A. Alhazov, M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: Communicative P systems with minimal cooperation. In: G. Mauri et al. (Eds.): *Lecture Notes in Computer Science* **3365** (2005), 161–177.

[6] A.Alhazov, Yu.Rogozhin: Minimal Cooperation in Symport/Antiport P Systems with One membrane. In: M.A. Gutierrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan (Eds.): *Proceedings of the Third Brainstorming Week on Membrane Computing*, Sevilla (Spain), January 31 – February 4, 2005, 29–34.

[7] A. Alhazov, Yu. Rogozhin, S. Verlan: Symport/antiport tissue P systems with minimal cooperation. In: *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects)*, Sevilla, Spain, January 31 – February 2, 2005, 37 – 52.

[8] F. Bernardini, M. Gheorghe: On the power of minimal symport/antiport. In: A. Alhazov, C. Martín-Vide, Gh. Păun (Eds.): *Workshop on Membrane Computing, WMC-2003*, Tarragona, July 17–22, 2003, Technical Report N. 28/03, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona (2003) 72–83.

[9] F. Bernardini, A. Păun: Universality of minimal symport/antiport: five membranes suffice. In: C. Martin-Vide et al. (Eds.), WMC 2003, *Lecture Notes in Computer Science* **2933** (2004) 43–45.

[10] C.S. Calude, Gh. Păun: Bio-steps beyond Turing, *CDMTCS research report* **226**, Auckland Univ. (2003).

[11] R. Freund, M. Oswald: GP systems with forbidding context. *Fundamenta Informaticae* **49**, 1–3 (2002) 81–102.

[12] R. Freund, M. Oswald: P Systems with Activated/Prohibited Membrane Channels. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19–23, 2002. Revised Papers. *Lecture Notes in Computer Science* **2597** (2003), 261–268.

[13] R. Freund, M. Oswald: A short note on analysing P systems with antiport rules. *Bulletin of the European Association for Theoretical Computer Science* **78** (2002), 231–236.

[14] 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, January 31 – February 2, 2005, 187–200.

[15] R. Freund, A. Păun: Membrane systems with symport/antiport: universality results. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19–23, 2002. Revised Papers. *Lecture Notes in Computer Science* **2597** (2003), 270–287.

[16] R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel states. In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini (Eds): *Second Brainstorming Week on Membrane Computing*. Technical report of Research Group on Natural Computing, University of Seville, TR 01/2004 (2004), 206–223 and *Theoretical Computer Science* **330** (2005), 101–116.

[17] R. Freund, Gh. Păun: On Deterministic P Systems, submitted (2003).

[18] P. Frisco: About P systems with symport/antiport. In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini (Eds): *Second Brainstorming Week on Membrane Computing*. Technical report of Research Group on Natural Computing, University of Seville, TR 01/2004 (2004), 224–236.

[19] P. Frisco, H.J. Hoogeboom: P systems with symport/antiport simulating counter automata. *Acta Informatica* **41**, 2–3 (2004), 145–170.

[20] P. Frisco, H.J. Hoogeboom: Simulating counter automata by P systems with symport/antiport. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing International Workshop, WMC-CdeA 02, Curtea de Arges, Romania, August 19–23, 2002. Revised Papers. *Lecture Notes in Computer Science* **2597** (2003), 288–301.

[21] L. Kari, C. Martín-Vide, A.Păun: On the universality of P systems with minimal symport/antiport Rules. *Lecture Notes in Computer Science* **2950** (2004), 254–265.

[22] M. Margenstern, V. Rogozhin, Yu. Rogozhin, S. Verlan: About P systems with minimal symport/antiport rules and four membranes. In: G. Mauri, Gh. Păun, C. Zandron (Eds.): *Pre-Proceedings of Fifth Workshop on Membrane Computing (WMC5)*, Universitá di Milano-Bicocca, Italy, June 14–16, (2004) 283–294.

[23] C. Martín-Vide, A. Păun, Gh. Păun: On the power of P systems with symport rules, *Journal of Universal Computer Science* **8**, 2 (2002), 317–331.

[24] C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón: Tissue P systems. *Theoretical Computer Science* **296** (2) (2003), 295–326.

[25] M.L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey (1967).

[26] Gh. Păun: Computing with Membranes. *Journal of Computer and Systems Science* **61** (2000), 108–143.

[27] A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing* **20** (2002), 295–305.

[28] Gh. Păun: *Membrane computing. An Introduction.* Springer-Verlag (2002).

[29] Gh. Păun, J. Pazos, M.J. Perez-Jimenez, A. Rodriguez-Paton: Symport/antiport P systems with three objects are universal. *Fundamenta Informaticae* **64**, 1–4 (2005).

[30] G. Rozenberg, A. Salomaa (Eds.): Handbook of Formal Languages (3 volumes). Springer-Verlag, Berlin (1997).

[31] Gy. Vaszil: On the size of P systems with minimal symport/antiport. In: G. Mauri, Gh. Păun, C. Zandron (Eds.): *Pre-Proceedings of Fifth Workshop on Membrane Computing (WMC5)*, Universitá di Milano-Bicocca, Italy, June 14 - 16, 2004, 422–431.

[32] S. Verlan: Optimal results on tissue P systems with minimal symport/antiport. Presented at *EMCC meeting*, Lorentz Center, Leiden, The Netherlands, 22–26 November, 2004.

[33] S. Verlan: Tissue P systems with minimal symport/antiport. In: C.S. Calude, E. Calude, M.J. Dinneen (Eds): Developments in Language Theory, DLT 2004. *Lecture Notes in Computer Science* **3340**, Springer-Verlag, Berlin (2004), 418–430.

On Evolutionary Lineages of Membrane Systems

Petr SOSÍK[1,2], **Ondřej VALÍK**[2]

[1]Facultad de Informática, Universidad Politécnica de Madrid
Campus de Montegancedo s/n, Boadilla del Monte
28660 Madrid, Spain

[2]Institute of Computer Science, Silesian University,
74601 Opava, Czech Republic
E-Mail: {petr.sosik,ondrej.valik}@fpf.slu.cz

Abstract

We introduce a simple model of P system motivated by certain restrictions found in biological systems. Its computational power is rather limited and corresponds to that of a finite transducer. An important characteristics of the model is its interactive behavior. Then we study a computational power of evolutionary lineages of such P systems. Referring to known results from the structural complexity theory (Karp and Lipton, Wiedermann and van Leeuwen), we show that a super-Turing computational potential can emerge in non-uniform lineages of these restricted P systems.

Furthermore, key features of our model are related to lineages of biological systems. Hence our results provide another argument supporting thesis from [14] and others that a super-Turing potential is *naturally and inherently present* in evolution of living organisms.

1 Introduction

Membrane systems (recently called P systems) are biologically inspired (particularly, cell inspired) formal computational models introduced in [8]. For an overview of membrane computing theory we refer the reader to [9]. Computational operations in P systems are motivated by some properties of living cells which are mathematically abstracted and generalized. Many of the recently studied variants of P systems achieve universal (in Turing sense)

computational power, provided that their membranes can contain an unlimited number of objects.

In this paper, however, we try to get a bit closer relation of our results to the computational potential of biological cells and their lineages and, too, to the potential of a physically feasible "wet computer". Therefore we adopt the assumption that *each living cell or a multicellular organism is a finite body and its behavior can in principle be modelled by a finite-size model, however complicated.* A similar approach can be found also in the recently developed model of *P colonies* [7]. One can argue that organisms can use their (potentially infinite) environment as an external memory, but even then there is a barrier of their limited length of life during which only a limited portion of "memory" can be used.

An important part of our model are certain properties of living organisms which, due to [12, 14], give them power beyond level of classical formal automata:

(i) *interactivity*, i.e. continuous flow of information, contrasting with a beforehand-given input of conventional automata;

(ii) *unpredictability* of interactions and information exchange (notice the difference from a nondeterministic Turing machine where all possible configurations can be found and simulated by its deterministic variant);

(iii) *continuous lineage of individuals*, transferring information from one generation to another, and capable of continuous changes due to intensive interactions with their environment.

Also nowadays computers connected into a worldwide network possess exactly the same properties (i)–(iii). Indeed, they intensively interact in an unpredictable manner (often too unpredictable to our taste). When solving difficult problems, they can consult other network machines. They get continuous upgrade through the network, often without our knowledge. Last, but not least, there exists a continuity between their generations. Therefore, the original paradigm of a computer as a Turing machine does not correspond to the recent situation and maybe the time has come for its change [13].

Based on this argumentation, we incorporate the properties (i)–(iii) into our P system model and study its power. Rather surprisingly, we show that the resulting lineage of simple P systems reaches a *super-Turing computational potential.*

The history of super-Turing computation originates, paradoxically, in the Turing's dissertation thesis [11]. Here the modified variant of the Turing machine augmented with the *oracle device* has been introduced and its power studied. The machine has a special *oracle tape* and a corresponding query state. The machine can write a question at the oracle tape and enter the query state. Then in one step, an (in principle uncomputable) answer appears at the oracle tape. There have been also considered limitations on the size of the oracle's answer. In 1980 there was defined a bit weaker *advice Turing machine* [6]. The advice differs from the oracle in the following: while the oracle assigns an individual answer to each query, the advice provides an answer due to the *length* of the query and ignores its content. Therefore, all queries with the same length are given the same answer. Usually, length of the advice is polynomially restricted w.r.t. the length of the query. We denote the advice Turing machine by TM/A. We refer the reader e.g. to [1] for more information on non-uniform complexity classes.

In 1980's and 1990's many super-Turing models were studied, as Interaction Machines, Site and Internet Machines, π-Kalkl, \$-Kalkl etc. The reader is referred to [3] or [4] for an overview. We mention two super-Turing computational models with biological inspiration. In [2], authors show a super-Turing power of deterministic P systems which can speed up their operations gradually (a reversed Achilles and Tortoise principle). Another paper [15] introduces the model called *bacterioid* which combines computational and non-computational mechanisms. The bacterioid conforms with the requirements for a minimal artificial life and exhibits also rudimentary cognitive properties.

In this context we argue together with [12, 14] and others that super-Turing potential can naturally emerge in evolutionary lineages of finite (biological or other) systems. Indeed, the requirements (i)–(iii) form elementary components of biological evolutionary processes. Together with the finiteness of living organisms we obtain a reasonable scenario suggesting that a super-Turing potential is not only possible, but probably rather necessary phenomenon during biological evolution.

2 Interactive Finite Machines

In this section we focus on the first natural property of the living entities – *interactivity*. We fix some basic notation first. An *alphabet* Σ is a finite and nonempty set of symbols. The set of all words over Σ is denoted by Σ^*. This set includes the *empty word* ϵ. The set of all nonempty words $\Sigma^* \setminus \{\epsilon\}$ is

denoted by Σ^+. The length of a word w is denoted by $|w|$. For a nonnegative integer n and a word w, we use w^n to denote the word that consists of n concatenated copies of w. For more information on formal language theory we refer the reader to [10].

Now we present a canonical model of an interactive finite machine – the *interactive finite transducer (IFT)*, introduced in [12], which is an analogy of Mealy automaton. Recall that Mealy automaton is a finite-state machine which at each computational step inputs and outputs symbols from a given alphabet. Of course, when considering suitable computational models for embodiment of the lineages of evolving organisms, we may think about finite-state neural networks or other models with evolutionary capabilities. However, as they are in general finite-state machines, IFT's appear to be a proper choice for study of their computational limits due to its simplicity.

Definition 2.1 *A* Mealy automaton *is a six-tuple* $\mathcal{M} = (I, O, Q, \delta, \rho, q_0)$, *where*

- I *is an input alphabet,*
- O *is an output alphabet,*
- Q *is a finite nonempty set of states,*
- $\delta : Q \times I \longrightarrow Q$ *is a transition function,*
- $\rho : Q \times I \longrightarrow O$ *is an output function,*
- $q_0 \in Q$ *is an initial state.*

The input of the Mealy automaton is an input tape containing a finite word over the input alphabet. At each step, the Mealy automaton reads a symbol from the input tape, changes eventually its state due to δ and outputs a symbol from the output alphabet selected by ρ.

IFT's differ from Mealy automata mainly in their computational scenario. Unlike a Mealy automaton, an IFT inputs a (potentially infinite) input stream over and input/output alphabet Σ via its input channel and outputs a (potentially infinite) output stream. Therefore we do not assume that the whole input is fixed and written in a tape beforehand. Let Σ^ω denote the *set of all infinite streams* over the alphabet Σ. Hence an IFT realizes a *translation* $\phi : \Sigma^\omega \longrightarrow \Sigma^\omega$. In the sequel, however, we use Mealy automata as a formal representation of IFT's.

2.1 Interactive Finite P Systems

Membrane computing offers a computational framework in which the assumptions (i)–(iii) from the section 1 can be naturally implemented. We build on a "classical" variant of P system computing with multisets of objects. However, we impose some restrictions motivated by reflections in Section 1. Also, communication with the outer environment is performed via an *input* and an *output* channel similarly as in P automata with communication rules [5]. At each step our model can receive at most one symbol via its input channel and send at most one symbol via its output channel. Of course, living organisms receive at each moment an n-tuple of "inputs" via their "input channels", but let us assume that n is limited from above and hence the the set of all such n-tuples can be mapped one-to-one into a finite alphabet. The resulting model is called *Interactive Finite P System (IFPS).*

Definition 2.2 *An* interactive finite P system *of degree n, $n \geq 1$, is the construct*

$$\Pi = (\Sigma, \Gamma, \sqcup, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n),$$

where:

(i) Σ *is an alphabet; its elements are called objects;*

(ii) $\Gamma \subset \Sigma$ *is an input/output alphabet;*

(iii) $\sqcup \in \Gamma$ *is a special symbol denoting an undefined input/output;*

(iv) μ *is a hierarchical structure of n membranes, with membranes denoted by integers $1, \ldots, n$; the outermost membrane is called also the* skin *membrane;*

(v) w_i, $1 \leq i \leq n$, *is an initial content of region i of the membrane structure μ; formally, w_i is a word in Σ^* representing by its Parikh vector a multiset $m(w_i)$ over Σ;*

(vi) R_i, $1 \leq i \leq n$, *is a finite set of evolutional rules $u \to v$ over Σ belonging to region i of the structure μ; forms of the rules can be:*

(a) $a \to b_\tau$, $a, b \in \Sigma$, $\tau \in (\{here\} \cup \{in_j \mid 1 \leq j \leq n\})$; *if $[_i]_i$ is not the skin membrane, then it is also allowed that $\tau = out$;*

(b) $ab \to a_{\tau_1} c_{\tau_2}$, $a, b, c \in \Sigma$, $\tau_1, \tau_2 \in (\{here\} \cup \{in_j \mid 1 \leq j \leq n\})$; *if $[_i]_i$ is not the skin membrane, then it is also allowed that $\tau_1 = out$ or $\tau_2 = out$ or both;*

(c) $ab \rightarrow a_\tau b_{out} c_{come}$, $a \in \Sigma$, $b, c \in \Gamma$, $\tau \in (\{here\} \cup \{in_j \mid 1 \leq j \leq n\})$*; these rules can exist only within the skin membrane.*

The components $w_1, \ldots, w_n$ form an initial state of Π. Generally, each n-tuple $w'_1, \ldots, w'_n$ is called a *configuration* of Π. For two configurations $C_1 = (w'_1, \ldots, w'_n)$ and $C_2 = (w''_1, \ldots, w''_n)$ we write $C_1 \Longrightarrow C_2$, and we say that we have a *transition* from C_1 to C_2, if we can pass from C_1 to C_2 by using the evolution rules appearing in $R_1, \ldots, R_n$ in the following manner:

- If an object appears in v in a form a_{here}, then it will remain in the same region i (instead of a_{here} we often write simply a).
- If an object appears in v in a form a_{out}, then a will exit the membrane i and will become an element of the region immediately outside it; or, in case of (c)-type rules, it is sent out of the system via its output channel.
- If an object appears in a form a_{in_q}, then a will be added to the multiset $m(w'_q)$, provided that a is adjacent to the membrane q.
- If an object appears in a form a_{come}, then a is imported into the skin membrane via input channel. *At each step, only one rule of type (c) can be applied.*

All these operations are done in parallel, for all possible applicable rules $u \rightarrow v$, for all occurrences of multisets u in the region associated with the rules, for all regions at the same time. The system continues these parallel steps until there remain any applicable rules in any compartment of Π. Both an input and an output of the system are infinite streams of symbols in Γ^ω.

In this paper we restrict ourselves to *deterministic* IFPS's, which at each configuration C_1 and for each symbol in the input channel can pass to at most one possible configuration C_2 and sent at most one possible symbol to its output channel.

Definition 2.3 *A translation mapping $\phi : \Sigma^\omega \longrightarrow \Sigma^\omega$ is called an* interactive translation *realized by a deterministic IFPS Π if the following holds: $\phi(x) = y$ iff Π with the input x never halts and outputs y, for $x, y \in \Sigma^\omega$.*

It follows by the above definition that those IFPS's which halt after a finite number of steps, as well as those which input/output only a finite number of symbols, do not realize an interactive translation.

Example 2.4 The following IFPS Π_{ab} searches the input stream for strings ab, see Fig. 1. Its response to such a string is the output #. In other cases the system just copies an input string to the output.

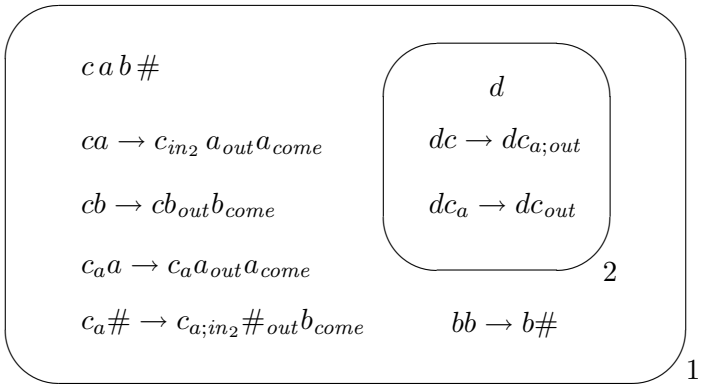


Figure 1: The interactive finite P system Π_{ab}.

Let $\Pi_{ab} = (\Sigma, \Gamma, \sqcup, \mu, w_1, w_2, R_1, R_2)$ be an IFPS of degree 2, where

$$
\begin{aligned}
\Sigma &= \{a, b, c, c_a, d, \sqcup, \#\} \\
\Gamma &= \{\sqcup, a, b, \#\} \\
\mu &= [_1[_2]_2]_1 \\
w_1 &= cab\# \\
w_2 &= d \\
R_1 &= \{r_1 : ca \to c_{in_2} a_{out} a_{come}, \\
&\quad\ r_2 : cb \to cb_{out} b_{come}, \\
&\quad\ r_3 : c_a a \to c_a a_{out} a_{come}, \\
&\quad\ r_4 : c_a \# \to c_{a;in_2} \#_{out} b_{come}, \\
&\quad\ r_5 : bb \to b\#\} \\
R_2 &= \{r_6 : dc_a \to dc_{out}, r_7 : dc \to dc_{a;out}\}
\end{aligned}
$$

The system Π_{ab} works as follows:

1. In the initial configuration only the rules r_1 or r_2 are applicable. If the input is:

   (a) then r_1 is applied, the object a is copied to the output and c is sent to region 2; we continue by step 2;
   (b) then r_2 is applied, b is sent to the output and we continue in the same configuration.

2. In region 2 the rule r_7 is now applicable, rewriting c to c_a and sending it to region 1.

3. Now in region 1 the rules r_3 or r_4 are applicable. If the input is:

   (a) then r_3 is applied, copying a to the output without a change of the recent configuration;
   (b) then r_4 is applied, sending # to the output as an indicator of the input ab, sending simultaneously c_a into region 2, and we continue by step 4.

4. Rules r_5 and r_6 are simultaneously applied, turning the system back into the initial configuration, and we continue by step 1.

2.2 Equivalence of IFT and IFPS

We show that interactive P systems compute exactly the same translation functions as IFT's. Our result extends Theorem 1 in [14] which states the equivalence of IFT with several other computational models as discrete neural nets or combinatorial circuits.

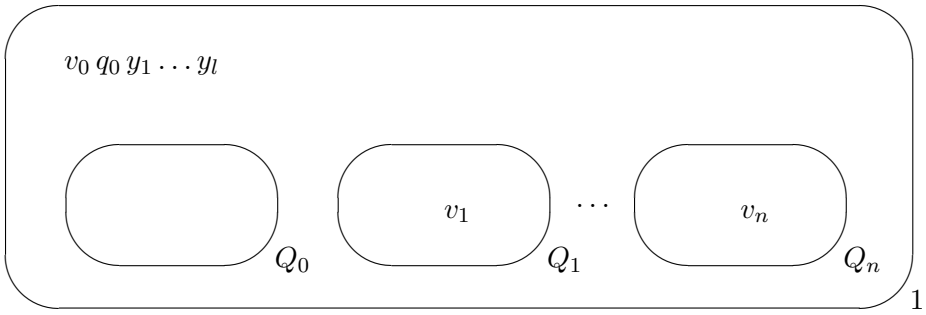


Figure 2: The interactive P system simulating IFT.

Theorem 2.5 *For a translation $\phi : \Sigma^\omega \to \Sigma^\omega$ the following is equivalent:*

(i) ϕ is realized by an interactive finite transducer;

(ii) ϕ is realized by a deterministic interactive finite P system.

Proof. (i)⇒(ii) Let $\mathcal{M} = (I, O, Q, \delta, \sqcup, q_0)$ be a Mealy automaton, where $I = \{x_1, \ldots, x_k\}$, $O = \{y_1, \ldots, y_l\}$ and $Q = \{q_0, \ldots, q_n\}$.

We construct a deterministic interactive P system $\Pi_{\mathcal{M}} = (\Sigma, \Gamma, \sqcup, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$ of degree $m = n + 2$ realizing the same translation.

- $\Gamma = \{\sqcup, x_1, \ldots, x_k, y_1, \ldots, y_l\}$
- $\Sigma = \{q_0, \ldots, q_n, p_0, \ldots, p_n\} \cup \Gamma$
- $\mu = [_1[_{Q_0}]_{Q_0} \ldots [_{Q_n}]_{Q_n}]_1$
- $w_1 = q_0 p_0 y_1 \ldots y_l$, $w_{Q_0} = \epsilon$ and $w_{Q_i} = p_i$, $1 \leq i \leq n$
- R_1 is constructed as follows: for each pair of rules $\delta(q_i, x) = q_j$, $\rho(q_i, x) = y$, $q_i, q_j \in Q$, $x \in I$, $y \in O$, we add to R_1 the rules:

$$\{p_i y \rightarrow p_{i;in_{Q_i}} y_{out} x_{come}\} \quad (1)$$
$$\{q_i x \rightarrow q_{i;in_{Q_j}} y\} \quad (2)$$

- $R_{Q_i} = \{p_i p \rightarrow p_{i;out} q_{i;out} \mid \exists\, p \in Q, x \in I : \delta(p, x) = q_i\}$, $0 \leq i \leq n$.

Each step of the automaton $\mathcal{M}$ is simulated by the P system $\Pi_{\mathcal{M}}$ as follows.

1. *Input and output of a symbol:* presence of an object p_i, $0 \leq i \leq n$, in the skin membrane represents the state q_i of $\mathcal{M}$. An application of a rule $\rho(q_i, x) = y$ of $\mathcal{M}$ is simulated by a rule of type (1). Notice that a complete set of output objects $y_1, \ldots, y_l$ is present within the skin membrane.
2. *State transition – phase I:* A rule of type (2) completes the set of output objects within the skin membrane. Simultaneously it sends the object q_i into the membrane Q_j which is equivalent to the rule $\delta(q_i, x) = q_j$ of $\mathcal{M}$.
3. *State transition – phase II:* Now the membrane Q_j contains objects p_j and q_i. The object q_i is rewritten to q_j and sent to the skin membrane together with p_j to represent the new state q_j of $\mathcal{M}$.

It follows by the above description that the deterministic IFPS $\Pi_{\mathcal{M}}$ realizes the same translation as the IFT $\mathcal{M}$.

(i)⇐(ii) Let $\Pi = (\Sigma, \Gamma, \sqcup, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$ be a deterministic IFPS which realizes a translation ϕ. Let us denote by $\mathcal{C}_\Pi$ the set of all configurations of Π. The number of all possible configurations is determined by the membrane structure μ, the size of the alphabet $l = |\Sigma|$, and the initial number of objects within the system, $m = |w_1| + |w_2| + \ldots + |w_n|$. Observe that the number of objects does not change during the work of the system.

The reader can verify that the system has $|\mathcal{C}_\Pi| = (m+ln-1)!/(m!(ln-1)!)$ possible configurations.

Then the IFPS Π can be simulated by a Mealy automaton $\mathcal{M}_\Pi = (\Gamma, \Gamma, Q, \delta, \rho, C_0)$, where:

- $Q = \{C_0\} \cup \{C \in \mathcal{C} \mid C_0 \Rightarrow^* C' \Rightarrow_r C,\ r = ab \rightarrow a_\tau b_{out} c_{come}\}$,
- $\delta(C, c) = C'$ and $\rho(C, c) = b$, $b, c \in \Gamma$, if and only if $C \Rightarrow^+_\Pi C'$, and this sequence of transitions involves exactly one application of a rule $ab \rightarrow a_\tau b_{out} c_{come}$ in its last step, for some $a \in \Sigma$, $\tau \in (\{here\} \cup \{in_j \mid 1 \leq j \leq n\})$.

Due to the determinism of Π, it is guaranteed that the sequence of transitions $C \Rightarrow^+_\Pi C'$ is unique for a given $c \in \Gamma$. The above description shows that the translation ϕ realized by Π is identical with the translation realized by $\mathcal{M}_\Pi$. □

3 From Machines to Lineages

In this section we consider the other two mentioned properties of communities of living organisms: continuous lineages of unpredictably evolving individuals. Similarly as in the previous section, we present first a canonical model of lineage [12] based on theory of finite automata. We denote by $\mathcal{U}$ a universe of possible states of all automata in the lineage.

Definition 3.1 *Let $\mathcal{A} = \{A_1, A_2, \ldots\}$ be a sequence of IFT's over an input/output alphabet Σ and let $Q_i \subseteq \mathcal{U}$ be a set of states of A_i. Let $G = \{G_1, G_2, \ldots\}$ be a sequence of states from $\mathcal{U}$ such that $G_i \subset Q_i$ and $G_i \subseteq G_{i+1}$, $i \geq 1$. Then $\mathcal{A}$ together with G is called a* sequence of IFT's with global states.

The sequence $\mathcal{A}$ is *non-uniform*, i.e. there is no algorithmic way how to describe its members. The only way how to define the sequence is to list all its members. The set $\bigcup_i G_i \subseteq \mathcal{U}$ is called the *set of global states* of $\mathcal{A}$. The sequence $\mathcal{A}$ processes an infinite input stream from Σ^ω as follows. At the beginning, the automaton A_1 processes the input stream using its local states $Q_1 - G_1$. At a certain moment A_1 enters a global state $g \in G_1$, finishes its computation and passes the control to A_2. The input stream is redirected to the input of A_2 which starts its computation in the same state $g \in G_2$ and processes another symbol. After a certain number of steps in

its local states, A_2 enters a global state g', passes control to A_3 et cetera. Although it is not explicitly mentioned in the definition, it is assumed that the number of states of automata $\{A_1, A_2, \ldots\}$ increases, although possibly non-monotonically (unlike the monotonic sequence $G_1 \subseteq G_2 \subseteq G_3 \subseteq \ldots$).

Therefore, the input stream is processed by automata with an increasing index i. The next active automaton represents a new generation with potentially richer configuration space. This mechanism allows for a transfer and improvement of structural information from the previous generation. These improvements are understood as a result of unpredictable interactions of an individual (transducer) with its environment and other individuals.

3.1 Computational Potential of Lineages of IFT's

We use an *interactive advice Turing machine* to characterize the computational power of non-uniform lineages of IFT's. The interactive Turing machine (ITM) is – similarly as an IFT – a computational device working over infinite input and output streams. Unlike IFT, however, it has an internal architecture of Turing machine with an infinite tape and therefore its configuration space is infinite. Besides tape operations in spirit of Turing machine, ITM at each step reads a symbol from its input channel and sends a symbol to its output channel. Moreover, after receiving a *nonempty* symbol from its input channel, the ITM is required to send a *nonempty* symbol to its output channel within a finite number of steps. In this way the ITM realizes an interactive translation $\phi : \Sigma^\omega \longrightarrow \Sigma^\omega$. We refer the reader to [12] for more details.

The computational power of ITM is in principle equivalent to that of a standard Turing machine. Indeed, an input/output of a standard TM can be a part of input/output streams of ITM, on one hand. On the other hand, each translation of ITM $\mathrm{Pref}(x) \longrightarrow \mathrm{Pref}(y)$ for finite prefixes of x and y of the same length is Turing-computable. The ingredient we add to ITM to increase its power is the *advice function* introduced in Section 1.

Definition 3.2 *An* advice *is a function* $f : \mathbb{N}^+ \longrightarrow \{0,1\}^*$. *We say that an advice* f *is* $S(m)$-bounded *if* $|f(m)| \leq S(m)$ *for each* $m \in \mathbb{N}$.

The resulting device is called *interactive advice Turing machine* (ITM/A). See [13] for motivation and more results about ITM/A. An ITM/A can in a step t ask only a query of length $t_1 \leq t$. To get an advice, ITM/A is is equipped with a special *advice tape* and an *advice state*. When ITM/A writes an argument t_1 to the advice tape and enters the advice state, the value of $f(t_1)$ rewrites in one step the original content of the

advice tape. Due to the possible non-computability of the advice, ITM/A is a super-Turing computational device [12]. The following result can be found in [14].

Theorem 3.3 *A translation $\phi : \Sigma^\omega \to \Sigma^\omega$ can be realized by a sequence of IFT's with global states iff it can be realized by an ITM/A.*

3.2 Lineages of IFPS's

Theorem 3.3 can be naturally extended to sequences of interactive finite P systems. Consider an IFPS

$$\Pi = (\Sigma, \Gamma, \sqcup, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m)$$

with a configuration $C = (w_1, \ldots, w_m)$. A *state* of Π is a pair (μ', C'), where C' is obtained from C by omitting all empty strings ϵ corresponding to membranes containing no objects, and μ' is obtained from μ by omitting these membranes. The *universe of states* $\mathcal{U}$ is the set of all possible states (including all possible membrane structures).

Consider further a sequence of IFPS's $\mathcal{P} = \{\Pi_1, \Pi_2, ...\}$ such that each Π_i is assigned a finite set of states $Q_i \subseteq \mathcal{U}$ determined by its structure and possible contents of its membranes. Some selected states form a set $G_i \subset Q_i$ of global states. The sequence $\mathcal{P}$ must satisfy the conditions of Definition 3.1: $G_i \subseteq G_{i+1}$, $i \geq 1$. Then $\mathcal{P}$ is called an *evolutionary sequence of interactive P systems.*

Example 3.4 Let $\mathcal{P} = \{\Pi_1, \Pi_2, ...\}$ be a sequence of IFPS's. Denote by Q_i the set of states of Π_i, $i \geq 1$. Let $\Delta_i \subseteq \Gamma_i$ be a nonempty alphabet of *global symbols*, where Γ_i is the alphabet of Π_i. Let further $\Delta_i \subseteq \Delta_{i+1}$, $i \geq 1$. Let *global states* of Π_i be those of its states which contain a symbol from Δ_i.

A transition from Π_i to Π_{i+1} is realized by its mutation, during which:

- a rule can be added/deleted/replaced,
- a symbol can be added to the system's alphabet,
- an empty membrane together with rules can be added.

When Π_i enters a global state, it is changed to Π_{i+1} which starts from the same state. Then Π_{i+1} operates over input/output streams until it enters again a global state. This can happen even in its first step if all global symbols are not removed during this step.

Hence the sequence of IFPS's $\mathcal{P} = \Pi_1, \Pi_2, ...$ satisfies the conditions of Definition 3.1 and we have the following result:

Theorem 3.5 *A translation $\phi : \Sigma^\omega \to \Sigma^\omega$ can be realized by a sequence of IFT's with global states iff it can be realized by an evolutionary sequence of IFPS's.*

Corollary 3.6 *A translation $\phi : \Sigma^\omega \to \Sigma^\omega$ can be realized by an evolutionary sequence of IFPS's iff it can be realized by an ITM/A.*

Note that all members of the sequence $\mathcal{P}$ operate with the same number of objects. The evolution changes only their alphabet, membrane structure and rules.

4 Conclusion

We study a simple variant of P system called the *interactive finite P system* – IFPS. An IFPS can at each step contain only a fixed, pre-defined number of objects and a fixed number of membranes, therefore its configuration space is finite. It communicates with an outer environment via an input and an output channel. The key ingredient increasing its power is the capability of evolutionary lineages of IFPS's to evolve from one generation to another in an unpredictable, non-computable manner.

We have shown that evolutionary lineages of IFPS's reach the super-Turing computational potential. This result extends work of [12, 14, 15] and others where authors study power of lineages of finite-state machines. Our biologically inspired model of IFPS, however, is restricted to use elementary cell-like computational operations. We have therefore settled the open question in [15] how to implement such lineages within the framework of P systems.

One might ask whether now we are able to solve some concrete, a priori given undecidable problems with lineages of IFPS's? On one hand, it has been shown in [12] that ITM/A (and in turn also lineages of IFPS's) is *strictly more powerful* than ITM (and hence than standard Turing machine). In other words, some undecidable problems (e.g. the Halting Problem) can in principle be solved by lineages of IFPS's in a finite number of steps. On the other hand, computational evolutionary processes are by definition of an interactive and unpredictable nature and can not be simulated by an equivalent deterministic device in a finite number of steps. (Unlike a non-deterministic TM which can be simulated by a deterministic TM with an

exponential slowdown.) Therefore, one can not solve non-computable problems "on command" with IFPS's. Which problems will be solved and when depends on the evolutionary process. What one can do is to increase chances for finding answers by providing a "rich and inspirative" evolutionary environment.

There remain many other open questions. For instance, we imposed a few restrictions on the form of evolution of IFPS's. However, in [12] authors show that polynomially bounded lineages of IFT's are computationally equivalent to logarithmic space-bounded ITM's with a polynomially bounded advice. Hence the complexity problems of lineages of IFPS's are subject of further research.

Similar open problems exist for *uniform* lineages of IFPS's with various evolutionary restrictions. We conjecture that NP-complete or PSPACE-complete problems can be solvable in polynomial time by certain uniform lineages of IFPS's. Interesting question is whether similar results can be obtained by even simpler membrane computing models, as recently introduced *P colonies* [7]. Communities of IFPS's or P colonies can not only reach the computational power exceeding the limit of each of its members [7], but it might also be useful for modelling complex social behavior of living cells, e.g. bacteria.

Acknowledgements

Research was supported by the Silesian University Science Foundation, grant No. 26/2005, and by the Czech Science Foundation, grant No. 201/04/0528. Authors are obliged to A. Alhazov, G. Păun and J. Wiedermann for comments improving the paper.

References

[1] J.L. Balcazar, J. Diaz, J. Gabarro: *Structural Complexity I, Second Edition.* Springer-Verlag, Berlin (1995).

[2] C.S. Calude, G. Păun: Bio-steps beyond Turing. *Biosystems* **77** (1–3) (2004), 175–194.

[3] B.J. Copeland (Ed.): *Minds and Machines* **12** (4) (2002) and **13** (1) (2003).

[4] E. Eberbach, P. Wegner: Beyond Turing machines. *Bulletin of the EATCS* **81** (2003), 279–304.

[5] R. Freund, M. Oswald, L. Staiger: ω-P automata with communication rules. In: C. Martín-Vide et al (Eds.): Workshop on Membrane Computing WMC 2003. *Lecture Notes in Computer Science* **2933**, Springer-Verlag, Berlin (2004), 203–217.

[6] R.M. Karp, R.J. Lipton: Some connections between nonuniform and uniform complexity classes. In: *Proc. 12th Annual ACM Symposium on the Theory of Computing* (STOC '80) (1980), 302–309.

[7] J. Kelemen, A. Kelemenova, Gh. Paun: P colonies. In: M. Bedan et al (Eds.): *Workshop on Artificial Chemistry*, ALIFE9. Boston, USA (2004), 82–86.

[8] G. Păun, Computing with Membranes, J*ournal of Computer and System Sciences* **61** (1) (2000), 108–143.

[9] G. Păun, *Membrane Computing: an Introduction.* Springer-Verlag, Berlin (2002).

[10] G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages.* Springer-Verlag, Berlin (1997).

[11] A.M. Turing: Systems of logic based on the ordinals. *Proceedings of the London Mathematical Society* **45** (1939), 161–228.

[12] J. van Leeuwen, J. Wiedermann: Beyond the Turing limit: evolving interactive systems. In: L. Pacholski, P. Rùžièka (Eds.): SOFSEM'01: Theory and Practice of Informatics. *Lecture Notes in Computer Science* **2234**, Springer, Berlin (2001), 90–109.

[13] J. van Leeuwen, J. Wiedermann: The Turing machine paradigm in contemporary computing. In: B. Enquist, W. Schmidt (Eds.): *Mathematics Unlimited – 2001 and Beyond.* Springer, Berlin (2001), 1139–1155.

[14] J. Wiedermann, J. van Leeuwen: The emergent computational potential of evolving artificial living systems. *AI Communications* **15** (4) (2002), 205–216.

[15] J. Wiedermann: Coupling computational and non-computational processes: minimal artificial life. In: G. Mauri, Gh. Paun, C. Zandron (Eds.): *Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5).* University of Milan – Bicocca (2004), 432–445.

Part II

Regular Contributions

Multithread Java P Systems Running on a Cluster of Computers

Manuel ALFONSECA,
Carlos CASTAÑEDA MARROQUÍN,
Marina DE LA CRUZ ECHEANDÍA,
Rafael NÚÑEZ HERVÁS,
Alfonso ORTEGA DE LA PUENTE

Departamento de Ingeniería Informática
Universidad Autónoma de Madrid
E-Mail: {manuel.alfonseca,carlos.castanneda,
marina.cruz,r.nunnez,alfonso.ortega}@uam.es

1 Introduction

P Systems are the basic model in Membrane Computing, and were proposed in the late 90s by Gh. Păun [15] as distributed parallel computing devices [16] that process *multisets* of objects in the space delimited by a set of membranes called *membrane structure*. The contents of the membranes are modeled by means of *multisets* of symbols (sets that allow repetitions). These symbols can pass through membranes and some of them may act as catalysts. A membrane and its contents is called *region*, and all of them evolve simultaneously according to a set of production rules that describe how the multisets of the regions change. One of the regions is considered the *output membrane*; when the P Systems halts, this membrane contains the result of the computation of the P System.

This paper tackles the simulation of P Systems with mobile catalysts and without priority between their rules.

Java is currently one of the most popular object oriented programming languages. Java may be slower than other programming languages for computation - intensive problems. Nevertheless it is possible to run Java programs on a cluster of computers by means of a special Distributed Java Virtual Machine (DJVM) which supports parallel execution of Java threads. In this way, a multithreaded Java application runs on a cluster just as if it

were running on a single machine, but with the same performance as a big multi-processor machine. DJVMs are not included in the Sun's standard Java distributions. There are several different kinds of DJVM [9].

The work described in this paper is not the first attempt to code P System simulators. Previous approaches simulate different kinds of P Systems with different features. There are applications written in Prolog ([10, 8]), functional programming languages (LISP [19] or Haskell [2]), object oriented programming languages (C++ [5] or Java [11]), etc. ([1, 12, 4, 6, 7, 18, 17]). Some of them work only in batch mode (the user writes in a file the description of the P System, then he runs the simulator that shows the result of the computation in an output file), while others include a graphic user interface. Reference [12] describes a C library available at runtime, reference [5] describes a C++ simulator running on a cluster of computers.

This paper shows a multithread Java P System simulator that runs on a cluster of computers. This simulator is only the first step to build a Java tool to automatically design P Systems to solve given tasks. Our tool will include the hardware platform that will allow an efficient execution of the programs. Our group is also interested in offering a full access to the tool by means of Internet (applets, servlets, etc.). None of the existing simulators exactly fit our needs, so we had to develop our own simulator.

2 Multithread Java P System Simulator Running on a Cluster of Computers

Informal Description of the Classes

The following classes are used to simulate P Systems:

- *MultiSetSymbol* represents the simplest component of a region: a pair with the symbol name and the number of its copies.
- *MultiSet*, for the left hand sides of the production rules and the contents of the regions. It has a collection of *MultiSetSymbols*.
- *RightMultiSetSymbol*, for the simplest component of the right hand side of the production rules. It contains a *MultiSetSymbol* and the target region of the *MultiSetSymbol*.
- *RightMultiSet*, for the right hand side of the production rules. It has a collection of *RightMultiSetSymbols*.

- *Rule* is made of a *MultiSet* and a *RightMultiSet*. Its main method is *apply*. They are implemented as Java threads because they are applied simultaneously.

- A *Region* has a *Body* and a collection of *Rules*. Its main method is *applyRules*.

- A *Body*, that stands for the content of a region, has the *MultiSet* with the contents of the region. Its main methods are *addMultiSet* and *removeMultiSet*, invoked by the rules. They can simultaneously access the content of the *Body*, the mutual exclusion condition is ensured by means of a Java monitor and Java synchronized methods.

- A *P System* has a collection of symbol names that represents its alphabet, a collection of *Regions* and an output *Region*. Its main method is *compute*, that applies the rules of each region in the P System.

Figure 1 shows the UML class diagram for the simulator.

Figure 2 shows a snapshot of the window in which the user defines the P Systems.

When the user pushes the "ok" button, the simulation of the P System begins. If it terminates, the results are shown as the contents of the output region.

The work described in this paper uses JESSICA2 DJVM, that runs on linux clusters. This DJVM supports the same code as the standard JVM, so the programmer can assume that the code will run on a single JVM. JESSICA2 has three main building blocks: the *thread scheduler*, that balances the load of each processor, the *execution engine*, and the *global object space* module, that provides a distributed single heap to the Java threads. Each computer communicates with each other by means of TCP connections.

3 Experiments Performed and Expected Results

The kind of P Systems simulated in this work has been previously used to implement logical circuits. Reference [3] describes the way in which basic logical gates can be simulated by P Systems and how these gates are combined to compute logical functions.

This paper shows the results of simulating the basic logical gates and some other logical functions. The main goal is to compare the CPU time spent by our cluster and by a two-processor linux server, and, subsequently, to study the conditions in which the cluster is faster than the server.

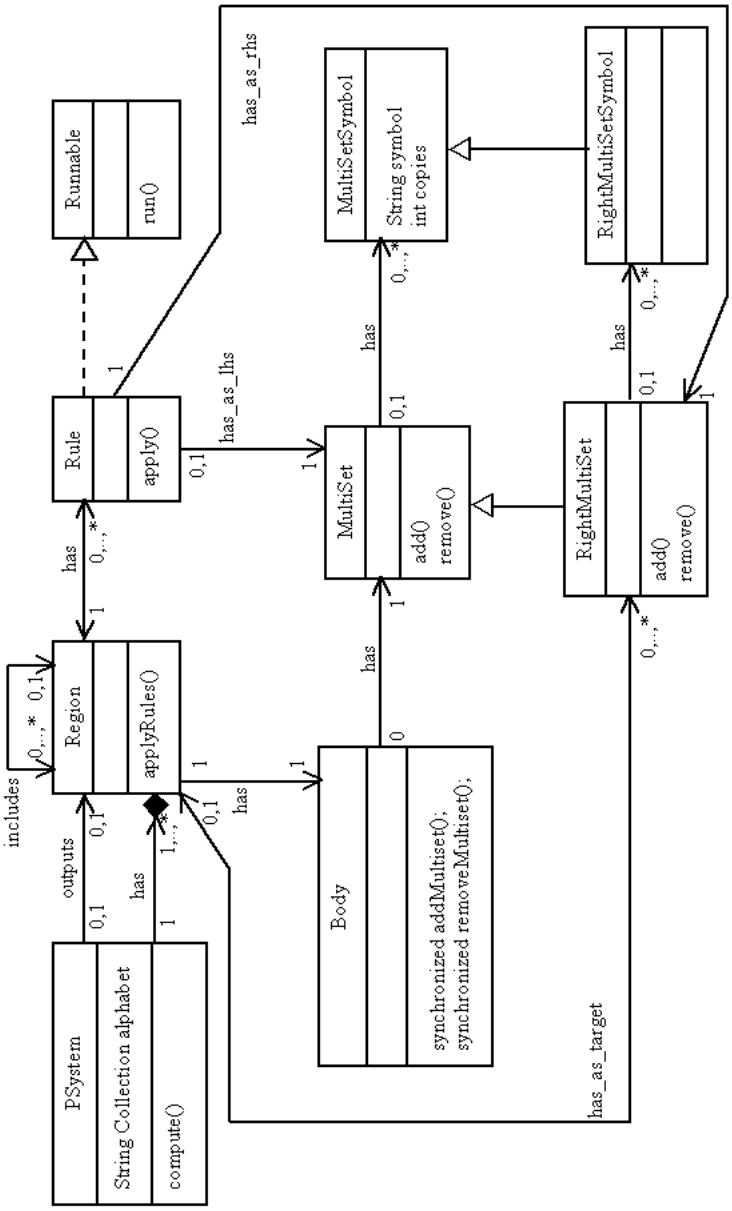


Figure 1: UML class diagram of the P System simulator

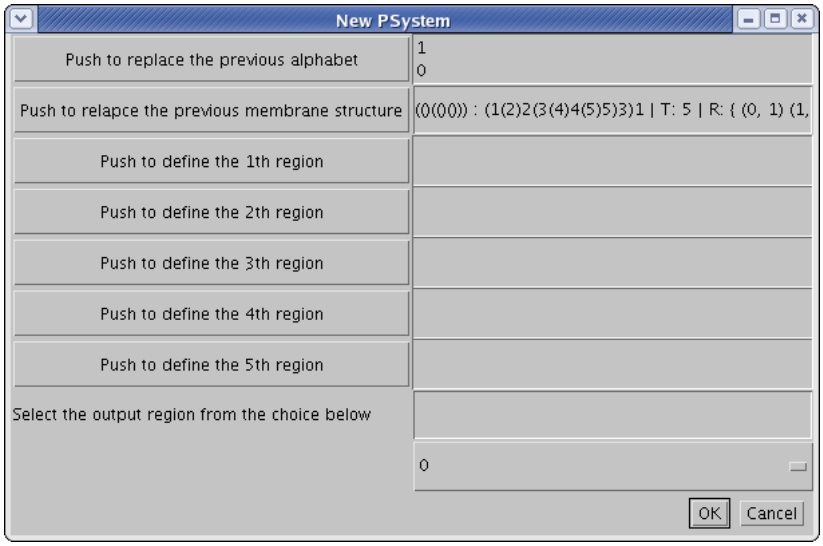


Figure 2: Window to define the regions of the P System

For every P System the following experiments are made:

- A maximum number of copies (*maxNumberOfCopies*) is specified for every P System
- The simulator is executed once for every number of copies (*currentCopies* from 1 to *maxNumberOfCopies*) both on the cluster and on the server.
- For every execution, the number of threads (*numberOfThreads*) and the CPU time spent (*CPUTime*) are computed.
- For every P System the relationship between the number of copies, threads and the time spent is analyzed and graphically represented. The results of the cluster are compared with those of the server, and we compute the values of *currentCopies* and *numberOfThreads* for which the cluster is faster than the server.

4 Future Research Lines

Some of the authors of this paper have previously proposed Christiansen Grammar Evolution [13], a new general purpose automatic programming

algorithm that improves the performance of Grammar Evolution by adding semantic restrictions.

Our group plans, in the future, to include this simulator into Christiansen Grammar Evolution in order to automatically design P Systems for solving given tasks. A first step is shown in [14], that proposes a Christiansen Grammar that fully describes the kind of P Systems simulated in this paper.

References

[1] P. C. Allilomes, E. G. Mamatas, K. T. Sotiriades, A. Syropoulos: A Distributed Simulation of Transition P Systems. In: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, International Workshop, WMC 2003, Revised Papers. *Lecture Notes In Computer Science* **2933**, Springer-Verlag, Heidelberg (2004), 357–368.

[2] F. Arroyo, A. V. Baranda, L. F. de Mingo, C. Luengo: A software simulation of transition P Systems in Haskell. *Pre-Proceedings of Second Workshop on Membrane Computing*. Curtea de Argeş, Romania (2002).

[3] R. Ceterchi, D. Sburlan: Simulating Boolean Circuits with P Systems. In: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, International Workshop, WMC 2003, Revised Papers. *Lecture Notes In Computer Science* **2933**, Springer-Verlag, Heidelberg (2004), 104–122.

[4] G. Ciobanu, D. Paraschiv: Membrane Software. A P System Simulator. In: *Pre-Proceedings of Workshop on Membrane Computing*. Curtea de Argeş, Romania (2001).

[5] G. Ciobanu, G. Wenyuan: P Systems Running on a Cluster of Computers. In: C. Martínn-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, International Workshop, WMC 2003, Revised Papers. *Lecture Notes In Computer Science* **2933**, Springer-Verlag, Heidelberg (2004), 123–139.

[6] G. Ciobanu, D. Paraschiv: P System Software Simulator. *Fundamenta Informaticae* **49** (1–3) (2002), 61–66.

[7] G. Ciobanu, G. Wenyuan: A parallel implementation of transition P Systems. In A. Alhazov, C. Martín-Vide, Gh. Păun (Eds.): *Preproceedings of the Workshop on Membrane Computing*, (2003),169–184

[8] A. Cordón-Franco, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, F. Sancho-Caparrini: A Prolog Simulator for Deterministic P Systems with Active Membranes. *New Generation Computing* **22** (4) (2004), 349–363.

[9] F. C. M Lau, C. Wang, W. Zhu: JESSICA2: A Distributed Java Virtual Machine with Transparent Thread Migration Support. *IEEE Fourth International Conference on Cluster Computing (CLUSTER 2002).* Chicago, USA (2002), 381–388.

[10] M. Malita: Membrane Computing in Prolog. *Pre-Proceedings Workshop on Multiset Processing.* Curtea de Argeş, Romania.

[11] I. A. Nepomuceno-Chamorro: A Java Simulator for Membrane Computing. *Journal of Universal Computer Science* **10** (5) (2004), 620–629.

[12] D. V. Nicolau Jr., G. Solana, F. Fulga, D. V. Nicolau: A. C. Library for Simulating P Systems. *Fundamenta Informaticae* **49** (1–3) (2002), 241–248.

[13] A. Ortega, M. de la Cruz, M. C. Alfonseca: Grammar Evolution: grammatical evolution with semantics. In press.

[14] A. Ortega de la Puente, R. Núñez Hervás, M. de la Cruz Echeandía, M. Alfonseca: Christiansen Grammar for some P systems. In: *Proceedings of the Third Brainstorming Week on Membrane Computing.* Sevilla, Spain (2005), 229–248.

[15] Gh. Păun: Computing with membranes. An introduction. *Bulletin of EATCS* **67** (1999), 139–152.

[16] Gh. Păun: *Membrane Computing: An introduction* Springer-Verlag, Berlin (2002).

[17] M. J. Pérez-Jiménez, F. J. Romero-Campero: A CLIPS Simulator for Recognizer P Systems with Active Membranes. In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini (Eds.): *Second Brainstorming Week on Membrane Computing.* Sevilla, Spain (2004), 387–413.

[18] B. Petreska, C. Teuscher: A Reconfigurable Hardware Membrane System. In: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, International Workshop,

WMC 2003, Revised Papers. *Lecture Notes In Computer Science* **2933**, Springer-Verlag, Heidelberg (2004), 269–285.

[19] Y. Suzuki, H. Tanaka: On a LISP Implementation of a Class of P Systems. *Romanian Journal of Information Science and Technology* **3** (2) (2000), 173–186.

Number of Protons/Bi-stable Catalysts and Membranes in P Systems. Time-Freeness

Artiom ALHAZOV

Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: artiome.alhazov@estudiants.urv.es

Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md

Abstract

Proton pumping P systems are a variant of membrane systems with both rewriting rules and symport/antiport rules, where a set of objects called protons is distinguished, every cooperative symport or antiport rule involves a proton, but no rewriting rule does. Time-freeness property means the result of all computations does not depend on the rules execution times.

This article's aim is to improve (showing that two membranes are sufficient) the known universality results on proton pumping P systems, establishing at the same time an upper bound on the number of protons, namely one, or four for time-free systems.

All results mentioned are valid for proton pumping P systems with non-cooperative rewriting and either symport/antiport rules of weight 1 (classical variant) or symport rules of weight at most 2. As a corollary, we obtain the universality of P systems with one membrane and one bi-stable catalyst, or the universality of time-free P systems with one membrane and four bi-stable catalysts. All universality results are stated as generating RE (except the time-free systems without targets generate $PsRE$).

1 Introduction

Membrane computing is a rapidly developing field, launched in 1998 by Gheorghe Păun, see [13] for a systematic survey and [15] for a comprehensive bibliography. It studies, among others, the computational power of devices with multisets distributed over a tree-like membrane structure and rules rewriting and/or moving objects (elements of these multisets).

In evolution-communication P systems as introduced in [5], there are two types of rules: simple rewriting rules associated to regions and symport/antiport rules associated to membranes. Rules of the first type change the objects in the region where they are, while the latter ones move the objects across the membrane; thus, separating evolution and communication.

Proton pumping P systems as introduced in [4] are a restricted variant of evolution-communication P systems: the set of protons is a subset of objects, no evolution rule involves a proton, while every cooperative communication rule has to involve exactly one proton. Thus, a proton is a "catalyst of communication". However, since the proton is also moved to another region, this hints its "multi-stability" (bi-stability if it moves between two regions), allowing the one-proton results of this paper.

It is worth mentioning that these models, although being formal and abstract, are motivated by cell biology (e.g., in many bacteria, the only antiports available are those that can exchange a proton with some chemical objects), see [1] and [14].

Time-freeness is a property introduced in [8],[6]. Consider a fixed P system Π and an arbitrary mapping e from the set of all rules to the set of positive integers. If the result of all halting computations of Π, where the rules are executed in the number of steps specified by e, is independent on e, then Π is called time-free.

It has been established that EC P systems with 2 membranes are universal with non-cooperative evolution and either symport/antiport rules of weight 1 ([2]) or symport rules of weight at most 2 ([10]); moreover, the constructions can be made time-free ([3]). For the proton pumping P systems with non-cooperative evolution and symport/antiport rules of weight 1, three membranes are enough for universality, while considering only one kind of protons and strong or weak priority of proton pumping rules, at least Parikh images of $ET0L$ languages can be generated with two membranes.

In this article we improve the universality result of proton pumping P systems with symport/antiport rules of weight 1 from three membranes and an unbounded number of protons to only two membranes and only 4 protons; moreover, the underlying system is even time-free. We also strengthen the

universality result of time-free evolution-communication P systems with two membranes and symport rules of weight at most 2 by proving the same result for proton pumping P systems with 4 protons.

Surprisingly, one can decrease the number of kinds of protons to one by giving up the time-freeness for the proton pumping P systems with non-cooperative evolution, symport of weight 1 and either form of minimal cooperation (antiport of weight 1 or symport of weight 2).

Finally, protons in two-membrane P systems behave like bi-stable catalysts in one region, so the corresponding corollaries hold for one-membrane P systems with bi-stable catalysts, improving results from [11], [6].

2 Definitions and Preliminaries

2.1 Proton Pumping

We will now recall from [4] the definition of proton pumping P systems. The notation has been changed a little, and the definition has been slightly reformulated (restricted). First, the multiset describing the initial contents of the environment is no longer considered, as the environment is initially empty. Second, all communicative rules are listed together (for a shorter description, symport and antiport rules associated to region i are not divided any more into two sets). Third, we now also require that cooperative symport rules also involve a proton (in [4], cooperative symport rules were not studied). Fourth, the rules involving a proton are now restricted to involve exactly one (like in the catalyst case).

Definition 2.1 *A* proton pumping P system *of degree $m \geq 1$, is defined as*

$$\Pi = (O, P, \mu, w_1, w_2, \cdots, w_m, R_1, \cdots, R_m, R'_1, \cdots, R'_m, i_0) \text{ where}$$

- *O is the alphabet of objects, $P \subseteq O$ is a set of protons;*
- *μ is a membrane structure with m membranes injectively labeled by $1, 2, \cdots, m$;*
- *w_i are strings which represent multisets over O associated with regions $1, 2, \cdots, m$ of μ;*
- *R_i, $1 \leq i \leq m$, are finite sets of simple evolution rules over O; R_i is associated with the region i of μ;*
- *R'_i, $1 \leq i \leq m$, are finite sets of symport/antiport rules over O of a restricted form; R'_i is associated with the membrane i of μ;*

- $i_0 \in \{0, 1, 2, \cdots, m\}$ *is the output region; if* $i_0 = 0$*, then it is the environment, otherwise* i_0 *is a label of some membrane of* μ*.*

A simple evolution rule is of the form $u \to v$*, where* u *and* v *are strings over* $O - P$ *(the variant can be extended by allowing to assign the target indications here, out,* in_j*, to the symbols in* v*; for evolution–communication P systems this was first used in [10]). The only symport/antiport rules allowed are of the following forms:* (a) *uniport rules:* (a, in)*,* (a, out)*,* $a \in O$ *(notice that in this article we never use uniport of protons);* (b) *antiport rules with a proton on one side:* $(x, out; p, in)$*,* $(p, out; x, in)$*,* $p \in P$*,* $x \in (O-P)^+$*;* (c) *symport rules with a proton:* (px, out)*,* (px, in)*,* $p \in P$*,* $x \in (O - P)^+$*.*

The m-tuple of multisets of objects present at any moment in the regions of Π represents the configuration of the system at that moment (the m-tuple $(w_1, \cdots, w_m)$ is the initial configuration). A transition between configurations is governed by the mixed application of the evolution rules and of the symport/antiport rules. All rules are applied in a maximally parallel way (no rules are applicable to the objects that remain idle), chosen non-deterministically.

The system continues parallel steps until there remain no applicable rules (evolution rules or symport/antiport rules) in any region of Π. Then the system halts, and we consider the number of objects in the output region i_0 at the moment when the system halts as the result of the computation of Π. The set of all natural numbers computed in this way is denoted by $N(\Pi)$. If instead of the total number, the multiplicities of objects are considered, then the result is denoted by $Ps(\Pi)$. In case of external output, one can also consider the sequence in which the objects are sent into the environment, denoting the result by $L(\Pi)$.

A *bi-stable catalyst* is a pair of symbols $c, c' \in O$ such that all rules where these symbols appear are of the following forms: $ca \to cv$, $ca \to c'v$, $c'a \to cv$, $c'a \to c'v$. When speaking of a P system with bi-stable catalysts, we will additionally specify the set C_b of bi-stable catalysts in the description of the P system. We use the following notations

$$\mathbf{X}ProP^{\mathbf{k}}_{\mathbf{m}}(\boldsymbol{\alpha}, tar, sym_{\mathbf{i}}, anti_{\mathbf{j}})$$

to denote the family of languages ($\mathbf{X} = L$), vector sets ($\mathbf{X} = Ps$) or number sets ($\mathbf{X} = N$) generated by proton pumping P systems with at most $\mathbf{m}$ membranes, $\mathbf{k}$ different types of protons (i.e., k is the cardinality of the set P), using symport rules of weight at most $\mathbf{i}$, antiport rules of weight at most $\mathbf{j}$, and non-cooperative ($\boldsymbol{\alpha} = ncoo$) or bi-stable catalytic ($\boldsymbol{\alpha} = 2cat_{\mathbf{l}}$) with $\mathbf{l}$

bi-stable catalysts evolution rules with targets. If targets are not allowed, then *tar* is removed from the notation (like any other unused feature). If one of the numbers m, k, i, j, l is unbounded, we write $*$ instead). For P systems without protons, we will replace $ProP^{\mathbf{k}}$ by OP in the notation and exclude the specification of the set P, as well as the sets of symport/antiport rules if they are not used, from the description of the P system.

2.2 Time-Freeness

We now recall from [8] the definition of time-free P systems for the case of proton pumping P systems (for P systems without protons it is done in the same way).

Given a time-mapping $e : R_1 \cup \cdots \cup R_m \cup R'_1 \cup \cdots \cup R'_m \longrightarrow \mathbb{N}_1$ and a proton pumping P system Π as defined above, it is possible to construct a *timed proton pumping P system* $\Pi(e)$ working in the following way.

We suppose the existence of an external and global clock that ticks at uniform intervals of time. At each time in the regions of the system we have both rules (both evolution and transport) in execution and rules not in execution. At each time all the evolution and transport rules that can be applied (started) in each region, have to be applied. If a rule $r \in R_i, R'_i, 1 \leq i \leq m$, is applied, then all objects that can be processed by the rule have to evolve by this rule (a rule is applied in a maximally parallel manner as standard in the P system area).

As usual, the rules from R_i are applied to objects in region i and the rules from R'_i govern the communication of objects through membrane i. There is no difference between evolution rules and communication rules: they are chosen and applied in the non-deterministic maximally parallel manner. When an evolution rule or a transport rule r is started at time j, its execution terminates at time $j+e(r)$. If two rules are started in the same time unit, then possible conflicts for using the occurrences of symbol-objects are solved assigning the objects in a non-deterministic way (again, in the way usually defined in the P system area). Notice that when the execution of a rule r is started, the occurrences of objects used by this rule are not anymore available for other rules during the entire execution of r.

A proton pumping P system Π is *time-free* if and only if every system in the set $\{\Pi(e) \mid e : R \longrightarrow \mathbb{N}_1\}$ (where $R = R_1 \cup \cdots \cup R_m \cup R'_1 \cup \cdots \cup R'_m$) produces the same result.

As all $\Pi(e)$ generate the same result, in this case the set of natural numbers (vectors, words) generated by a time-free proton pumping P system Π is denoted by $N(\Pi)$ ($Ps(\Pi)$, $L(\Pi)$). For the notation of what is generated

by a family of *time-free* P systems, we add f to the notation introduced before: $f\mathbf{X}ProP^{\mathbf{k}}_{\mathbf{m}}(\boldsymbol{\alpha}, tar, sym_{\mathbf{i}}, anti_{\mathbf{j}})$, $f\mathbf{X}OP^{\mathbf{k}}_{\mathbf{m}}(\boldsymbol{\alpha}, tar, sym_{\mathbf{i}}, anti_{\mathbf{j}})$.

2.3 Register Machines

In what follows, we will use register machines as an important tool for showing the computational completeness results. Let us recall their definitions from [12].

An n-register machine is a construct $M = (n, l_0, l_h, I)$ where:

- n is the number of registers;
- I is a set of labeled instructions of the form $(l : op(i), l', l'')$ where $op(i)$ is an operation on register i of M; symbols l, l', l'' belong to the set of labels associated in a one-to-one manner with instructions of I;
- l_0 is the initial label;
- l_h is the final label.

The instructions allowed by an n-register machine are:

- $(l : A(i), l', l'')$ – add one to the contents of register i and proceed to instruction l' or to instruction l'';
- $(l : S(i), l', l'')$ – jump to instruction l' if register i is empty; otherwise subtract one from register i and jump to the instruction labeled by l'' (these two cases are often called *zero-test* and *decrement*);
- $(l_h : halt)$ – finish the computation. This is the unique instruction with label h.

If a register machine $M = (n, l_0, l_h, I)$, starting from the instruction labeled by l_0 with all registers being empty, halts with values n_j in register j, $1 \leq j \leq m$, and the contents of registers $m+1, \cdots, n$ being empty, then it generates the vector $(n_1, \cdots, n_k) \in \mathbb{N}^m$. Any recursively enumerable set of vectors of nonnegative integers can be generated by a register machine.

It is known that register machines with $m+2$ registers can generate all recursively enumerable sets of m-dimensional vectors (we can also require that the only instructions associated to the output registers are increment instructions). Moreover, in this case, if incrementing register i ($1 \leq i \leq m$) is interpreted as writing a symbol a_i, (similar to reading a symbol a_i by counter machines), then a word is written instead of a vector being generated; in this way, register machines can generate languages (with $m+2$ registers, any recursively enumerable language over $T = \{a_1, \cdots, a_m\}$ can be generated).

3 Time-Free Results

Theorem 3.1 $fPsProP_2^4(ncoo, sym_1, anti_1) = PsRE.$

Proof. We only prove the inclusion $\supseteq$. Consider an arbitrary recursively enumerable language with m symbols: $L \subseteq \{a_i \mid 1 \leq i \leq m\}^*$. Then there is a register machine $M = (m+2, l_0, l_h, I)$ generating L, and let $I_- = \{l : (S(i), l', l'') \in I\}$.

We will construct a P system Π simulating M in such a way that the value of register $i \in W = \{m+1, m+2\}$ is represented by the multiplicity of the object a_i in the skin region. The proton D_i will be used to decrement the value of register i, while E_i will be used to check if the register i is empty.

$$\begin{aligned}
\Pi &= (O, P, [_1\ [_2\ ]_2\ ]_1, w_1, w_2, R_1, R_2, R_1', R_2'), \text{ where} \\
O &= \{a_i \mid 1 \leq i \leq m+2\} \cup \{l_j \mid l \in I_-, 1 \leq j \leq 4\} \cup \{\#_1, \#_2\} \cup I \cup P, \\
P &= \{D_i, E_i \mid i \in W\}, \\
w_1 &= l_0 D_{m+1} D_{m+2} Z_{m+1} Z_{m+2} \#_1,\ w_2 = \lambda,
\end{aligned}$$

and the sets of rules are the following:

For each instruction $l : (A(i), l', l'') \in I$,

$$l \to a_i l',\ l \to a_i l'' \in R_1.$$

Moreover, for $1 \leq i \leq m$ we have the rules

$$(a_i, out) \in R_1'.$$

For each instruction $l : (S(i), l', l'') \in I$,

$$\begin{aligned}
& (l, in) \in R_2', \\
\text{(decrement)} \quad & l \to l_4 \in R_2, \\
& (l_4, out; D_i, in), (D_i, out; a_i, in), (D_i, out; \#_1, in) \in R_2', \\
& l_4 \to l' \in R_1, \\
\text{(zero test)} \quad & l \to l_1,\ l_2 \to l_3 \in R_2, \\
& (E_i, in; l_1, out),\ (E_i, out; a_i, in),\ (E_i, out; l_2, in) \in R_2', \\
& (l_3, out) \in R_2', \\
& l_1 \to l_2,\ l_2 \to \#_2,\ l_3 \to l'' \in R_1.
\end{aligned}$$

Finally, we also have the rules

$$\begin{aligned}
\#_1 \to \#_1 \in R_2, \\
\#_2 \to \#_2 \in R_1.
\end{aligned}$$

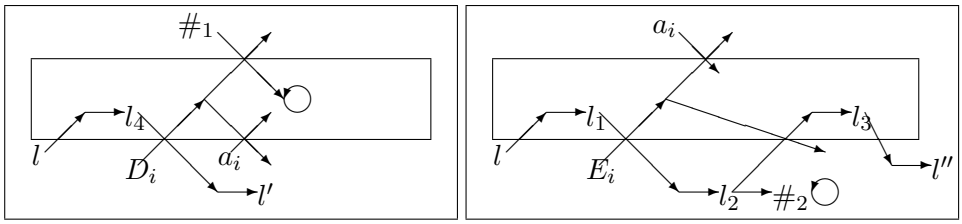


Figure 1: using $(ncoo, sym_1, anti_1)$ decrement: left, zero-test: right

The system constructed in that way simulates the corresponding register machine. The increment instructions are simulated in one step: the instruction symbol changes to a symbol corresponding to the next instruction and a symbol corresponding to the register being incremented.

Decrement: l comes to region 2, changes to l_4 and returns to region 1, bringing D_i to region 2, and then changes to l'. The "duty" of D_i is to decrement register i by returning to region 1 and removing one copy of a_i from region 1. If register i is empty, then D_i exchanges with $\#_1$ and the computation never halts (if decrement is possible, D_i can still exchange with $\#_1$, but this case is not productive).

Zero-test: after l has come to region 2, it changes to l_1 and returns to region 1, bringing E_i to region 2, and then changes to l_2. The "duty" of E_i is to check that register i is empty by waiting for l_2. If register i is not empty, then E_i will immediately exchange with a_i and then l_2 will change to $\#_2$, so the computation will never halt (if E_i waits for l_2, l_2 can still change to $\#_2$, but this case is not productive).

The decrement and zero-test are illustrated in Figure 1. From these figures it is clear that the system is time-free: most of the correct simulation is sequential, and we only remark one point - the time it takes to exchange D_i and a_i is not relevant because after the start of the rule a_i is already unavailable in region 1, and, moreover, if D_i still has not returned to region 1 for the next decrement instruction, the system will simply wait for it. □

Theorem 3.2 $fPsProP_2^4(ncoo, sym_2) = PsRE$.

Proof. This is a "dual" theorem: the simulation of a register machine is done in exactly the same way, except that the protons that were in region 1

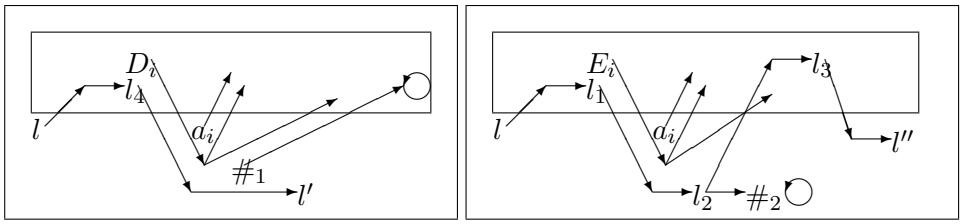


Figure 2: using $(ncoo, sym_2)$ decrement: left, zero-test: right

are now in region 2, and vice-versa. The system we consider is:

$$
\begin{aligned}
\Pi &= (O, P, [_1\ [_2\ ]_2\ ]_1, w_1, w_2, R_1, R_2, R_1', R_2'), \text{ where} \\
O &= \{a_i \mid 1 \leq i \leq m+2\} \cup \{l_j \mid l \in I_-, 1 \leq j \leq 4\} \cup \{\#_1, \#_2\} \cup I \cup P, \\
P &= \{D_i, E_i \mid i \in W\}, \\
w_1 &= l_0 \#_1,\ w_2 = D_{m+1} D_{m+2} Z_{m+1} Z_{m+2}, \\
R_1 &= \{l \to a_i l', l \to a_i l'' \mid l : (A(i), l', l'') \in I\} \cup \{\#_2 \to \#_2\} \\
&\cup \{l_4 \to l', l_1 \to l_2, l_2 \to \#_2, l_3 \to l'' \mid l : (S(i), l', l'') \in I\}, \\
R_2 &= \{l \to l_4, l \to l_1, l_2 \to l_3 \mid l : (S(i), l', l'') \in I\} \cup \{\#_1 \to \#_1\}, \\
R_1' &= \{(a_i, out) \mid 1 \leq i \leq m\}, \\
R_2' &= \{(l, in), (l_4 D_i, out), (a_i D_i, in), (\#_1 D_i, in), \\
&\quad (l_1 E_i, out), (a_i E_i, in), (l_2 E_i, in), (l_3, out) \mid l : (S(i), l', l'') \in I\}.
\end{aligned}
$$

The system constructed above simulates the corresponding register machine. The increment instructions are simulated in one step: the instruction symbol changes to a symbol corresponding to the next instruction and a symbol corresponding to the register being incremented.

Decrement: l comes to region 2, changes to l_4 and returns to region 1 with D_i, and then changes to l'. The "duty" of D_i is to decrement register i by returning to region 2 and removing one copy of a_i from region 1. If register i is empty, then D_i exchanges with $\#_1$ and the computation never halts (if decrement is possible, D_i can still exchange with $\#_1$, but this case is not productive).

Zero-test: after l has come to region 2, it changes to l_1 and returns to region 1 with E_i, and then changes to l_2. The "duty" of E_i is to check that register i is empty by waiting for l_2. If register i is not empty, then E_i will immediately exchange with a_i and then l_2 will change to $\#_2$, so the computation will never halt (if E_i waits for l_2, l_2 can still change to $\#_2$, but this case is not productive).

As in the previous proof, the time-freeness of the system immediately becomes clear from the illustrations in Figure 2. □

In the proofs of the preceding theorems the output symbols are generated in the right order; however, generating languages by these constructions is not time-free because the different execution times of the rules sending output symbols to the environment might lead to changing the order of symbols in the output word.

Nevertheless, if target indications are allowed, then, replacing rules $l \to a_i l' \in R_1$, $l \to a_i l'' \in R_1$, $(a_i, out) \in R'_1$ for $1 \leq i \leq m$ by $l \to (a_i)_{out} l' \in R_1$, $l \to (a_i)_{out} l'' \in R_1$, one obtains time-free P systems generating RE.

Corollary 3.1 $fLProP_2^4(ncoo, tar, sym_1, anti_1) = RE$, $fLProP_2^4(ncoo, tar, sym_2) = RE$.

4 One Proton

We will now show that even one proton is enough for computational completeness, again with only two membranes.

Theorem 4.1 $LProP_2^1(ncoo, sym_1, anti_1) = RE$.

Proof. We only prove the inclusion $\supseteq$. Consider an arbitrary recursively enumerable language with m symbols: $L \subseteq \{a_i \mid 1 \leq i \leq m\}^*$. Then there is a register machine $M = (m+2, l_0, l_h, I)$ generating L, and let $I_- = \{l : (S(i), l', l'') \in I\}$.

We will construct a P system Π simulating M in such a way that the value of register $i \in W = \{m+1, m+2\}$ is represented by the multiplicity of the object a_i in region $i - m$. The proton p will be used to decrement/zero test the value of the working registers.

$$\begin{aligned}
\Pi &= (O, P, [_1\ [_2\ ]_2\ ]_1, w_1, w_2, R_1, R_2, R'_1, R'_2), \text{ where} \\
O &= \{a_i, a'_i \mid 1 \leq i \leq m+2\} \cup \{l_j \mid l \in I, 1 \leq j \leq 9\} \cup I \cup P \\
&\cup \{\#, I_{2,4}, I_{1,3}, I_{0,2}\} \cup \{I_j \mid 0 \leq j \leq 2\} \cup \{O_j \mid 0 \leq j \leq 5\}, \\
P &= \{p\},\ w_1 = pI_1,\ w_2 = o_0 l_0,
\end{aligned}$$

and the sets of rules are the following:

Rules related to special objects which "wait" for a certain time and then must exchange with the proton (or else the trap symbol will be introduced):

$$
\begin{aligned}
&I_{2,4} \to I_{1,3},\ I_{1,3} \to I_{0,2} \in R_2,\\
&(I_{0,2}, out) \in R_2',\\
&I_{0,2} \to I_0 I_2 \in R_1,\\
&I_{j+1} \to I_j \in R_1,\ 0 \le j \le 1,\\
&(p, out; I_0, in) \in R_2',\\
&O_0 \to \lambda,\ I_0 \to \#,\ \# \to \# \in R_1,\\
&O_{j+1} \to O_j \in R_2,\ 0 \le j \le 4,\\
&(O_0, out; p, in) \in R_2',\\
&I_0 \to \lambda,\ O_0 \to \#,\ \# \to \# \in R_2.
\end{aligned}
$$

Rules of interaction of the proton and register symbols:

$$(p, out; a_{m+1}, in),\ (a_{m+2}, out; p, in) \in R_2'.$$

For each instruction $l : (A(i), l', l'') \in I$,

$$
\begin{aligned}
&l \to a_i' l_1 O_3 O_1 I_{0,2} \in R_2,\\
&l_j \to l_{j+1} \in R_2,\ 1 \le j \le 2,\\
&l_3 \to l',\ l_3 \to l'' \in R_2.
\end{aligned}
$$

The register symbol a_i in region $\max(i - m, 0)$ is produced by the rules

$$
\begin{aligned}
&a_{m+2}' \to a_{m+2} \in R_2,\\
&(a_i', out) \in R_2',\ 1 \le i \le m+1,\\
&a_i' \to a_i \in R_1,\ 1 \le i \le m+1,\\
&(a_i, out) \in R_1',\ 1 \le i \le m.
\end{aligned}
$$

For each instruction $l : (S(m+1), l', l'') \in I$,

$$
\begin{aligned}
\text{(decrement)} \quad &l \to l_1 O_1 O_3 O_5 I_{2,4} \in R_2,\\
&l_j \to l_{j+1} \in R_2,\ 1 \le j \le 4,\\
&l_5 \to l' \in R_2,\\
\text{(zero test)} \quad &l \to l_6 O_1 O_4 I_{1,3} \in R_2,\\
&l_j \to l_{j+1} \in R_2,\ 6 \le j \le 8,\\
&l_9 \to l'' \in R_2.
\end{aligned}
$$

For each instruction $l : (S(m+2), l', l'') \in I$,

$$\begin{aligned}
\text{(decrement)} \qquad & l \to l_1 O_3 I_{0,2} \in R_2, \\
& l_j \to l_{j+1} \in R_2,\ 1 \leq j \leq 2, \\
& l_3 \to l' \in R_2, \\
\text{(zero test)} \qquad & l \to l_6 O_2 O_4 I_{1,3} \in R_2, \\
& l_j \to l_{j+1} \in R_2,\ 6 \leq j \leq 8, \\
& l_9 \to l'' \in R_2.
\end{aligned}$$

For terminating the computation we have

$$l_h \to \lambda \in R_2.$$

The simulation is illustrated by the tables below. Notice that every time an antiport rule is possible it must be executed, otherwise one of the objects O_0, I_0 will change to #, leading to an infinite computation.

The intuitive idea behind this construction is to create a "predefined scenario" for the proton; if the system tries to decrement a empty register or the system zero-tests a non-empty register, then the proton ends up in a "wrong" region and cannot follow the "scenario" anymore. We now list the scenarios for the proton, for different instructions:

- Decrement a_{m+1}: p exchanges with O_0, then with I_0, then with O_0, then with a_{m+1}, then with O_0, and finally with I_0.
- Zero-test a_{m+1}: p exchanges with O_0, then with I_0, then with O_0, then waits one step because there is no a_{m+1}, and finally with I_0.
- Decrement a_{m+2}: p exchanges with O_0, then with I_0, then with a_{m+2}, and finally I_0.
- Zero-test a_{m+2}: p exchanges with O_0, then with I_0, then waits one step because there is no a_{m+2}, then with O_0, and finally I_0.
- Increment any register: p exchanges with O_0, then with I_0, then with O_0, and finally I_0.

Notice that the first two steps of the simulation are always the same. This is needed to "keep the proton busy" while the object associated to the instruction creates the rest of the scenario. The scenario is created by producing objects O_0 in region 2 and objects I_0 in region 1, with corresponding delays.

When the output register is incremented, the corresponding symbol is sent to the environment, contributing to the result. At the end of the correct simulation, object l_h is erased, registers $m+1$ and $m+2$ are empty, so no objects are present in region 2, while region 1 only contains p. □

Instr	Decrement a_{m+1}		Zero-test a_{m+1}	
Step	Region 1	Region 2	Region 1	Region 2
1	$a_{m+1}I_1p$	lO_0	I_1p	lO_0
2	$a_{m+1}I_0O_0$	$l_1I_{2,4}O_5O_3O_1p$	I_0O_0	$l_6I_{1,3}O_4O_1p$
3	$a_{m+1}p$	$l_2I_{1,3}O_4O_2O_0I_0$	p	$l_7I_{0,2}O_3O_0I_0$
4	$a_{m+1}O_0$	$l_3I_{0,2}O_3O_1p$	$I_{0,2}O_0$	l_8O_2p
5	$I_{0,2}p$	$a_{m+1}l_4O_2O_0$	I_2I_0	l_9O_1p
6	$I_2I_0O_0$	$a_{m+1}l_5O_1p$	I_1p	$l''O_0I_0$
7	I_1p	$a_{m+1}l'O_0I_0$	Next instr.	Next instr.

Instr	Decrement a_{m+2}		Zero-test a_{m+2}	
Step	Region 1	Region 2	Region 1	Region 2
1	I_1p	$a_{m+2}lO_0$	I_1p	lO_0
2	I_0O_0	$a_{m+2}l_1I_{0,2}O_3p$	I_0O_0	$l_6I_{1,3}O_4O_2p$
3	$I_{0,2}p$	$a_{m+2}l_2O_2I_0$	p	$l_7I_{0,2}O_3O_1I_0$
4	$a_{m+2}I_2I_0$	l_3O_1p	$I_{0,2}p$	$l_8O_2O_0$
5	$a_{m+2}I_1p$	$l'O_0I_0$	$I_2I_0O_0$	l_9O_1p
6	Next instr.	Next instr.	I_1p	$l''O_0I_0$

Instr	Increment a_i		$i \leq m$		$i=m+1$	$i=m+2$	Terminate	
R.	1	2	0	1	1	2	1	2
1	I_1p	lO_0					pI_1	l_hO_0
2	I_0O_0	$l_1a_i'I_{0,2}O_3O_1p$					O_0I_0	p
3	$I_{0,2}p$	$l_2O_2O_0I_0$		a_i'	a_{m+1}'	a_{m+2}	p	I_0
4	$I_2I_0O_0$	l_3O_1p		a_i	a_{m+1}	a_{m+2}	p	
5	I_1p	(l' or l'')O_0I_0	a_i		a_{m+1}	a_{m+2}	Halt	Halt

Theorem 4.2 $LProP_2^1(ncoo, sym_2) = RE$.

Proof. This is a "dual" theorem: the simulation of a register machine is done in exactly the same way, except that the proton that was in region 1

is now in region 2, and vice-versa, and except that the halting is slightly modified such that the proton stays in region 1.

$$
\begin{aligned}
\Pi &= (O, P, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, R_1, R_2, R'_1, R'_2), \text{ where} \\
O &= \{a_i, a'_i \mid 1 \leq i \leq m+2\} \cup \{l_j \mid l \in I, 1 \leq j \leq 9\} \cup I \cup P \\
&\cup \{\#, I_{2,4}, I_{1,3}, I_{0,2}\} \cup \{I_j \mid 0 \leq j \leq 2\} \cup \{O_j \mid 0 \leq j \leq 5\}, \\
P &= \{p\},\ w_1 = I_1,\ w_2 = po_0l_0,
\end{aligned}
$$

and the sets of rules are the following:

$$
\begin{aligned}
R_1 &= \{I_{0,2} \to I_0I_2, O_0 \to \lambda, I_0 \to \#, \# \to \#\} \\
&\cup \{I_{j+1} \to I_j \in R_1 \mid 0 \leq j \leq 1\} \\
&\cup \{a'_i \to a_i \mid 1 \leq i \leq m+1\}, \\
R_2 &= \{l \to a'_il_1O_3O_1I_{0,2}, l_3 \to l', l_3 \to l'' \mid l : (A(i), l', l'') \in I\} \\
&\cup \{l_j \to l_{j+1} \mid 1 \leq j \leq 2,\ l : (A(i), l', l'') \in I\} \\
&\cup \{l \to l_1O_1O_3O_5I_{2,4}, l_5 \to l', l \to l_6O_1O_4I_{1,3}, l_9 \to l'' \\
&\quad \mid l : (S(m+1), l', l'') \in I\} \\
&\cup \{l \to l_1O_3I_{0,2}, l_3 \to l', l \to l_6O_2O_4I_{1,3}, l_9 \to l'' \\
&\quad \mid l : (S(m+2), l', l'') \in I\} \\
&\cup \{l_j \to l_{j+1} \mid 1 \leq j \leq 4,\ l : (S(m+1), l', l'') \in I\} \\
&\cup \{l_j \to l_{j+1} \mid 1 \leq j \leq 2,\ l : (S(m+2), l', l'') \in I\} \\
&\cup \{l_j \to l_{j+1} \mid 6 \leq j \leq 8,\ l : (S(i), l', l'') \in I\} \\
&\cup \{I_{2,4} \to I_{1,3}, I_{1,3} \to I_{0,2}, I_0 \to \lambda, O_0 \to \#, \# \to \#\} \\
&\cup \{a'_{m+2} \to a_{m+2}, l_h \to O_1\} \cup \{O_{j+1} \to O_j \mid 0 \leq j \leq 4\}, \\
R'_1 &= \{(a_i, out) \mid 1 \leq i \leq m\}, \\
R'_2 &= \{(I_{0,2}, out), (pI_0, in), (pO_0, out), (pa_{m+1}, in), (pa_{m+2}, out)\} \\
&\cup \{(a'_i, out) \mid 1 \leq i \leq m+1\}.
\end{aligned}
$$

The simulation is illustrated by the tables below. Notice that every time an antiport rule is possible it must be executed, otherwise one of the objects O_0, I_0 will change to #, leading to an infinite computation.

Like in the previous proof, to arrive at a halting configuration, the proton must follow the "predefined scenario" created by instruction objects. If the system tries to decrement an empty register or the system zero-tests a non-empty register, then the proton ends up in a "wrong" region and cannot follow the "scenario". We now list the proton's scenarios.

- Decrement a_{m+1}: p accompanies O_0, then I_0, then O_0, then a_{m+1}, then O_0, and finally I_0.
- Zero-test a_{m+1}: p accompanies O_0, then I_0, then O_0, then waits one step because there is no a_{m+1}, and finally goes with I_0.
- Decrement a_{m+2}: p moves O_0, then I_0, then a_{m+2}, and finally I_0.
- Zero-test a_{m+2}: p accompanies O_0, then I_0, then waits one step because there is no a_{m+2}, then goes with O_0, and finally with I_0.
- Increment any register: p accompanies O_0, then I_0, then O_0, and finally I_0.
- Halt: p accompanies O_0, then I_0, and finally O_0.

Again, the first two steps are the same, to "keep the proton busy" while the object associated to the instruction creates the rest of the scenario. The scenario is created by producing objects O_0 in region 2 and objects I_0 in region 1, with corresponding delays.

When the output register is incremented, the corresponding symbol is sent to the environment, contributing to the result. At the end of the correct simulation, object l_h changes to O_0 in 3 steps, moving p to region 1. Since registers $m+1$ and $m+2$ are empty, no objects are present in region 2, while region 1 only contains p. □

Instr	Decrement a_{m+1}		Zero-test a_{m+1}	
Step	Region 1	Region 2	Region 1	Region 2
1	$a_{m+1}I_1$	lO_0p	I_1	lO_0p
2	$a_{m+1}I_0O_0p$	$l_1I_{2,4}O_5O_3O_1$	I_0O_0p	$l_6I_{1,3}O_4O_1$
3	a_{m+1}	$l_2I_{1,3}O_4O_2O_0I_0p$		$l_7I_{0,2}O_3O_0I_0p$
4	$a_{m+1}O_0p$	$l_3I_{0,2}O_3O_1$	$I_{0,2}O_0p$	l_8O_2
5	$I_{0,2}$	$a_{m+1}l_4O_2O_0p$	I_2I_0p	l_9O_1
6	$I_2I_0O_0p$	$a_{m+1}l_5O_1$	I_1	$l''O_0I_0p$
7	I_1	$a_{m+1}l'O_0I_0p$	Next instr.	Next instr.

Instr	Decrement a_{m+2}		Zero-test a_{m+2}	
Step	Region 1	Region 2	Region 1	Region 2
1	I_1	$a_{m+2}lO_0p$	I_1	lO_0p
2	I_0O_0p	$a_{m+2}l_1I_{0,2}O_3$	I_0O_0p	$l_6I_{1,3}O_4O_2$
3	$I_{0,2}$	$a_{m+2}l_2O_2I_0p$		$l_7I_{0,2}O_3O_1I_0p$
4	$a_{m+2}I_2I_0p$	l_3O_1	$I_{0,2}$	$l_8O_2O_0p$
5	$a_{m+2}I_1$	$l'O_0I_0p$	$I_2I_0O_0p$	l_9O_1
6	Next instr.	Next instr.	I_1	$l''O_0I_0p$

I.	Increment a_i		$i \leq m$		$i=m+1$	$i=m+2$	Terminate	
R.	1	2	0	1	1	2	1	2
1	I_1	lO_0p					I_1	l_hO_0p
2	I_0O_0p	$l_1a'_iI_{0,2}O_3O_1$					O_0I_0p	O_1
3	$I_{0,2}$	$l_2O_2O_0I_0p$		a'_i	a'_{m+1}	a_{m+2}		I_0O_0p
4	$I_2I_0O_0p$	l_3O_1		a_i	a_{m+1}	a_{m+2}	O_0p	
5	I_1	l'/l'' O_0I_0p	a_i		a_{m+1}	a_{m+2}	p	

Consider either of the theorems above. Remove from the construction all rules with a'_i or a_i, $1 \leq i \leq m$ on the left-hand side. Add rules $a'_i \to a_i$ to R_2. The output of the system is now internal: when it halts, one can consider objects a_i, $1 \leq i \leq m$ in the elementary membrane as a result (no other objects will be there). Let the superscript *int* stand for systems with internal output and let subscript *ne* mean that no rule uses the environment and the skin membrane.

Corollary 4.1 $PsProP^{1,int}_{2,ne}(ncoo, sym_1, anti_1) = PsRE$, $PsProP^{1,int}_{2,ne}(ncoo, sym_2) = PsRE$.

5 Bi-stable Catalysts

An interesting observation is that, interpreting the same object in different regions of the system as different objects in the same region (encoding regions in objects), one can easily see that the proton becomes a bi-stable catalyst. Let us explain this more formally.

Given a proton pumping P system with two membranes $\Pi = (O, P, [_1\ [_2\ \ ]_2\ ]_1, w_1, w_2, R_1, R_2, R'_1, R'_2)$ such that the communication rules are minimally cooperative (either symport rules of weight at most two and antiport rules of weight 1) and the only rules associated to the skin membrane

are the rules that output the terminal symbols, one can construct a P system with bi-stable catalysts in the following way:

$$
\begin{aligned}
\Pi' &= (O', C_b, [_1\ \ ]_1, w_1', R') \text{ where} \\
O' &= \{a, h(a) \mid a \in O\} \cup \{b_p \mid \{p, h(p)\} \in C_b\}, \\
C_b &= \{\{p, h(p)\} \mid p \in P\}, \\
w_1' &= h_b(w_1 h(w_2)), \\
R' &= R_1 \cup \{h(u) \to h(v) \mid (u \to v) \in R_2\} \cup R'', \\
R'' &= \{u \to u_{out} \mid (u, out) \in R_1'\} \cup \{h(u) \to u \mid (u, out) \in R_2'\} \\
&\cup \{u \to h(u) \mid (u, in) \in R_2'\} \cup \{h(u)v \to h(v)u \mid (u, out; v, in) \in R_2'\} \\
&\cup \{h(p)b_p \to pb_p \mid (p, out) \mid R_2',\ p \in P\} \\
&\cup \{pb_p \to h(p)b_p \mid (p, in) \mid R_2',\ p \in P\},
\end{aligned}
$$

where $h : O \to \{a' \mid a \in O\}$ and $h_b : O \to O^*$ are morphisms defined by $h(a) = a'$ for every $a \in O$, $h(a) = a$ for $a \in O - \{p, p' \mid p \in P\}$, $h(p) = pb_p$ for $p \in P$, and $h(p') = p'b_p$: h is the priming morphism for objects of region 2, and h_b is the morphism adding objects b_p to objects p or p'.

It is easy too see that the behavior of Π' is exactly the same as that of Π: the objects in region 1 of Π are also in Π', while the objects in region 2 of Π are renamed (i.e., primed) and also placed in region 1 of Π, and the rules are changed accordingly. The role of extra objects b_p (one copy for every copy of bi-catalytic symbols in w_1 and w_2) is to transform all non-cooperative proton rules in cooperative bi-stable catalytic rules (because rules $p \to p'$ or $p' \to p$, $\{p, p'\} \in C_b$, are forbidden by the definition of P system with bi-stable catalysts).

Clearly, non-cooperative rules (except the uniport of protons) remain non-cooperative, while other rules are changed as follows:

In Π	(pa, out)	(pa, in)	$(p, out; a, in)$	$(a, out; p, in)$
In Π'	$p'a' \to pa$	$pa \to p'a'$	$p'a \to pa'$	$pa' \to p'a$

In Π	(p, out)	(p, in)
In Π'	$p'b_p \to pb_p$	$pb_p \to p'b_p$

We can now claim that during this transformation the proton pumping computational completeness constructions become the computational completeness constructions of P systems with (the same number as protons in the original construction) bi-stable catalysts.

Example 5.1 *Transformed time-free P system from Corollary 3.1 to Theorem 3.2 (extra objects are not needed: the construction does not have uniport*

rules of protons).

$$
\begin{aligned}
\Pi &= (O, C_b, [_1\ \ ]_1, w_1, R_1), \text{ where} \\
O &= \{a_i, a_i' \mid 1 \le i \le m+2\} \cup \{l_j, l_j' \mid l \in I_-, 1 \le j \le 4\} \\
&\cup \{\#_1, \#_2, \#_1', \#_2'\} \cup \{l, l' \mid l \in I\} \cup P, \\
C_b &= \{\{D_i, D_i'\}, \{E_i, E_i'\} \mid i \in W\}, \\
w_1 &= l_0 \#_1 D_{m+1}' D_{m+2}' Z_{m+1}' Z_{m+2}', \\
R_1 &= \{l \to (a_i)_{out} l^{(1)}, l \to (a_i)_{out} l^{(2)} \mid l : (A(i), l^{(1)}, l^{(2)}) \in I,\ 1 \le i \le m\} \\
&\cup \{l \to a_i l^{(1)}, l \to a_i l^{(2)} \mid l : (A(i), l^{(1)}, l^{(2)}) \in I, i \in W\} \cup \{\#_2 \to \#_2\} \\
&\cup \{l_4 \to l^{(1)}, l_1 \to l_2, l_2 \to \#_2, l_3 \to l^{(2)} \mid l : (S(i), l^{(1)}, l^{(2)}) \in I\}, \\
&\cup \{l' \to l_4', l' \to l_1', l_2' \to l_3' \mid l : (S(i), l^{(1)}, l^{(2)}) \in I\} \cup \{\#_1' \to \#_1'\}, \\
&\cup \{l \to l', l_4' D_i' \to l_4 D_i, a_i D_i \to a_i' D_i', \#_1' D_i' \to \#_1 D_i, l_1' E_i' \to l_1 E_i, \\
&\quad a_i E_i \to a_i' E_i', l_2 E_i \to l_2' E_i', l_3' \to l_3 \mid l : (S(i), l^{(1)}, l^{(2)}) \in I\}.
\end{aligned}
$$

Thus we obtain a (clearly, optimal) computational completeness result for systems with one bi-stable catalyst: $LOP_1(2cat_1, tar) = RE$, improving $NOP_5(cat_2, 2cat_1, tar) = NRE$ from [11]. Another new result (see the example above) is that time-free systems with four bi-stable catalysts are computationally complete: $fLOP_1(2cat_4, tar) = RE$ (improving $fPsOP_1(2cat_*, tar) = PsRE$ from [6]).

6 Concluding Remarks

We have studied proton pumping P systems, a variant of P systems which is both biologically motivated and mathematically elegant. The obtained results are then transferred to P systems with bi-stable catalysts. Since every object only carries a finite amount of information, the cooperation of objects (i.e., the exchange of information) is crucial to obtain any non-trivial computational device. Here, the cooperation is reduced to the minimum: objects can only cooperate directly with protons, by moving together to another region, or with bi-stable catalysts, by changing their state.

Nevertheless, this is enough to reach computational completeness, even with low parameters like rewriting objects in **two regions** and communicating them across **one membrane** using just **one proton**, **or** rewriting objects in **one region** using just **one bi-stable catalyst**. The latter result nicely correlates with the computational completeness of P systems with two catalysts, [9]. The same systems are computationally complete in a **time-free** way with **four protons/bi-stable catalysts** instead of one.

Yet another point worth mentioning is that the constructions in the proofs have a low number of **cooperative rules: four** for both one-proton constructions and $|\mathbf{I}_+| + \mathbf{2}|\mathbf{I}_-| + \mathbf{6}$ (where I_+ is the number of ADD instructions and I_- is the number of SUB instructions in the simulated register machine) for both time-free constructions (exactly the same results can be claimed for P system with bi-stable catalysts).

The one-proton results obtained here are optimal for P systems with external output in terms of number of membranes and protons, assuming that the skin membrane is only used to output the result: with only one membrane (i.e., output membrane) or zero protons the behavior of the system is non-cooperative. However, some challenging open problems remain:

- Is rewriting in both regions necessary for completeness (most of the constructions in evolution–communication and proton pumping P systems heavily rely on rewriting in all regions)?
- What is the generative power of proton pumping P systems with one membrane and internal output?
- What about restricted proton pumping P systems, where the only uniport rules allowed are uniport rules of protons (i.e., protons appear in no evolution rules but in all communication rules)?
- Are four protons (or bi-stable catalysts) necessary for time-free computational completeness?

Acknowledgements The author is thankful to Francesco Bernardini for suggesting to use the register machines to show that four protons are enough, and to Rudolf Freund for the useful discussions of the results on catalytic P systems. The paper was written during the author's visit of the Vienna University of Technology.

The author is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona, and acknowledges the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034 for providing a challenging and fruitful framework for cooperation.

References

[1] B. Alberts et al.: *Essential Cell Biology, An Introduction to the Molecular Biology of the Cell*. Garland Publ, New York, London (1998).

[2] A. Alhazov: Minimizing Evolution-Communication P Systems and Automata. In: [7], 23–31, and *New Generation Computing* **22**, 4 (2004), 299–310.

[3] A. Alhazov, M. Cavaliere: Evolution-communication P systems: Time-freeness. In: M.A. Gutiérrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan (Eds.): *Proceedings of the Third Brainstorming Week on Membrane Computing*. Technical Report **01/2005**, Sevilla University, Sevilla (2005), 11–18.

[4] A. Alhazov, M. Cavaliere: Proton Pumping P Systems. In: A. Alhazov, C. Martín-Vide, Gh. Păun (Eds.): *Preproceedings of the Workshop on Membrane Computing*. Technical Report **28/03**, Rovira i Virgili University, Tarragona (2003), 1–16, and in: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, International Workshop, WMC 2003, Tarragona, 2003, Revised Papers. *Lecture Notes in Computer Science* **2933**, Springer (2004), 1–18.

[5] M. Cavaliere: Evolution-Communication P Systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Argeş. *Lecture Notes in Computer Science* **2597**, Springer-Verlag, Berlin (2003), 134–145.

[6] M. Cavaliere, V. Deufemia: Further results on time-free P systems. In: M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez (Eds.): *Proceedings of the ESF PESC Exploratory Workshop on Cellular Computing (Complexity Aspects)*. Fénix Editora, Sevilla (2005), 95–116.

[7] M. Cavaliere, C. Martín-Vide, Gh. Păun (Eds.): *Brainstorming Week on Membrane Computing*. Technical Report **26/03**, Rovira i Virgili University, Tarragona (2003).

[8] M. Cavaliere, D. Sburlan: Time-independent P systems. In: G. Mauri, Gh. Paun, M.J. Peréz-Jiménez, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, *International Workshop WMC5*, Milano, Italy, 2004, Selected Papers. *Lecture Notes in Computer Science* **3365**, Springer (2005).

[9] R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science* **330**, 2 (2005), 251–266.

[10] S.N. Krishna, A. Păun: Some Universality Results on Evolution-Communication P Systems. In [7], 207–215, and *New Generation Computing* **22**, 4 (2004), 377–394.

[11] S.N. Krishna, A. Păun: Three Universality Results on P Systems. In [7], 198–206.

[12] M.L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey (1967).

[13] Gh. Păun: *Computing with Membranes: An Introduction*. Springer-Verlag, Berlin (2002).

[14] M.H. Saier, jr.: A Functional-Phylogenetic Classification System for Transmembrane Solute Transporters. *Microbiology and Molecular Biology Reviews* (2000), 354–411.

[15] The P systems Web Page: `http://psystems.disco.unimib.it`

Symbol / Membrane Complexity of P Systems with Symport / Antiport Rules

Artiom ALHAZOV[1,2], **Rudolf FREUND**[3], **Marion OSWALD**[3]

[1] Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
E-mail: artiome.alhazov@estudiants.urv.es

[2] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD 2028, Moldova
E-mail: artiom@math.md

[3] Faculty of Informatics, Vienna University of Technology
Favoritenstr. 9–11, A–1040 Vienna, Austria
E-mail: {rudi,marion}@emcc.at

Abstract

We consider P systems with symport / antiport rules and small numbers of symbols and membranes and present several results for P systems with symport / antiport rules simulating register machines with the number of registers depending on the number s of symbols and the number m of membranes. For instance, any recursively enumerable set of natural numbers can be generated (accepted) by systems with $s \geq 2$ symbols and $m \geq 1$ membranes such that $m + s \geq 6$. In particular, the result of the original paper [16] proving universality for three symbols and four membranes is improved (e.g., three symbols and three membranes are sufficient). The general results that P systems with symport / antiport rules with s symbols and m membranes are able to simulate register machines with $\max\{m(s-2), (m-1)(s-1)\}$ registers also allows us to give upper bounds for the numbers s and m needed to generate/accept any recursively enumerable set of k-dimensional vectors of non-negative integers or to compute any partial recursive function $f : \mathbb{N}^{\alpha} \to \mathbb{N}^{\beta}$. Finally, we also study the computational power of P systems with symport / antiport rules and only

one symbol: with one membrane, we can exactly generate the family of finite sets of non-negative integers; with one symbol and two membranes, we can generate at least all semilinear sets. The most interesting open question is whether P systems with symport / antiport rules and only one symbol can gain computational completeness (even with an arbitrary number of membranes) as it was shown for tissue P systems in [1].

1 Introduction

In the area of membrane computing there are two main classes of systems: P systems with a hierarchical (tree-like) structure as already introduced in the original paper of Gheorghe Păun (see [14]) and tissue P systems with cells arranged in an arbitrary graph structure (see [11], [9]). We here consider "classical" P systems using symport / antiport rules for the communication through membranes (these communication rules first were investigated in [13]).

It is well known that equipped with the maximally parallel derivation mode P systems / tissue P systems with only one membrane / one cell already reach universal computational power, even with antiport rules of weight two (e.g., see [4] and [8]); yet on the other hand, in these P systems the number of symbols remains unbounded.

Considering the generation of recursively enumerable sets of natural numbers we may also ask the question how many symbols we need for obtaining computational completeness in a small number of membranes. In [16] the quite surprising result was proved that three symbols are enough in the case of P systems with symport / antiport rules. The specific type of maximally parallel application of at most one rule in each connection (link) between two cells or one cell and the environment, respectively, in tissue P systems allowed for an even more surprising result proved in [10]: The minimal number of one symbol is already sufficient to obtain computational completeness, e.g., it was shown that any recursively enumerable set of natural numbers can be generated by a tissue P system with at most seven cells using symport / antiport rules of only one symbol. The question remained open whether such a result for the minimal number of symbols can also be obtained for "classical" P systems with symport / antiport rules.

The study of the computational power of tissue P systems depending on the number of cells and symbols was continued in [1]; many classes of these tissue P systems characterize the class of recursively enumerable sets of natural numbers, and some of them were shown to characterize or at

least to include the families of finite and regular sets of natural numbers, respectively.

In this paper we continue the direction of [2] and consider "classical" P systems with symport / antiport rules simulating register machines with the number of registers depending on the number s of symbols and the number m of membranes. After some definitions in Sections 2 and 3, in Subsection 3.1, we show that P systems with one symbol and one membrane can exactly generate the family of finite sets of non-negative integers. In Subsections 3.2 and 3.3, some general results for the simulation of register machines by P systems with symport / antiport rules with s symbols and m membranes that allow us to give upper bounds for the numbers s and m needed to generate/accept any recursively enumerable set of vectors of non-negative integers or to compute any partial recursive function are elaborated: We show that any recursively enumerable set of natural numbers can be generated (accepted) by systems with $s \geq 2$ symbols and $m \geq 1$ membranes such that $m + s \geq 6$. In particular, the result of the original paper [16] proving universality for three symbols and four membranes is improved (i.e., three symbols and three membranes or two symbols and four membranes are shown to be sufficient). Finally, in Subsection 3.4 we show that P systems with symport / antiport rules with one symbol and two membranes can generate at least all semilinear (i.e., regular) sets of natural numbers. A summary of the obtained results and some open questions conclude the paper.

2 Preliminaries

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [3] and [18]. We just list a few notions and notations: $\mathbb{N}$ denotes the set of non-negative integers (natural numbers). V^* is the free monoid generated by the alphabet V under the operation of concatenation and the empty string, denoted by λ, as unit element; by RE ($RE(k)$) we denote the family of recursively enumerable languages (over a k-letter alphabet). By $\Psi_T(L)$ we denote the Parikh image of the language $L \subseteq T^*$, and by $PsFL$ we denote the set of Parikh images of languages from a given family FL. $PsRE(k)$ corresponds with the family of recursively enumerable sets of k-dimensional vectors of non-negative integers; for $PsRE(1)$ we also write NRE. N_lREG denotes the family of regular sets of numbers not containing any number smaller than l; if $l = 0$ we simply write $NREG$. $NFIN$ denotes the family of finite

sets of natural numbers.

2.1 Register Machines

The proofs of the main results established in this paper are based on the simulation of register machines; we refer to [12] for original definitions, and to [4] for definitions like those we use in this paper:

An *n-register machine* is a construct $M = (n, R, l_0, l_h)$, where n is the number of registers, R is a finite set of instructions injectively labelled with elements from a given set $lab(M)$, l_0 is the initial/start label, and l_h is the final label.

The instructions are of the following forms:

- $l_1 : (A(r), l_2, l_3)$,
  Add 1 to the contents of register r and proceed to one of the instructions (labelled with) l_2 and l_3. (We say that we have an ADD instruction.)

- $l_1 : (S(r), l_2, l_3)$,
  If register r is not empty, then subtract 1 from its contents and go to instruction l_2, otherwise proceed to instruction l_3. (We say that we have a SUB instruction.)

- $l_h : halt$,
  Stop the machine. The final label l_h is only assigned to this instruction.

(Deterministic) register machines can be used to compute any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$; starting with $(n_1, ..., n_\alpha) \in \mathbb{N}^\alpha$ in registers 1 to α, M has computed $f(n_1, ..., n_\alpha) = (r_1, ..., r_\beta)$ if it halts in the final label h with registers 1 to β containing r_1 to r_β. If the final label cannot be reached, $f(n_1, ..., n_\alpha)$ remains undefined.

A deterministic register machine can also analyze an input $(n_1, ..., n_\alpha) \in \mathbb{N}^\alpha$ in registers 1 to α, which is recognized if the register machine finally stops by the halt instruction with all its registers being empty. If the machine does not halt, the analysis was not successful.

A (non-deterministic) register machine M is said to generate a vector $(s_1, \ldots, s_k)$ of natural numbers if, starting with the instruction with label l_0 and all registers containing the number 0, the machine stops (it reaches the instruction $l_h : halt$) with the first k registers containing the numbers $s_1, \ldots, s_k$ (and all other registers being empty).

Without loss of generality, in the succeeding proofs we will assume that in each ADD instruction $l_1 : (A(r), l_2, l_3)$ and in each SUB instruction $l_1 : (S(r), l_2, l_3)$ the labels l_1, l_2, l_3 are mutually distinct (for a short proof see [9]).

The register machines are known to be computationally complete, equal in power to (non-deterministic) Turing machines: they generate exactly the sets of vectors of natural numbers which can be generated by Turing machines, i.e., the family $PsRE$.

The results proved in [5] (based on the results established in [12]) as well as in [6] and [7] immediately lead to the following results:

Proposition 2.1 *For any partial recursive function $f : \mathbb{N}^\alpha \to \mathbb{N}^\beta$ there exists a deterministic $(\max\{\alpha, \beta\} + 2)$-register machine M computing f in such a way that, when starting with $(n_1, ..., n_\alpha) \in \mathbb{N}^\alpha$ in registers 1 to α, M has computed $f(n_1, ..., n_\alpha) = (r_1, ..., r_\beta)$ if it halts in the final label h with registers 1 to β containing r_1 to r_β, and all other registers being empty; if the final label cannot be reached, $f(n_1, ..., n_\alpha)$ remains undefined.*

In particular we know that $k+2$-register machines generate/ accept any recursively enumerable set of k-dimensional vectors of non-negative integers (see [4], [12]):

Proposition 2.2 *For any recursively enumerable set $L \subseteq \mathbb{N}^\beta$ of vectors of non-negative integers there exists a non-deterministic $(\beta + 2)$-register machine M generating L in such a way that, when starting with all registers 1 to $\beta + 2$ being empty, M non-deterministically computes and halts with n_i in registers i, $1 \leq i \leq \beta$, and registers $\beta + 1$ and $\beta + 2$ being empty if and only if $(n_1, ..., n_\beta) \in L$.*

Proposition 2.3 *For any recursively enumerable set $L \subseteq \mathbb{N}^\alpha$ of vectors of non-negative integers there exists a deterministic $(\alpha + 2)$-register machine M accepting L in such a way that M halts with all registers being empty if and only if M starts with some $(n_1, ..., n_\alpha) \in L$ in registers 1 to α and the registers $\alpha + 1$ to $\alpha + 2$ being empty.*

From the main result in [12] that the actions of a Turing machine can be simulated by a 2-register machine (using a prime number encoding of the configuration of the Turing machine) we also know that the halting problem is undecidable for 2-register machines.

Moreover, it is well-known that 1-register machines can generate/ accept $NREG$.

2.2 P Systems with Symport / Antiport Rules

The reader is supposed to be familiar with basic elements of membrane computing, e.g., from [15]; comprehensive information can be found on the P systems web page `http://psystems.disco.unimib.it`.

A P system (of degree $m \geq 1$) *with symport / antiport rules* (in the following we shall only speak of a *P system*) is a construct

$$\Pi = (O, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m),$$

where

- O is the alphabet of *objects*,
- μ is the *membrane structure* (it is assumed that we have m membranes, labelled with $1, 2, \ldots, m$, the skin membrane usually being labelled with 1),
- w_i, $1 \leq i \leq m$, are strings over O representing the *initial* multiset of *objects* present in the membranes of the system,
- R_i, $1 \leq i \leq m$, are finite sets of *symport / antiport rules* of the form x/y, for some $x, y \in O^*$, associated with membrane i (if $|x|$ or $|y|$ equals 0 then we speak of a symport rule, otherwise we call it an antiport rule).

An antiport rule of the form $x/y \in R_i$ means moving the objects specified by x from membrane i to the surrounding membrane j (to the environment, if $i = 1$), at the same time moving the objects specified by y in the opposite direction. (The rules with one of x, y being empty are, in fact, symport rules, but in the following we do not explicitly consider this distinction here, as it is not relevant for what follows.) We assume the environment to contain all objects in an unbounded number.

The computation starts with the multisets specified by $w_1, \ldots, w_m$ in the m membranes; in each time unit, the rules assigned to each membrane are used in a maximally parallel way, i.e., we choose a multiset of rules at each membrane in such a way that, after identifying objects inside and outside the corresponding membranes to be affected by the selected multiset of rules, no objects remain to be subject to any additional rule at any membrane. The computation is successful if and only if it halts; depending on the function of the system, the input and the output may be encoded by

different symbols in different membranes, the input then being added in the initial configuration as the corresponding number of respective symbols in the designated membranes.

The set of all k-dimensional vectors generated/ accepted in this way by the system Π is denoted by $g(k) N(\Pi)$ and $a(k) N(\Pi)$, respectively. The family of sets $g(k) N(\Pi)$/ $a(k) N(\Pi)$ of vectors computed as above by systems with at most m membranes and at most s symbols is denoted by $g(k) NO_s P_m$ and $a(k) NO_s P_m$, respectively. The family of functions from k-dimensional vectors to l-dimensional vectors computed as above by P systems with at most m membranes and at most s symbols is denoted by $f(k,l) NO_s P_m$. When any of the parameters k, l, m, s is not bounded, it is replaced by $*$.

3 Results

We now establish our results for P systems with symport/ antiport rules and small numbers of membranes and symbols. The main constructions show that a P system with symport/ antiport rules and $m \geq 1$ membranes as well as $s \geq 2$ symbols can simulate a register machine with $\max\{m(s-2), (m-1)(s-1)\}$ registers. For example, in that way we improve the result $NRE = g(1) NO_3 P_4$ as established in [16] to $NRE = g(1) NO_3 P_3 = g(1) NO_2 P_4$.

3.1 One Membrane

The following characterization of $NFIN$ by P systems with only one membrane and only one symbol corresponds with the similar characterization by tissue P systems with only one cell and only one symbol as established in [1].

Example 3.1 $g(1) NO_1 P_1 = NFIN$.

Consider an arbitrary non-empty set $M \in NFIN$. Then we construct a P system $\Pi = (\{a\}, [_1\ ]_1, w_1, R_1)$ where $w_1 = a^m$ with $m = \max(M) + 1$ and $R_1 = \{a^m/a^j \mid j \in M\}$.

Clearly, $j < m$ for any $j \in M$, so the computation finishes in one step generating the elements of M as the corresponding number of symbols a in the skin membrane.

The special case of generating the empty set can be done by the following trivial P system: $\Pi = (\{a\}, [_1\ ]_1, a, \{a/a\})$. A computation in this system will never halt.

The inclusion $NFIN \supseteq g(1)\, NO_1P_1$ *can easily be argued (like in [1]) as follows:*

Consider a P system $\Pi = (\{a\}, [_1\ ]_1, w_1, R_1)$.

Let $m = \min\{j \mid j/i \in R_1 \text{ for some } i\}$. *Then a rule from* R_1 *can be applied as long as region* 1 *contains at least* m *objects. Therefore,* $g(1)\, N(\Pi) \subseteq \{j \mid j < m\}$; *hence,* $g(1)\, N(\Pi) \in NFIN$.

Let us recall another relatively simple construction for tissue P systems from [1] that also shows a corresponding result for the membrane case.

Example 3.2 $g(1)\, NO_2P_1 \supseteq NREG$.

We will use the fact that for any regular set M *of nonnegative integers there exist finite sets of numbers* M_0, M_1 *and a number* k *such that* $M = M_0 \cup \{i + jk \mid i \in M_1, j \in \mathbb{N}\}$ *(this follows, e.g., from the shape of the minimal finite automaton accepting the unary language with length set* M*).*

We now construct a P system $\Pi = (\{a, p\}, [_1\ ]_1, w_1, R_1)$ *where* $w_1 = pp$ *and* $R_1 = \{pp/a^i \mid i \in M_0\} \cup \{pp/pa, pa/pa^{k+1}\} \cup \{pa/a^i \mid i \in M_1\}$, *which generates* M *as the number of symbols* a *in the skin membrane in halting computations.*

Initially, there are no objects a *in region* 1, *so the system "chooses" between generating an element of* M_0 *in one step or exchanging* pp *by* pa. *In the latter case, there is only one copy of* p *in the system. After an arbitrary number* j *of applications of the rule* pa/pa^{k+1} *a rule exchanging* pa *by* a^i *for some* $i \in M_1$ *is eventually applied, generating* $jk + i$ *symbols* a. *Hence,* $g(1)\, N(\Pi) = M_0 \cup \{i + jk \mid i \in M_1, j \in \mathbb{N}\} = M$.

We will now show two simple constructions to illustrate the accepting power of P systems with one membrane.

Example 3.3 $\{ki \mid i \in \mathbb{N}\} \in a(1)\, NO_1P_1$ *for any* $k \in \mathbb{N}$.

The set of numbers divisible by a fixed number k *(represented by the multiplicity of the object* a *in the initial configuration) can be accepted by the P system* $\Pi = (\{a\}, [_1\ ]_1, w_1, \{a^k/\lambda, a/a\})$; w_1 *is the input of the P system in the initial configuration. The rule* a^k/λ *sends objects out in groups of* k, *while the rule* a/a *"keeps busy" all objects not used by the other one. Hence, the system halts if and only if a multiple of* k *symbols* a *has been sent out in several steps finally not using the antiport rule* a/a *anymore.*

Example 3.4 $NFIN \subseteq a(1)\, NO_2P_1$.

Any finite set M *of natural numbers (represented by the multiplicity of the object* a *in the initial configuration) can be accepted by the P system*

$\Pi = (\{a, p\}, [_1\ ]_1, pw_1, \{a/a, p/p\} \cup \{pa^n/\lambda \mid n \in M\})$; *$w_1$ is the input of the P system in the initial configuration as the number of symbols a in the skin membrane representing the corresponding element from M. The rule pa^n/λ can send out p together with a "correct" number of objects a, while the rules a/a and p/p (in the case of $w_1 = \lambda$) "keep busy" all other objects.*

Example 3.3 illustrates that even P systems with one membrane and one object can accept some infinite sets (as opposed to the generating case, where we exactly get all finite sets). Example 3.4 shows that when using two objects it is already possible to accept all finite sets.

3.2 At Least Three Symbols

It was already shown in [2] that any d-register machine can be simulated by a P system in one membrane using $d + 2$ symbols. In this subsection we generalize this result: P systems with m membranes and $s \geq 3$ symbols can simulate $m(s-2)$-register machines:

Theorem 3.1 *Any mn-register machine can be simulated by a P system with $2 + n$ symbols and m membranes.*

Proof. Let us consider a register machine $M = (d, R, l_1, l_{halt})$ with $d = mn$ registers. No matter what the goal of M is (generating / accepting vectors of natural numbers, computing functions), we can construct the P system (of degree m)

$$\begin{aligned}
\Pi &= (O, \mu, w_1, \cdots, w_m, R_1, \cdots, R_m), \\
O &= \{p, q\} \cup \{a_j \mid 1 \leq j \leq n\}, \\
\mu &= [_1\ [_2\ ]_2 \cdots [_m\ ]_m\ ]_1, \\
w_1 &= w_0 \textstyle\prod_{j=1}^{n} a_j^{r_j}, \\
w_i &= \textstyle\prod_{j=1}^{n} a_j^{r_{j+(i-1)n}},\ 2 \leq i \leq m,
\end{aligned}$$

that simulates the actions of M as follows. The symbols p and q are needed for encoding the instructions of M; q also has the function of a trap symbol, i.e., in case of the wrong choice for a rule to be applied we take in so many symbols q that we can never again rid of them and therefore get "trapped" in an infinite loop. Throughout the computation, the value of register $j + (i-1)n$ is represented by the multiplicity of symbol a_j in region i. In the generating case, $w_1 = w_0$ and $w_i = \lambda$ for $2 \leq i \leq m$; in the accepting case and in the case of computing functions, the numbers of symbols a_j as defined above specify the input.

An important part of the proof is to define a suitable encoding $c : \mathbb{N} \to \mathbb{N}$ (a strictly monotone linear function) for the instructions of the register machine: As we will use at most 6 different subsequent labels for each instruction, without loss of generality we assume the labels of M to be positive integers such that the labels assigned to ADD and SUB instructions have the values $6i-5$ for $1 \leq i < t$, as well as $l_0 = 1$ and $l_{halt} = 6\,(t-1)+1$, for some $t \geq 1$.

For the operations assigned to a label l and working on register r, we will use specific encodings by the symbols p and q which allow us to distinguish between the operations ADD, SUBTRACT, and ZERO TEST. As we have d registers, this yields $3d$ multisets for specifying operations. The number of symbols p and q in these operation multisets is taken in such a way that the number of symbols p always exceeds the number of symbols q. Finally, the number of symbols q can never be split into two parts that could be interpreted as belonging to two operation multisets.

Hence, the range for the number of symbols q is taken as the interval $[3d+1, 6d]$ and the range for the number of symbols p is taken as the interval $[6d+1, 9d+1]$. Thus, with $h = 12d+1$ we define the following operation multisets:

$$\begin{array}{ll} ADD: & \alpha_+(r) = q^{3d+r}p^{h-(3d+r)},\ 1 \leq r \leq d, \\ SUBTRACT: & \alpha_-(r) = q^{4d+r}p^{h-(4d+r)},\ 1 \leq r \leq d, \\ ZEROTEST: & \alpha_0(r) = q^{5d+r}p^{h-(5d+r)},\ 1 \leq r \leq d. \end{array}$$

The encoding $c : \mathbb{N} \to \mathbb{N}$ which shall encode the instruction l of M to be simulated as $p^{c(l)}$ also has to obey to the following conditions:

- For any i, j with $1 \leq i, j \leq 6t-5$, $c\,(i) + c\,(j) > c\,(6t-4)$, i.e., the sum of the codes of two instruction labels has to be larger than the largest code we will ever use for the given M, hence, if we do not use the maximal number of symbols p as interpretation of a code for an instruction (label), then the remaining rest of symbols p cannot be misinterpreted as the code for another instruction label.

- The distance g between any two codes $c\,(i)$ and $c\,(i+1)$ has to be larger than any of the multiplicities of the symbol p which appear besides codes in the rules defined above.

As we shall see in the construction of the rules below, we may take

$$g = 2h = 24d + 2.$$

In sum, for a function c fulfilling all the conditions stated above we can take

$$c\,(x) = g(x + 6t - 4) \text{ for } x \geq 0.$$

For example, with this function, for arbitrary $i, j \geq 1$ we get

$$c\,(i)+c\,(j) = g(i+6t-4)+g(j+6t-4) > g(6t-4+6t-4) = c\,(6t-4)\,.$$

Moreover, for $l_1 = 1$ we therefore obtain

$$c\,(l_1) = g\,(6t-3) = (24d+2)\,(6t-3)$$

as well as

$$w_0 = p^{c(l_1)} = p^{(24d+2)(6t-3)}.$$

Finally, we have to find a number f which is so large that after getting f symbols we inevitably enter an infinite loop with the rule

$$q^f / q^{3f};$$

as we shall justify below, we can take

$$f = c\,(l_{halt} + 1) = 2g(6t-4).$$

Equipped with this coding function and the constants defined above we are now able to define the following set of symport / antiport rules assigned to the membranes for simulating the actions of the given register machine M :

$$
\begin{aligned}
R_1 &= \{p^{c(l_1)}/p^{c(l_2)}a_s, p^{c(l_1)}/p^{c(l_3)}a_s \mid \\
&\quad l_1 : (A(s), l_2, l_3) \in R, 1 \le s \le n\} \\
&\cup \{p^{c(l_1)}/p^{c(l_1+1)}\alpha_+(s+(s'-1)n)a_s, p^{c(l_1+1)}/p^{c(l_1+2)}, \\
&\quad p^{c(l_1+2)}/p^{c(l_1+3)}, p^{c(l_1+3)}\alpha_+(s+(s'-1)n)/p^{c(l_2)} \\
&\quad p^{c(l_1+3)}\alpha_+(s+(s'-1)n)/p^{c(l_3)} \mid \\
&\quad l_1 : (A(s+(s'-1)n), l_2, l_3), 1 \le s \le n, 2 \le s' \le m\} \\
&\cup \{p^{c(l_1)}a_s/p^{c(l_2)}, p^{c(l_1)}/p^{c(l_1+1)}\alpha_0(s), \\
&\quad p^{c(l_1+1)}/p^{c(l_1+2)}, \alpha_0(s)a_s/q^{3f}, \\
&\quad p^{c(l_1+2)}\alpha_0(s)/p^{c(l_3)} \mid l_1 : (S(s), l_2, l_3) \in R, 1 \le s \le n\} \\
&\cup \{p^{c(l_1)}/p^{c(l_1+1)}\alpha_-(s+(s'-1)n), p^{c(l_1+1)}/p^{c(l_1+2)}, \\
&\quad p^{c(l_1+2)}/p^{c(l_1+3)}, p^{c(l_1+3)}\alpha_-(s+(s'-1)n)a_s/p^{c(l_2)}, \\
&\quad p^{c(l_1)}/p^{c(l_1+4)}\alpha_0(s+(s'-1)n), p^{c(l_1+4)}/p^{c(l_1+5)}, \\
&\quad p^{c(l_1+5)}\alpha_0(s+(s'-1)n)/p^{c(l_3)}, \\
&\quad \alpha_-(s+(s'-1)n)/q^{3f} \mid \\
&\quad l_1 : (S(s+(s'-1)n), l_2, l_3), 1 \le s \le n, 2 \le s' \le m\} \\
&\cup \{p^{c(l_{halt})}/\lambda, p^h/q^{3f}, q^f/q^{3f}\}
\end{aligned}
$$

as well as for $2 \le s' \le m$

$$
\begin{aligned}
R_{s'} &= \{\lambda/\alpha_+(s+(s'-1)n)a_s, \alpha_+(s+(s'-1)n)/\lambda \mid \\
&\quad l_1 : (A(s+(s'-1)n), l_2, l_3), 1 \le s \le n, 2 \le s' \le m\} \\
&\cup \{a_s/\alpha_-(s+(s'-1)n), \alpha_-(s+(s'-1)n)/\lambda, \\
&\quad a_s/\alpha_0(s+(s'-1)n) \mid \\
&\quad l_1 : (S(s+(s'-1)n), l_2, l_3), 1 \le s \le n, 2 \le s' \le m\}
\end{aligned}
$$

The correct work of the rules can be described as follows:

1. Throughout the whole computation in Π, it is directed by the code $p^{c(l)}$ for some $l \le 6t-5$; in order to guarantee the correct sequence of encoded rules the trap is activated in case of a wrong choice, which in any case guarantees an infinite loop with the symbols q by the "trap rule"

   q^f/q^{3f}.

   The minimal number of superfluous symbols p to start the trap is h and causes the application of the rule p^h/q^{3f}.

2. For each ADD instruction $l_1 : (A(s), l_2, l_3)$ of M, i.e., for incrementing register s for $1 \le s \le n$, we use the following rules in R_1:

   $p^{c(l_1)}/p^{c(l_2)}a_s$, and

$p^{c(l_1)}/p^{c(l_3)}a_s$.

In that way, the ADD instruction $l_1 : (A(s), l_2, l_3)$ of M for one of the first n registers is simulated in only one step: the number of symbols p representing the instruction of M labelled by l_1 is replaced by the number of symbols p representing the instruction of M labelled by l_2 or l_3, respectively, in the same moment also incrementing the number of symbols a_s. Whenever a wrong number of symbols p is taken, the remaining symbols cannot be used by another rule than the "trap rule" p^h/q^{3f}, which in the succeeding computation steps inevitably leads to the repeated application of the rule q^f/q^{3f} thus flooding the skin membrane with more and more symbols q.
On the other hand, incrementing register $s + (s' - 1)n$, for $1 \leq s \leq n$, $2 \leq s' \leq m$, i.e., registers $n + 1$ to nm is accomplished by the rules

$p^{c(l_1)}/p^{c(l_1+1)}\alpha_+(s + (s' - 1)n)a_s$,
$p^{c(l_1+1)}/p^{c(l_1+2)}$
$p^{c(l_1+2)}/p^{c(l_1+3)}$,
$p^{c(l_1+3)}\alpha_+(s + (s' - 1)n)/p^{c(l_2)}$
$p^{c(l_1+3)}\alpha_+(s + (s' - 1)n)/p^{c(l_3)}$

in R_1 as well as by the rules

$\lambda/\alpha_+(s + (s' - 1)n)a_s$,
$\alpha_+(s + (s' - 1)n)/\lambda$ in $R_{s'}$

Hence, adding one to the contents of registers $n + 1$ to nm now needs four steps: the number of symbols p representing the instruction of M labelled by l_1 is replaced by $p^{c(l_1+1)}$ together with $3d + s + (s' - 1)n$ additional symbols q, $h - (3d + s + (s' - 1)n)$ symbols p and the symbol a_s. In the second step, $p^{c(l_1+1)}$ is exchanged with $p^{c(l_1+2)}$, while at the same time the additional $3d + s + (s' - 1)n$ symbols q and $h - (3d + s + (s' - 1)n)$ symbols p are introduced together with a_s in membrane s'. In the third step, the $c(l_1 + 2)$ symbols p in the skin membrane are exchanged with $c(l_1 + 2)$ symbols p from the environment, whereas the additional $3d + s + (s' - 1)n$ symbols q and $h - (3d + s + (s' - 1)n)$ symbols p pass out from membrane r. Finally, in the fourth step, these latter symbols together with $p^{c(l_1+3)}$ in the skin membrane are replaced by the number of symbols p representing the next instruction of M labelled by l_2 or l_3, respectively.

3. For simulating the decrementing step of a SUB instruction $l_1 : (S(s), l_2, l_3)$ from R we introduce the following rules:

   $p^{c(l_1)} a_s / p^{c(l_2)}$

   for decrementing the contents of register s, for $1 \leq s \leq n$, represented by the symbols a_s in the skin membrane.
   In that way, the decrementing step of the SUB instruction $l_1 : (S(s), l_2, l_3)$ of M now is also simulated in one step: together with $p^{c(l_1)}$ we send out one symbol a_s and take in $p^{c(l_2)}$, which encodes the label of the instruction that has to be executed after the successful decrementing of register s, for $1 \leq s \leq n$.
   For decrementing the registers $s+(s'-1)n$, for $1 \leq s \leq n$, $2 \leq s' \leq m$, we need the following rules:

   $p^{c(l_1)} / p^{c(l_1+1)} \alpha_-(s + (s'-1)n)$,
   $p^{c(l_1+1)} / p^{c(l_1+2)}$
   $p^{c(l_1+2)} / p^{c(l_1+3)}$,
   $p^{c(l_1+3)} \alpha_-(s + (s'-1)n) a_s / p^{c(l_2)}$ in R_1

   as well as

   $a_s / \alpha_-(s + (s'-1)n)$,
   $\alpha_-(s + (s'-1)n) / \lambda$ in R_r.

   In this case, the SUB instruction is simulated in four steps: $p^{c(l_1)}$ is replaced by $p^{c(l_1+1)}$ together with the "operation multiset" $\alpha_-(s+(s'-1)n)$, i.e., $q^{4d+r} p^{h-(4d+r)}$, $r = s + (s'-1)n$, for $1 \leq s \leq n$, $2 \leq s' \leq m$. While in the next two steps, two intermediate exchanges of symbols p with the environment take place, the symbol a_s is exchanged with $\alpha_-(s + (s'-1)n)$ in membrane r, that, in the third step, goes out again to the skin membrane, where it can now together with $p^{c(l_1+3)}$ be exchanged with $p^{c(l_2)}$, i.e., the representation of the next instruction of M.
   Again we notice that if we do not choose the correct rule, then the trap is activated by the rule p^h / q^{3f}, especially if no symbol a_s is present in membrane r, then we have to apply the "trap rule" $\alpha_-(s + (s'-1)n) / q^{3f}$.

4. For simulating the zero test, i.e., the case where we check the contents of register r to be zero, of a SUB instruction $l_1 : (S(s), l_2, l_3)$ from R for registers 1 to n we take the following rules:

   $p^{c(l_1)} / p^{c(l_1+1)} \alpha_0(s)$,

$p^{c(l_1+1)}/p^{c(l_1+2)}$, and

$p^{c(l_1+2)}\alpha_0(s)/p^{c(l_3)}$ in R_1.

If the rule $\alpha_0(s)a_s/q^{3f}$ from R_1 can be applied, then in the next step we cannot apply $p^{c(l_1+2)}\alpha_0(s)/p^{c(l_3)}$ from R_1, hence, only a rule using less than $c(l_1+2)$ symbols p can be used together with the "trap rule" p^h/q^{3f}.

For simulating the zero test, i.e., the case where we check the contents of register r to be zero, of a SUB instruction $l_1 : (S(s), l_2, l_3)$ from R for registers $n+1$ to nm we now take the following rules:

$p^{c(l_1)}/p^{c(l_1+4)}\alpha_0(s+(s'-1)n)$,

$p^{c(l_1+4)}/p^{c(l_1+5)}$, and

$p^{c(l_1+5)}\alpha_0(s+(s'-1)n)/p^{c(l_3)}$ in R_1.

If the rule $a_s/\alpha_0(s+(s'-1)n)$ from R_r can be applied, then in the next step we cannot apply $p^{c(l_1+5)}\alpha_0(s+(s'-1)n)/p^{c(l_3)}$ from R_1, hence, only a rule using less than $c(l_1+5)$ symbols p can be used together with the "trap rule" p^h/q^{3f}.

5. The number of symbols p never exceeds $c\,(l_{halt}) = 2g(6t-4)$ as long as the simulation of instructions from R works correctly. By definition, $f = c\,(l_{halt}+1) = 2g(6t-4)$, hence, there will be at least three times more symbols q in region 1 than symbols p in the system after having applied a "trap rule", thus introducing $3f$ symbols q. As by any rule in R_1, the number of symbols p coming in is less than double the number sent out, the total number of symbols p in the system, in one computation step, can at most be doubled in total, too. As every rule that removes some symbols q from region 1 involves at least as many symbols p as symbols q, the "trap rule" q^f/q^{3f} guarantees that in the succeeding steps this relation will still hold true, no matter how the present symbols p and q are interpreted for rules in Π. Therefore, if as soon as a "trap rule" has been applied, then the number of objects q will grow and the system will never halt.

6. Finally, for the halt label $l_{halt} = 6t-5$ we only take the rule

   $p^{c(l_{halt})}/\lambda$,

   hence, the work of Π will stop exactly when the work of M stops (provided the trap has not been activated due to a wrong non-deterministic choice during the computation).

From the explanations given above we conclude that Π halts if and only if M halts, and moreover, the final configuration of Π represents the final contents of the registers in M. These observations conclude the proof. □

As already proved in [2], when using P systems with only one membrane, at most five objects are needed to obtain computational completeness:

Corollary 3.1 $g(1)\,NO_5P_1 = a(1)\,NO_5P_1 = NRE$.

Moreover, from Theorem 3.1 we can also conclude that P systems with two membranes are computationally complete with only four objects:

Corollary 3.2 $g(1)\,NO_4P_2 = a(1)\,NO_4P_2 = NRE$.

3.3 At Least Two Symbols and at Least Two Membranes

On the other hand, for $s, m \geq 2$, we can show that P systems with s symbols and m membranes can simulate $(s-1)(m-1)$-register machines:

Theorem 3.2 *Any mn-register machine can be simulated by a P system with $n+1$ symbols and $m+1$ membranes, with $n, m \geq 1$.*

Proof. Consider a register machine $M = (d, R, l_0, l_{halt})$ with $d = mn$ registers. We construct the P system

$$\begin{array}{lcl} \Pi & = & (O, \mu, w_1, \cdots, w_{m+1}, R_1, \cdots, R_{m+1}), \\ O & = & \{p\} \cup \{a_j \mid 1 \leq j \leq n\}, \\ \mu & = & [_1\,[_2\,]_2 \cdots [_{m+1}\,]_{m+1}\,]_1, \\ w_1 & = & w_0, \\ w_{i+1} & = & \prod_{j=1}^{n} a_j^{r_{j+(i-1)n}},\ 1 \leq i \leq m, \end{array}$$

that simulates the actions of M as follows. The contents of register $j+(i-1)n$ is represented by the multiplicity of symbols a_j in region $i+1$, whereas the symbol p is needed for encoding the instructions of M; this time, too many copies of a_1 in the skin membrane have the function of trap symbols.

Again, an important part of the proof is to define a suitable encoding $c : \mathbb{N} \to \mathbb{N}$ for the instructions of the register machine, and at most 6 different subsequent labels will be used for each instruction, hence, without loss of generality we assume the labels of M to be positive integers such that the labels assigned to ADD and SUB instructions have the values $6i-5$ for $1 \leq i < t$, as well as $l_0 = 1$ and $l_{halt} = 6(t-1)+1$, for some $t \geq 1$.

Since one copy of a_s will be used for addition/subtraction, now the operation multisets will be encoded by even numbers of object a_1, i.e., we take $h = 2\,(12d + 1) = 24d + 2$ and define the following operation multisets:

$$\begin{array}{ll} ADD: & \alpha_+(r) = {a_1}^{6d+2r} p^{h-(6d+2r)},\ 1 \leq r \leq d, \\ SUBTRACT: & \alpha_-(r) = {a_1}^{8d+2r} p^{h-(8d+2r)},\ 1 \leq r \leq d, \\ ZEROTEST: & \alpha_0(r) = {a_1}^{10d+2r} p^{h-(10d+2r)},\ 1 \leq r \leq d. \end{array}$$

In a similar way as before, we now take

$$g = 2h = 48d + 4$$

and define the function c by

$$c\,(x) = g(x + 6t - 4) \text{ for } x \geq 0.$$

For $l_1 = 1$ we therefore obtain

$$c\,(l_1) = g\,(6t - 3) = (48d + 4)\,(6t - 3)$$

as well as

$$w_0 = p^{c(l_1)} = p^{(48d+4)(6t-3)}.$$

Finally, for f we again take

$$f = c\,(l_{halt} + 1) = 2g(6t - 4)$$

which is so large that after getting f symbols we inevitably enter an infinite loop with the rule

$${a_1}^f / {a_1}^{3f}.$$

Equipped with this coding function and the constants defined above we are now able to define the following set of symport / antiport rules assigned to the membranes for simulating the actions of the given register machine M :

$$
\begin{aligned}
R_1 \;=\;& \{p^{c(l_1)}/p^{c(l_{1+1})}\alpha_+(s+(s'-1)n)a_s, p^{c(l_1+1)}/p^{c(l_1+2)},\\
& p^{c(l_1+2)}/p^{c(l_1+3)}, p^{c(l_1+3)}\alpha_+(s+(s'-1)n)/p^{c(l_2)},\\
& p^{c(l_1+3)}\alpha_+(s+(s'-1)n)/p^{c(l_3)} \mid\\
& l_1 : (A(s+(s'-1)n), l_2, l_3), 1 \le s \le n, 1 \le s' \le m\}\\
\cup\;& \{p^{c(l_1)}/p^{c(l_1+1)}\alpha_-(s+(s'-1)n), p^{c(l_1+1)}/p^{c(l_1+2)},\\
& p^{c(l_1+2)}/p^{c(l_1+3)}, p^{c(l_1+3)}\alpha_-(s+(s'-1)n)a_s/p^{c(l_2)},\\
& \alpha_-(s+(s'-1)n)/{a_1}^{3f},\\
& p^{c(l_1)}/p^{c(l_1+4)}\alpha_0(s+(s'-1)n), p^{c(l_1+4)}/p^{c(l_1+5)},\\
& p^{c(l_1+5)}\alpha_0(s+(s'-1)n)/p^{c(l_3)} \mid\\
& l_1 : (S(s+(s'-1)n), l_2, l_3), 1 \le s \le n, 1 \le s' \le m\}\\
\cup\;& \left\{p^{c(l_{halt})}/\lambda, p^h/{a_1}^{3f}, {a_1}^f/{a_1}^{3f}\right\}
\end{aligned}
$$

and for $1 \le r \le m$,

$$
\begin{aligned}
R_{r+1} \;=\;& \{\lambda/\alpha_+(s+(s'-1)n)a_r, \alpha_+(s+(s'-1)n)/\lambda \mid\\
& l_1 : (A(s+(s'-1)n), l_2, l_3), 1 \le s \le n, 1 \le s' \le m\}\\
\cup\;& \{a_s/\alpha_-(s+(s'-1)n), \alpha_-(s+(s'-1)n)/\lambda,\\
& a_s/\alpha_0(s+(s'-1)n) \mid\\
& l_1 : (S(s+(s'-1)n), l_2, l_3), 1 \le s \le n, 1 \le s' \le m\}.
\end{aligned}
$$

The operations ADD, SUBTRACT, and ZERO TEST now are carried out for all registers r as in the preceding proof for the registers $r > n$. Hence, we do not repeat all the arguments of the preceding proof, but stress the following important differences:

We now take advantage that the operation multisets additionally satisfy the property that now the number of symbols p can never be split into two parts that could be interpreted as belonging to two operation multisets; this guarantees that during a correct simulation, inside an elementary membrane at most one operation can be executed - and if it is the wrong one (i.e., we do not use all symbols p, but instead use more symbols a_1 from the amount representing the contents of a register), then we return a number of symbols p which is too small to allow the correct rule to be applied from R_1, instead the "trap rule" $p^h/{a_1}^{3f}$ will be applied. □

From the result proved above we can immediately conclude the following, thus also improving the result from [16] where $g(1)\,NO_3P_4 = NRE$ was proved: we can reduce the number of membranes from four to three when using only three objects or the number of symbols from three to two when using four membranes.

Corollary 3.3 $NRE = g(1)NO_3P_3 = a(1)NO_3P_3 = g(1)NO_2P_4 = a(1)NO_2P_4$.

3.4 One Symbol

If only one symbol is available, then so far we do not know whether computational completeness can be obtained even when not bounding the number of membranes (in contrast to tissue P systems which have been shown to be computationally complete with at most seven cells, see [1]). Yet at least we can generate any regular set of natural numbers in only two membranes (remember that with only one membrane we have got a characterization of $NFIN$, see Example 3.1).

Example 3.5 $g(1)NO_1P_2 \supseteq NREG$.

Any finite set can be generated without using the second membrane (see Example 3.1), so we proceed with infinite sets. Let $M \in NREG - NFIN$, then there exist finite sets M_0, M_1 with $M_1 \neq \emptyset$ and a number $k > 0$ such that $M = M_0 \cup \{i + jk \mid i \in M_1,\ j \in \mathbb{N}\}$.

Let m be the smallest element of M such that $m > \max(M_0 \cup M_1 \cup \{2k\})$; moreover, let $m' = m + 2k$ (thus, $m' \in M$). Then we consider the P system constructed as follows:

$\Pi = (\{a\}, [_1\ [_2\ ]_2\ ]_1, a^{m'}, \lambda, R_1, R_2)$ *where*

$R_1 = \{a^{m'}/a^i \mid i \in M_0\} \cup \{a^{m'}/a^m, a^m/a^{m+k}\} \cup \{a^m/a^i \mid i \in M_1\}$,

$R_2 = \lambda/a$,

We assume the result of a halting computation to be collected in the second membrane, and we claim $g(1)N(\Pi) = M$:

$g(1)N(\Pi) \supseteq M$:

The elements of M_0 are generated in one step, while the rest of M can be generated by

$[_1\ a^{m'}[_2\ ]_2\ ]_1 \Rightarrow [_1\ a^m[_2\ ]_2\ ]_1 => [_1\ a^{m+k}[_2\ ]_2\ ]_1 \Rightarrow^{j-1}$

$[_1\ a^{m+k}[_2\ a^{(j-1)k}\ ]_2\ ]_1 \Rightarrow [_1\ a^i[_2\ a^{jk}\ ]_2\ ]_1 \Rightarrow [_1\ [_2\ a^{i+jk}\ ]_2\ ]_1$

or by

$[_1\ a^{m'}[_2\ ]_2\ ]_1 \Rightarrow [_1\ a^m[_2\ ]_2\ ]_1 \Rightarrow [_1\ a^i[_2\ ]_2\ ]_1 \Rightarrow [_1\ [_2\ a^i\ ]_2\ ]_1$.

$g(1)N(\Pi) \subseteq M$:

What other derivations can we get different from those described above?

– *If all m' symbols enter membrane 2, $m' \in M$.*

– *If all m symbols enter membrane 2 (possibly after some additions of k), $m + jk \in M$ (by the definition of m, $m \in M$ and, moreover, it can be*

prolongued by multiples of k).

– *If during the first step m copies of the symbol a are used instead of m' (and 2k fall inside), then the system generates some number $2k + (i + jk)$ or $2k + (m + jk)$; all these numbers belong to M, too.*

Nothing else can happen, because $m + k < m'$ and $\max(M_0 \cup M_1) < m$ and because all symbols not used by R_1 fall into region 2.

4 Summary and Open Questions

From the main theorems (Theorem 3.1 and Theorem 3.2) established in the preceding section showing that P systems with symport/ antiport rules and $m \geq 1$ membranes as well as $s \geq 2$ symbols can simulate a register machine with $\max\{m(s-2), (m-1)(s-1)\}$ registers in combination with Propositions 2.2 and 2.3 we infer the following general results:

Theorem 4.1 *$g(1)NO_sP_m = a(1)NO_sP_m = NRE$, for $m \geq 1$, $s \geq 2$, $m + s \geq 6$.*

We conjecture that these results establishing the computational completeness bounds are optimal.

As the halting problem for d-register machines is undecidable for $d \geq 2$, from Theorems 3.1 and 3.2 we also obtain the following result:

Theorem 4.2 *The halting problem for P systems with symport/ antiport rules and $s \geq 2$ symbols as well as $m \geq 1$ membranes such that $m + s \geq 5$ is undecidable.*

As 1-register machines can generate/ accept all regular number sets, we obtain the following:

Theorem 4.3 *$g(1)NO_3P_1 \cap g(1)NO_2P_2 \cap a(1)NO_3P_1 \cap a(1)NO_2P_2 \supseteq NREG$.*

The main results established in this paper now can be summarized in the following table:

	Membranes					
$\|O\|$	1	2	3	4	5	m
1	A	B	B	B	B	B
2	C	1	2 (U)	$\boxed{3}$	4	$m-1$
3	1	**2** (U)	$\boxed{4}$	6	8	$2m-2$
4	2 (U)	$\boxed{4}$	**6**	9	12	$3m-3$
5	$\boxed{3}$	6	9	**12**	16	$4m-4$
6	4	8	12	16	**20**	$5m-5$
s	$s-2$	$2s-4$	$3s-6$	$4s-8$	$5s-10$	$\max\{m\,(s-2)\,,$ $(m-1)\,(s-1)\}$

In the table depicted above, the class of P systems indicated by

A generates exactly $NFIN$;

B generates at least $NREG$;

C generates at least $NREG$ and accepts at least $NFIN$;

d can simulate any d-register machine.

A box around a number indicates a known computational completeness bound, (U) indicates a known unpredictability bound, and a number in boldface shows the diagonal where Theorem 3.2 and Theorem 3.1 provide the same result (because in that case $m\,(s-2)$ equals $(m-1)\,(s-1)$); the numbers above this diagonal are taken from Theorem 3.2, while the numbers below the diagonal are taken from Theorem 3.1.

Based on these simulation results, we now could discuss in more detail how many symbols s and membranes m at most are needed to accept or generate recursively enumerable sets of vectors of natural numbers or compute functions $\mathbb{N}^k \to \mathbb{N}^l$ (e.g., recursively enumerable sets of d-dimensional vectors, $d \geq 1$, can be generated / accepted by P systems with symport / antiport rules using at most $d+4$ symbols in one membrane, see also [2]). Yet as all these results are direct consequences of the corresponding computational power of the simulated register machines (see Propositions 2.3, 2.2, 2.1), we do not follow this line any further.

Some interesting questions still remain open, let us list a few of them:

- Are P systems with one symbol universal? (The corresponding result holds for tissue P systems.)

- Can P systems with one symbol accept $NREG$? Can they accept at least $NFIN$?
- What is the class of number sets accepted by P systems with only one membrane and only one symbol? (Conjecture: contained in $NREG$, incomparable with $NFIN$.)

Acknowledgements. Artiom Alhazov is supported by the project TIC2002-04220-C03-02 of the Research Group on Mathematical Linguistics, Tarragona; he also acknowledges the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research and Development Foundation (CRDF), Award No. MM2-3034. This paper was written during his stay at the Vienna University of Technology.

The work of M. Oswald is supported by FWF-project T225-N04.

References

[1] A. Alhazov, R. Freund, M. Oswald: Tissue P Systems with Antiport Rules and a Small Number of Symbols and Cells. In: C. De Felice, A. Restivo (Eds.): Developments in Language Theory, 9th International Conference, DLT 2005, Palermo, Italy, July 4 – 8, 2005. *Lecture Notes in Computer Science* **3572** (2005), 100–111.

[2] A. Alhazov, R. Freund: P systems with one membrane and symport / antiport rules of five symbols are computationally complete. M.A. Gutierrez-Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan (Eds.): *Proceedings of the Third Brainstorming Week on Membrane Computing*, Sevilla (Spain), January 31 – February 4 (2005), 19–28.

[3] J. Dassow, Gh. Păun: Regulated Rewriting in Formal Language Theory. Springer-Verlag, Berlin (1989).

[4] R. Freund, M. Oswald: P Systems with activated/prohibited membrane channels. In: [17], 261–268.

[5] R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae* **49**, 1–3 (2002), 81–102.

[6] R. Freund, Gh. Păun: On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. In: M. Margenstern,

Yu. Rogozhin (Eds.): Machines, Computations, and Universality, Third International Conference, MCU 2001, Chisinau, Moldavia, May 23-27, 2001. *Lecture Notes in Computer Science* **2055**, Springer-Verlag, Berlin (2001), 214–225.

[7] R. Freund, Gh. Păun: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *Theoretical Computer Science* **312** (2004), 143–188.

[8] R. Freund, A. Păun: Membrane systems with symport / antiport rules: universality results. In: [17], 270–287.

[9] R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel states. In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini (Eds.): *Proceedings of the Brainstorming Week on Membrane Computing*, Sevilla, February 2004, **TR 01/04** of Research Group on Natural Computing, Sevilla University (2004) 206–223 and *Theoretical Computer Science* **330** (2005), 101–116.

[10] R. Freund, M. Oswald: Tissue P systems with symport / antiport rules of one symbol are computationally complete. In: M.A. Gutiérrez-Naranjo, Gh. Păun, M.J. Pérez-Jiménez (Eds.): *Cellular Computing. Complexity Aspects*. Fénix Editora, Sevilla (2005), 185–197.

[11] C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón: Tissue P systems. *Theoretical Computer Science* **296** (2) (2003), 295–326.

[12] M.L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA (1967).

[13] A. Păun, Gh. Păun: The power of communication: P systems with symport / antiport. *New Generation Computing* **20**, 3 (2002), 295–306.

[14] Gh. Păun, Computing with membranes, *Journal of Computer and System Sciences* **61**, 1 (2000), 108–143 and
TUCS Research Report **208** (1998) (`http://www.tucs.fi`).

[15] Gh. Păun: *Computing with Membranes: An Introduction*. Springer-Verlag, Berlin (2002).

[16] Gh. Păun, J. Pazos, M.J. Pérez-Jiménez, A. Rodríguez-Patón: Symport / antiport P systems with three objects are universal. *Fundamenta Informaticae* **64**, 1–4 (2005), 353-367.

[17] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing. International Workshop WMC 2002, Curtea de Argeş, Romania, Revised Papers. *Lecture Notes in Computer Science* **2597**, Springer-Verlag, Berlin (2003).

[18] G. Rozenberg, A. Salomaa (Eds.): Handbook of Formal Languages (3 volumes). Springer-Verlag, Berlin (1997).

[19] The P Systems Web Page, `http://psystems.disco.unimib.it`

Software Tools / P Systems Simulators Interoperability

Fernando ARROYO[1], **Juan CASTELLANOS**[2], **Luis FERNÁNDEZ**[1], **Victor J. MARTÍNEZ**[3], **Luis F. MINGO**[4]

[1]Dpto. de Lenguajes, Proyectos y Sistemas Informáticos
Escuela Universitaria de Informática
Universidad Politécnica de Madrid
http://www.lpsi.eui.upm.es
E-mail: {setillo,farroyo}@eui.upm.es

[2]Dpto. de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
http://www.dia.fi.upm.es
E-mail: jcastellanos@fi.upm.es

[3]Dpto. de Arquitectura y Tecnología de Computadores
Escuela Universitaria de Informática
Universidad Politécnica de Madrid
http://www.atc.eui.upm.es
E-mail: victormh@eui.upm.es

[4]Dpto. de Organización y Estructura de la Información
Escuela Universitaria de Informática
Universidad Politécnica de Madrid
Crta. de Valencia Km. 7
http://www.oei.eui.upm.es

Abstract

This paper presents a specific XML vocabulary for Transition P systems, tpsML, that facilitates the interoperability among the different existing simulators. Moreover, two interactive tools for edition and visualization of tpsMl documents, provide to P systems simulators users of the necessary transparency from the persistence format

of the P systems to simulate. It has been developed two translation tools from previous P systems simulators to tpsML and vice versa. They show the integration possibilities that they offer to the desired interoperability. The development of these tools have produced SAX-tpsML and DOM-tpsML libraries and XSL-tpsML, which increase the productivity bench mark in the development of new simulators. This improvement is produced because they (SAX-tpsML, DOM-tpsML and XSL-tpsML) assume the lexical, syntactic and semantic analysis and the error control, and then, it permits to concentrate only in the development of the desired functionality.

1 Introduction

Membrane Computing was introduced by Gh. Păun in 1998 [15]. This paradigm is based on the cell functioning and the way cells compute. The most relevant characteristics of this new computational model are its non-deterministic and massive parallelism. There are many different variant of membrane systems, starting from the simplest models (Transition P systems) to the most complex (tissue P systems); with many different characteristic related to evolution rules classes: catalyst, cooperative, non-cooperative; with or without priorities; with actions δ and τ; or even different membrane classes: active, bilayer, etc.

Every new computational model requires an appropriate set of specific tools for the model. This set of tools must constitute a complete software development suite according to its characteristics. Nowadays, without hardware specifically designed to P systems, many software simulators had been developed. These simulators can execute some different variants of P systems and they had shown their capability for proving some proposed solutions to some problems. However, and with the growing interest in producing simulators, it will be also necessary: editors to facilitate the edition of P systems; analyzers, debuggers, tools for visualizations, and so on.

Each one of these tools need a grammar which determines the input and/or output files format supporting the P system in a persistent manner in the computer. If everyone of these tools establishes its own grammar for these files, make their own lexical, syntactic and semantic analysis; there will be a lot of redundant work and the production bench mark will decrease. On the other hand, users of different tools will need the knowledge about these tools. These two aspects, from the user and developer point of view, redound against these tools interoperability because the produced output by one tool cannot be used as input by another one, due to the different

grammars in their working files.

In order to avoid this problem, a based XML solution is proposed. The solution establishes a new standard vocabulary for Transition P systems (tpsML) and specific libraries that permit to check the vocabulary. The main goal is to facilitate the development of generic tools in such a way that: grammars of different tools are transparent to users and to provide to developer software reusability because they only need to define the new functionalities.

The structure of this paper is the following: First of all, a brief art state is presented about the different software tools available today. After that, it is shown the appropriate XML technologies for improving the current situation of software tools design in Membrane Computing. And then, the new tpsML vocabulary model is established. It is also presented, starting from tpsML vocabulary, the design of generic libraries (SAX-tpsML, DOM-tpsML, XSL-tpsML) and the generic tools (editor, validator, categorizator, visualizator, translators, etc.). Finally, some conclusions are presented about the developed work.

2 Related Work

Today, there are several software tools and some hardware proposal for implementing membrane systems in digital devices. Each one of the different tools implements different variant of P systems: Transition P systems [2], [19] and [5]; membrane systems with catalytic cells [18] and [7]; P systems with Active membrane [9]; or even some variant of P systems in the same simulator [11]. Moreover, each one of the implementations takes into account: the graphical interface for presenting the membrane system through the configurations the system reaches [7] and [14]; the maximal parallelization intra-inter regions [19]; a Web distributed system for the membrane community [6], etc. On the other hand, each one of the different tools has several limitations: memory, maximal number of membranes, rules, etc.; or even they do not limit the number of evolution step what can drive the system to an infinite process.

What it seams to be clear is that the objectives of researchers in membrane computing must drive the software tools development for this research area. Moreover, these objectives determine the programming language and the technologies associated for implementing the software system. This situation is illustrated today by the number of different programming languages and paradigms they have been used in membrane computing: declarative

languages like Prolog [12] [9], Scheme [5], LISP [18] and Haskell [2]; imperative languages like C/C++ [6] and Java [13] [8]; and even also CLIPS production rules systems [16].

A common aspect of each one of the analyzed tools is the presence of configuration files including the P system to simulate. Sometimes, it is specified through a formal grammar [2], which determines the correction of these configuration files. Sometimes, through syntactic structures accordingly to the programming language used for implementing the corresponding tool like in Prolog [9] or CLIPS [16]. Finally, in other cases the simulation tools represent the P systems with owner format files. These files are dependent of the implementation technology like in [7] using serialized objects with Java.

In the case of hardware processors, a similar situation is found for the different variants of P systems and different formats of the configuration files for the P system. In this case it is possible to find from connectivity arrays [3] to VHDL automatically generated [17].

It seems to be clear that this situation is going to be a problem. Each new hardware/software simulation tool on new variants or with new goals could establish new formats for the configuration files supporting the membrane system to simulate. This situation has two different inconveniences for the two different rôles of the tool: the P systems programmer (user) and the tools developer:

i. One hypothetic case of use for the programmer could be the following: he/she uses one simulator for a particular model with the classical limitations and particular objectives. In this case, he/she desires to analyze the possible results of his/her P system. In order to facilitate the debug it could also be necessary to use a graphical representational tool or a logs system; but this simulation tool does not support some of them. However, there are other simulation tool, with other objectives, that offers the needed services. Hence, it is necessary to translate the configuration files to this new software and then to debug it. Moreover, once the system has been debugged and in order to improve the system performance, it is needed to implement it in a distributed system. The current situation, in this case, obliges to interact with different tools with their different editors and/or to learn different languages grammar (Bio-language, Prolog, etc.)

ii. For a tools developer it may happen that some of the objectives for the first version of a simulator determine the election of a given

language/technology, but in the second version some new objectives are incorporated and then it could be necessary to change the language/technology for the development. In this case, it could be necessary to re-encode the semantic, syntactic and lexical analysis of the same grammar or to redefined the one of the file when it is dependent of implementation language data structures.

In both rôles, programmer or developer, there is a work overload -which is not necessary- due to the owner character of the configuration files grammar supporting the P system with which software tools interact.

3 XML Technologies

This section shows the convenience of XML Technologies in order to avoid the enumerated problem in the previous section. At the beginning the eXtensible Markup Language was designed oriented to electronic publication in wide scale. Nowadays, it is playing a very important rôle in the field of data interchange among computer systems due, mainly, to its simpleness and independence. This important rôle is coincident with the necessity, previously stated, of data interchange among different software simulation tools implemented with different technologies in the field of membrane systems.

Any XML document containing information is hierarchically structured. Every relevant information is enclosed in between two nested labels or tags, which include information about information -meta information. Hence, it is a very good situation in order to define information about P systems because such systems are, in essential, a hierarchy of membranes including information about evolution rules and multisets of objects.

The extensible character of this technology is completely open, but also it permits to restrict the set of tags and its hierarchy accordingly to the necessities of a specific community of developers and users. Validation rules for a particular community are defined through Document Type Definition (**DTD**) or **XML Schema**. Moreover, for each technology assuming validation of XML documents, there are validators for checking the correctness of the document accordingly to a DTD. Next section proposes a DTD model for Transition P systems community.

On the other hand, there are libraries for almost every programming language to manage any aspect of XML documents through processes in execution in the system: Prolog [22], Esqueme [23], Haskell [24], Java and C++ [25], Lips [26], etc.

There also are, two standard programming interfaces independent of the programming language: **SAX** -simple access xml- that improves efficiency in sequential retrieving information from XML documents through a callbacks based patterns, and **DOM** -document object model- for a random access, which load information in memory through a composite pattern of nodes associated to each tag. These two libraries are in charge of task related to: syntactical and lexical analysis and control error of XML documents. In this framework, the programmer only has to develop the synthesis phase: the visualizator, the simulator, etc. Hence, the development of one transition P system tool based on XML document only has to retrieve the information about the P system, accordingly to the DTD which is presented in the next section, and then to implement the simulation task. This avoid to develop the lexical, syntactical and semantical analyzer and the error free check in the input file.

As can be seen, incorporating XML technologies in the P systems software simulators, focusing in the proposed tpsML vocabulary, avoid the problems presented in the previous section because:

i. Users do not need to know about different grammar. The new standard is common to all of them. This increases the transparency bench mark in using tools.

ii. Developers avoid to define a grammar for theirs input files to the software tools they are developing. The grammar is given through the corresponding DTD. Moreover, they do not need to implement the corresponding analyzers. These is done by the SAX DOM libraries. These facts increase the reusability and productivity bench marks.

Moreover, it can also be assumed every P system simulator previously implemented. There is the XSL -styles language for XML- that permits to translate it in a very easy way the supported information by a XML document into several and different formats (text, HTML, pdf, etc.). In particular, a XSL can translate a tspML into a text accordingly to a determined grammar that specify the input file for a given simulation tool of P systems (bio-language, prolog, VHDL, etc.) without grammar modifications.

This is the way to achieve the interoperability among software and hardware simulation tools. Every new software / hardware tool simulating P systems accordingly to the standard shall produce and shall consume tpsML documents. Moreover, those previously developed can be incorporated to the general interoperability with the appropriate translator.

4 pstML Vocabulary

As it was shown in the previous section, the structure of a new vocabulary is given by a Document Type Definition (DTD). The DTD will establish the logical structure of the document that follows the vocabulary. The DTD fixes tags and their attributes and the needed nests among tags in order to facilitate the way of storing the appropriate information of P systems. In particular, the DTD has tags collecting information about objects, rules with their antecedent and consequent -with objects and their targets IN, OUT, HERE; rules priorities, objects δ and τ as it is shown in Table 1

```
<!ELEMENT object EMPTY>
<!ATTLIST object value CDATA #REQUIRED
                 multiplicity CDATA #REQUIRED>
<!ELEMENT targetedObject (IN | OUT)?>
<!ATTLIST targetedObject value CDATA #REQUIRED
            multiplicity CDATA #REQUIRED>
<!ELEMENT in EMPTY>
<!ATTLIST in target IDREF #IMPLIED>
<!ELEMENT out EMPTY>
<!ELEMENT rule (object+,targetedobject*)>
<!ATTLIST rule  id ID #REQUIRED
                action (d| T) #IMPLIED >
<!ELEMENT priority EMPTY>
<!ATTLIST priority  greater CDATA #REQUIRED
            minor CDATA #REQUIRED>
```

Table 1: Elements description in the DTD

In general, the vocabulary structures the information of static (Table 2) and dynamic (Table 3) aspects of the P system. The static aspects take into account the alphabet and the probably subset of catalyst in the P system and, also, the hierarchical membrane structure with the rules they have.

The dynamical aspects will be determined by the consecutive configurations of the different multisets included inside membranes, with their corresponding thickness, and the applied rules in order to achieve the transition between one configuration to the next one. This is included in order to follow the evolution of the system through some visualization tools.

Finally, it is possible to establish some classification of P systems attending different criteria: kind of rules (cooperative, non cooperative, catalyst),

```
<!ELEMENT alphabet (symbol+)>
<!ELEMENT symbol EMPTY>
<!ATTLIST symbol name CDATA #REQUIRED
    catalityc (yes | no) "no">
<!ELEMENT membrane (rule*,priority*,membrane*)>
<!ATTLIST membrane  id ID #REQUIRED>
```

Table 2: Static aspect of P system in the DTD

```
<!ELEMENT region (object*, region*)>
<!ATTLIST region  id CDATA #REQUIRED
          thickness (unthin | thin) #REQUIRED>
<!ELEMENT configuration (applied_rule*, object*, region,
     configuration*)>
<!ELEMENT applied_rule EMPTY>
<!ATTLIST applied_rule idrule IDREF #REQUIRED
                       idmembrane IDREF #REQUIRED>
```

Table 3: Dynamic aspect of P system in the DTD

target of the rules consequents (weak - strong targeting) and permeability control and/or dissolution (δ and/or τ). Optionally, is specified the output membrane or in absence the environment is considered as the system output (Table 4).

```
<!ELEMENT psystem (alphabet?, membrane, configuration?)
<!ATTLIST psystem rule-type (coo | ncoo | cat )   #IMPLIED
target-type (strong | weak)   #IMPLIED
type (delta | tau | delta-tau)   #IMPLIED
                  exit IDREF           #IMPLIED>
```

Table 4: Classification of P systems in the DTD

This vocabulary has been designed with the proposal of be able to store the different necessities of any Transition P systems simulation tools. Due to the recursive character of the configuration tag, it is possible to establish a configuration tree in which is included every possible P system execution. This characteristic will make possible to analyze the P system evolution. But also, the same framework, storing only one branch of the configuration

tree one specific P system execution is represented. Taking into account only the root of the tree, the initial configuration of the P system is depicted. Moreover, in the total absence of the configuration tree, only the static structure of the P system is represented; with the membrane hierarchy and its rules, open to any initial multisets configuration like in [10] occurs in the publication of general problem solutions.

5 Generic Libraries and Tools for tpsML

A kit of generic tools is presented here in order to facilitate the development of new Transition P systems simulators. This kit is useful for completing other specific tools necessities. All of them are developed in Java for Web because its multi-platform character gives access to every user in the P system community. The relation of these generic tools is:

- Document tpsML Editor (Figure 1) that is able to generate a valid tpsML interacting with specific controls buttons. The editor incorporates the dynamic control of errors accordingly with the DTD of the vocabulary.

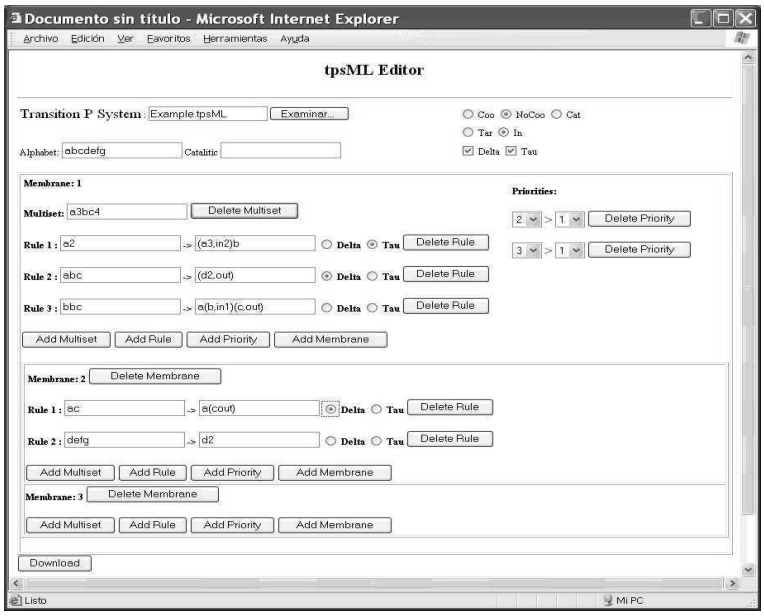


Figure 1: tpsML Editor

- Static Visualizator (Figure 2) for presenting the static component of a valid tpsML document, and also the multisets values in a given configuration. This tool does not can edit the tpsML document. It incorporates also errors detection accordingly with the DTD of the vocabulary.

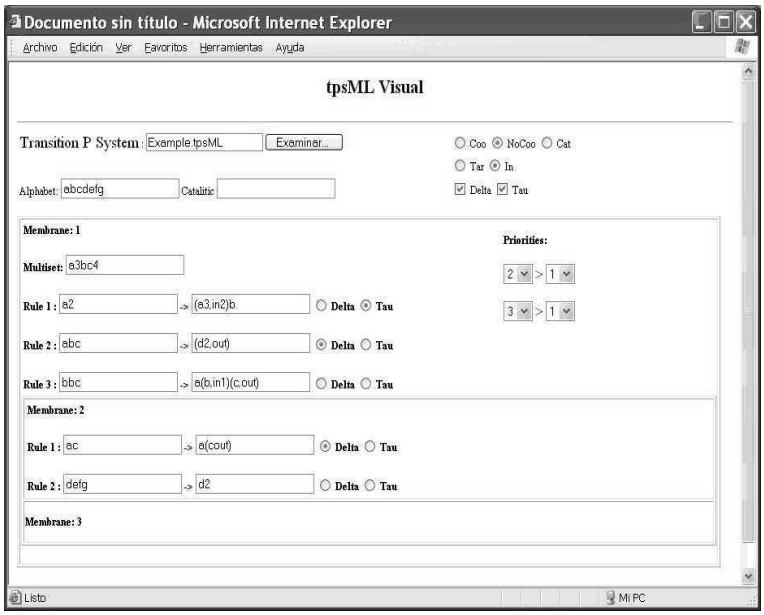


Figure 2: tpsML Visualizator

- Translators from tpsML to Bio-language and Connectivity arrays. These translators make feasible, first of all, to integrate into the new technology a previously developed tool [1]; and secondly, the first step to load information of the P system into a hypothetic membrane processor [3]. These two features show the interoperability among different tools.

The experience got in the development of these generic tools have produced the design of a class framework in Java. This framework is based in the combination of several software patterns as composite, visitor, bridge, etc. Moreover, the framework is completely reusable in the development process of new specific tools. The SAX-tpsML and DOM-tpsML libraries host this framework with specific class about the P systems domain. These classes are different from the SAX and DOM classes because they have specific behavior accordingly to the P system domain. In a similar way, it

has been also developed XSL-tpsML, incorporating specific templates corresponding to the proposed DTD. XSL-tpsML facilitates the development of new translators only with a few knowledge about XSL technology.

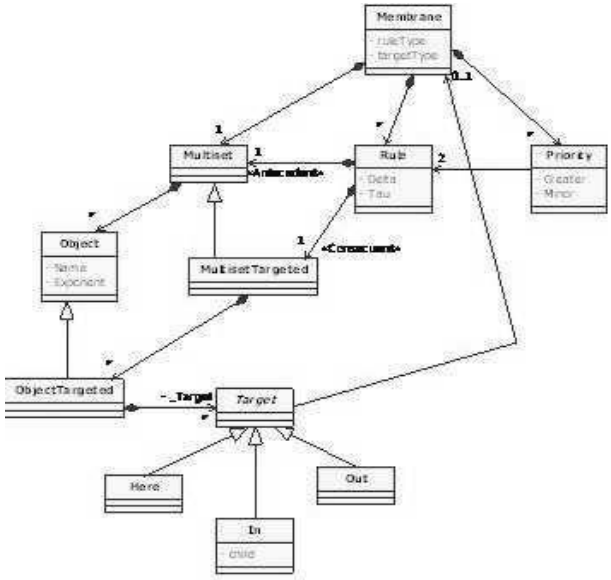


Figure 3: Class diagram of DOM-tpsML

6 Future Work

The presented framework of the previous section is based on three main axes: DTD defining the tpsML vocabulary, SAX-tpsML and DOM-tpsML libraries and the generic tools for editing, presenting and translating P systems. The main future working areas are addressed in these three directions.

On one hand, it is possible to study a new vocabulary taking into account different membrane systems models (active membranes, tissue P systems, etc.) or even, to abroad a new general vocabulary including other research field related to membrane computing areas like NEP's. Consequently, they could be developed SAX and DOM libraries and XSL for these new vocabularies. Finally, it is necessary consider to enrich the tools suite with dynamic visualizators, debugger, etc. and to complete the interoperability with new translator into simulator that they have been developed jet.

7 Conclusions

P systems have demonstrated their usefulness as a new computational model. This fact makes new development tools to be necessary for researchers and developers. Those tools should be focused to this new paradigm. It is considered necessary that development community joint efforts to increase productivity and to make as transparent as possible this environments. These objectives land in interoperability of simulators. Here, it is presented a first step to this interoperability through a standard for the persistence of P systems based on XML technologies. Tools and libraries in this framework show the accomplishment of these objectives: transparency for users and reusability for developers.

References

[1] F. Arroyo, A.V. Baranda, J. Castellanos, C. Luengo, L. F. Mingo: Structures and Bio-Language to Simulate Transition P Systems on Digital Computers. In: C.S. Calude, Gh. Paun, G. Rozenberg, A. Salomaa (Eds.): Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View. *Lecture Notes in Computer Science* **2235**, Springer-Verlag (2001), 1–16.

[2] F. Arroyo, C. Luengo, A. V. Baranda, L. F. de Mingo: A software simulation of transition P systems in Haskell. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): *Pre-Proceedings of Second Workshop on Membrane Computing*, Curtea de Argeş, Romania (2003) and *Lecture Notes in Computer Science* **2597**, Springer-Verlag (2003), 19–32.

[3] F. Arroyo, C. Luengo, J. Castellanos, L. F. de Mingo: A binary data structure for membrane processors: Connectivity arrays. In: A. Alhazov, C. Martín-Vide, Gh. Păun (Eds.): *Preproceedings of the Workshop on Membrane Computing.* Tarragona (2003), 41–52 and in: C. Marín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *Lecture Notes in Computer Science* **2933**, Springer-Verlag (2004), 19–30.

[4] F. Arroyo, C. Luengo, L. Fernandez, L. F. Mingo, J. Castellanos: Simulating membrane systems in digital computers. *International Journal Information Theories and Applications* **11** (1) (2004), 29–34.

[5] D. Balbontín-Noval, M. J. Pérez-Jiménez, F. Sancho-Caparrini. A MzScheme Implementation of Transition PSystems. In: Gh. Păun,

G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing 2002. *Lectured Notes in Computer Science* **2597**, Springer-Verlag, Berlin (2003), 57–73.

[6] C. Bonchis, C. Isbasa, D. Petcu, G. Ciobanu: WebPS: A web-based P system simulator with query facilities. *Proceedings of the Third Brainstorming Week on Membrane Computing.* Sevilla, Spain (2005), 63–72.

[7] G. Ciobanu, D. Paraschiv: Membrane Software. A P System Simulator. *Pre-Proceedings of Workshop on Membrane Computing,* Curtea de Argeş, Romania. *T*echnical Report 17/01 of Research Group on Mathematical Linguistics, Rovira i Virgili University, Tarragona, Spain (2001), 45–50 and *Fundamenta Informaticae* **49** (1-3) (2002), 61–66.

[8] G. Ciobanu, D. Petcu. P accelerators: Parallelization of sequential simulators. *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects)*, Sevilla, Spain (2005), 177–186.

[9] A. Cordón-Franco, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, F. Sancho-Caparrini: A Prolog Simulator for Deterministic P Systems with Active Membranes. In: M. Cavaliere, C. Martín-Vide, Gh. Păun (Eds.): *Brainstorming Week on Membrane Computing.* Rovira i Virgili Univ., Tech. Rep. No. 26*,* Tarragona (2003), 141–154 and *New Generation Computing* **22** (4) (2004), 349–363.

[10] A. Cordón-Franco, M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez. A. Riscos-Núñez, F.Sancho-Caparrini: Implementing in Prolog an effective cellular solution to the Knapsack problem. In C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, (Eds.): Membrane Computing. *Lecture Notes in Computer Science* **2933**, Springer-Verlag, Berlin (2004), 140–152.

[11] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A. Riscos-Núñez: A simulator for confluent P systems. *Proceedings of the Third Brainstorming Week on Membrane Computing.* Sevilla, Spain (2005), 169–184.

[12] M. Malita: Membrane Computing in Prolog. *Pre-proceedings of the workshop on multiset Progcessing.* Curtea de Argeş, Romania, CDMTCS TR 140 Univ. of Auckland (2000), 159–175.

[13] I. A. Nepomuceno-Chamorro: Simulaciones de P Systemas en Java, Aplicación SimCA. *CCIA Universidad de Sevilla Sección III* **N 3** (2003), 1–115.

[14] I. A. Nepomuceno-Chamorro: A Java Simulator for Basic Transition P Systems. In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, (Eds.): *Second Brainstorming Week on Membrane Computing.* Sevilla, Spain (2004), 309–315 and *Journal of Universal Computer Science* **10** (5) (2004), 620–619.

[15] Gh. Păun: Computing with membranes. Turku Center for Computer Science-TUCS Report 208 (1998) and *Journal of Computer and System Sciences* **61** (2000).

[16] M. J. Pérez-Jiménez, F. J. Romero-Campero: A CLIPS simulator for recognizer P systems with active membranes. *Proceedings 2nd Brainstorming Week on Membrane Computing*, University of Sevilla Tech Rep (2004), 387–413.

[17] B. Petreska, C. Teuscher: A hardware membrane system. In: A. Alhazov, C. Martín-Vide, Gh. Păun (Eds.): *Preproceedings of the Workshop on Membrane Computing.* Tarragona (2003), 343–355.

[18] Y. Suzuki, H. Tanaka: On a Lisp implementation of a class of P Systems. *Romanian Journal of Information Science and Technology* **3** (2) (2000), 173–186.

[19] A. Syropoulos, E. G. Mamatas, P. C. Allilomes, K. T. Sotiriades: A distributed simulation of P systems. In A. Alhazov, C. Martin-Vide, Gh.Păun (Eds.): *Preproceedings of the Workshop on Membrane Computing.* Tarragona (2003), 455–460.

[20] J. Joyce: XML. *Scientific Computing & Instrumentation* **v17 i7** (2000), 10.

[21] R. Kay: QuickStudy: XSL. *Computerworld May 3* **v38 i18** (2004), 28.

[22] J. Wielemaker: SWI-Prolog SGML/XML parser. SWI, University of Amsterdam, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands, (2002). URL: `http://www.swi-prolog.org/packages/sgml2pl.html`.

[23] O. Kiselyov: SXML Specification. *ACM SIGPLAN Notices* **v.37** (6) (2002), 52–58.

[24] M. Wallace, C. Runciman: Haskell and XML: Generic Combinators or Type-Based Translation? In: *Proceedings of the Fourth*

ACM SIGPLAN International Conference on Functional Programming (ICFP'99).

[25] E. J. Bruno: C++, Java, & XML processing: creating and parsing XML with Java and C++. *C/C++ Users Journal* **v22 i7 6**(6) (2004).

[26] K. Nørmark: Using Lisp as a Markup Language – The LAML Approach. *European Lisp User Group Meeting*, Amsterdam (1999).

[27] The P System Web Page: `http://psystems.disco.unimib.it/`

Gene Regulatory Network Modelling by Means of Membrane Systems

Nicolae BARBACARI[1], **Aurelia PROFIR**[2], **Cleopatra ZELINSCHI**[3]

[1]Institute of Genetics
Academy of Sciences, Republic of Moldova
E-mail: nbarbacar@hotmail.com

[2]Institute of Applied Physics
Academy of Sciences
Republic of Moldova
E-mail: aurelia@cc.acad.md

[3]Department of Mathematics and Computer Science
State University of Moldova
E-mail: cleopatra@usm.md

Abstract

One objective of Systems biology is to create predictive quantitative models of gene regulatory networks that govern numerous cellular functions. Our aim[1] is to model the functioning of genetic machinery to elucidate the *continuous* aspects of gene structure-functional organization. A novelty of the DNA modelling is that the DNA (in genes) is mapped by means of systems of elementary membranes, arranged into chains, for capture in detail gene structure and all functional aspects of temporal behavior of gene expression regulation. A P transducer model of the living cell that provides system integration of all interacting components at the main hierarchical levels of cell organization is introduced. Important properties of the gene function such as continuous-valued internal state of a gene, different levels of gene expression, differential gene expression, etc. are illustrated through simulation. We also describe some generalizations of the P transducer model of the living cell and discuss their relevance to the real-world gene networks.

[1]This work is supported by CRDF-MRDA project, and BGP-II, Award No. MM2-3034.

Introduction

Regulation of gene expression is a vital process in the cell and involves the functional organization of the chromosomal DNA. It also involves the action of specific protein factors, which can act at different steps in the gene expression pathway. Methods to control gene expression are important both for functional genomics and for potential therapeutic applications and will be of strategic importance in biological research.

Numerous models have already been proposed for the analysis of gene networks data. The most commonly used models rely on gene network representation where each gene has one of only two states: on and off. Examining some of the gene expression data sets, it is clear that genes spend a lot of their time at intermediate values: gene expression levels tend to be continuous rather than discrete, and discretization can lead to a large loss of information [1]. Unlike other models, our gene regulatory network model, based on membrane systems, opens a new view of gene regulatory network representation. Continuous aspects of structure-functional organization of the DNA (in genes) are elucidated by using continuous-time P systems of elementary membranes, arranged into chains. Consequently, the integration of gene expression models with other biological processes (cellular functions) models is facilitated. This view in detail captures gene structure and all functional features of temporal behavior of gene expression regulation. On the one hand, this approach can easily describe the discrete aspects of gene regulation such as all the relevant molecular interactions one-by-one with the DNA. On the other hand, the continuous-valued internal state of each gene, dictated by the gene regulatory network of the living cell, is illustrated.

It is known that Systems biology is focused on understanding cellular networks (gene regulatory networks, protein networks, and membrane networks) to build integrated formal and computational models, with suitable notations, necessary to describe all cellular networks as an entire system [2]. Within this overall framework, we introduce the P transducer model of the living cell, which can be seen as a coherent conceptual framework that provides system integration of all interacting components of the main hierarchical levels of organization: DNA, gene regulatory network, and cellular levels. Using the concept of the P transducers [3] we introduce a new quantitative computational model of the living cell as a basis of analysis and simulation of real-wold gene networks, which govern cellular functions, and the dynamic behavior of the living cell.

1 Gene regulatory network modelling

An important role in gene regulation is played by specific proteins, called transcription factors, which influence the transcription of particular genes by binding to their regulatory regions. In this way a product of one gene can influence the expression of another gene, and we can consider a network of gene regulation. A gene regulatory network is a collection of DNA segments (genes with their regulatory regions) which interact with each other and with other substances in the cell, thereby governing the rates at which genes are transcribed into $mRNA$. The molecular output of a gene regulatory network is the constellation of $RNAs$ and proteins encoded by network target genes, each with a specific role to play. The structural and functional characteristics of different types of cells are determined by the nature of the proteins present [4].

Our goal is to model the structure-functional organization of genes with their regulatory regions both for prokaryotic and eukaryotic cells for easy analysis of important properties of gene function such as continuous-valued internal state of a gene, all the relevant molecular interaction with DNA, all functional aspects of real-time behavior of gene expression regulation, etc.

Genes with their regulatory regions show diversity in size and organization [5]. There are, however, several conserved features. We consider a simplified view of a gene with its regulatory region as a structure consisting of *regulatory region* (RR), *gene coding region* and *gene transcription termination site* (T) (Fig. 1).

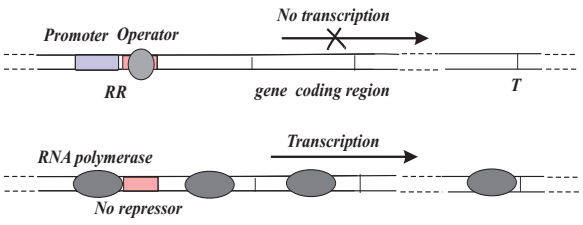


Figure 1: A schematic representation of a gene with the negative regulation of gene expression (for prokaryotic cell). RR – regulatory region consisting of a promoter and an operator; T – transcription termination site.

To elucidate continuous features of gene expression (both for prokaryoric and eukaryoric genes) we map the DNA (in a gene) by means of a system of elementary membranes, arranged into a chain (Fig. 2). The direct com-

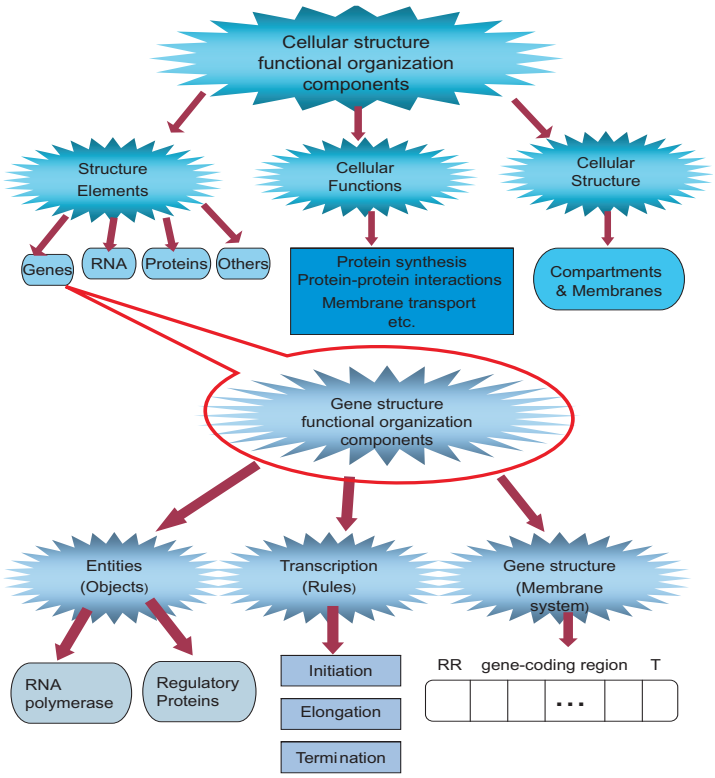


Figure 2: Schematic representation of cellular structure-functional organization on the basis of the P transducer concept. Genes with their regulatory regions are mapped by systems of elementary membranes, arranged into chains.

munication between elementary membranes along the chain is done in a *one-way* manner. The first elementary membrane of the chain represents the gene regulatory region consisting of a promoter and transcription factor binding sites. The last elementary membrane of the chain represents the gene transcriptional termination site. The rest of elementary membranes of the chain represent the coding region of the gene. The number of elementary membranes of the chain correlates with the maximal number of RNA polymerases simultaneously transcribing the activated gene.

High-speed and slow biochemical reactions (depending on the concentration of the reactants) evolve in a continuous way and in parallel in living cells. Such timing hierarchy is an essential, distinctive feature of biosys-

tems. On the other hand, discrete-time P systems have limited applicability to common biological data sets. We propose a new formalization of P systems, continuous-time P systems, that combines the discrete and continuous aspects of biosystem representation. We introduce the notion of *rate*, denoted by symbol $r_m \in R$, $m \in N$, associated to rule. A rule is applied at the instant $t_{i+1} \in R^+$ according to as follows. We consider time delay (denoted by Δt), which is inverse proportional to the rate r_m, between the instant $t_i \in R^+$ when the rule became to be available and the instant t_{i+1} when the rule is applied. $\Delta t = t_{i+1} - t_i$.

Definition 1 *The continuous-time P systems of elementary membranes, arranged into chains, representing continuous aspects of the structure-functional organization of the g_p genes linked as a network, are constructs of the following forms:*

$$\Pi_{g_p} = (O_{g_p}, \mu_{g_p}, w_{g_p(1)}, w_{g_p(2)}, \ldots, w_{g_p(l_p)}, E, R_{g_p(1)}, R_{g_p(2)}, \ldots, R_{g_p(l_p)}),$$

$$1 \leq p \leq n,$$

where:

- n is the number of the g_p genes linked as a network;
- $l_p \geq 1$ are degrees of the Π_{g_p} systems (number of elementary membranes), $1 \leq p \leq n$;
- O_{g_p} are alphabets of objects associated with the Π_{g_p} systems, $1 \leq p \leq n$;
- μ_{g_p} are membrane structures of the Π_{g_p} systems, $1 \leq p \leq n$. The Π_{g_p} systems are formed by l_p membranes, arranged into chains. The direct communication between membranes along chains (representing the g_p genes with their regulatory regions) is done in an *one-way* manner: $g_p(1) \rightarrow g_p(2) \rightarrow g_p(3) \rightarrow \ldots \rightarrow g_p(l_p)$. The first membranes labelled by $g_p(1)$ represent the g_p gene *regulatory region* and they are named the *input/output membranes*. The last membranes of the chains labelled by $g_p(l_p)$, named as the *output membranes*, represent the *transcriptional termination sites*. The elementary membranes labelled by $g_p(2), g_p(3), \ldots, g_p(l_{p-1})$ represent the g_p gene coding regions;

- $w_{g_p(1)}, w_{g_p(2)}, \ldots, w_{g_p(l_p)}$ are objects over $O^*_{g_p}$ $(1 \leq p \leq n)$ initially present in each elementary membrane of the membrane structure μ_{g_p};

- E is the set of objects present in arbitrarily many copies in the environment of the Π_{g_p};

- $R_{g_p(1)}$, $R_{g_p(2)}$, $\ldots$, $R_{g_p(l_p)}$ are finite sets of rules associated with each membrane of the membrane systems μ_{g_p}, $1 \leq p \leq n$. They illustrate the communication between the elementary membranes of the chains, and between the membranes of the Π_{g_p} systems and their environment. The rates $r_m \in R$, $m \in N$ of biological reactions, modelled by the rules of the Π_{g_p} systems, are indicated as superscripts. All available rules are applied according to their rates in a non-deterministic maximally parallel manner. Only one object is specified in all rules described below. Each rule has one of the following forms:

  –$(x, in)|^{r_m}_a$ or $(x, in)|^{r_m}_{\neg a}$, $x \in O_{g_p}$, means that from the environment the object x enters the input/output membrane only in the presence or in the absence of a, where a is a multiset of objects, $a \subset O^*_{g_p}$. These conditional rules reflect the functional organization of the gene regulatory region.

  –$(x, out)|^{r_m}$, $x \in O_{g_p}$, means that the object x is sent *out*, from the elementary membrane with which the rule is associated, into the environment.

  –$(x, go)|^{r_m}$, $x \in O_{g_p}$, means that the object x leaves the membrane, with which the rule is associated, and passes to the next membrane of the chain.

  –$(y, out; x, go)|^{r_m}$, means that the object $y \in O_{g_p}$ is sent out from the input/output membrane into the environment and simultaneously the object $x \in O_{g_p}$ leaves the input/output membrane and passes to the next membrane of the chain.

  These rules model in detail three stages of the g_p gene transcription process. Rules, associated with the input/output membrane, model the initiation of transcription: RNA polymerase and transcription factors interaction with the gene regulatory region. Rules, associated with the membranes that map the gene-coding region, model the movement of RNA polymerase along the DNA. Rules, associated with the output membrane, model the RNA polymerase leaving the end of gene.

2 P Transducer model of the living cell

A living cell is a self-organizing, self-regulating, multi-scale, and multi-functional unit that recognizes physical and chemical stimuli and produces measurable signals in response to the environmental changes, considered as input signals. The gene regulatory networks reveal that gene expression programs and cellular functions are highly connected. Cell structure and functions are closely related and cellular membranes are essential for cell's integrity and function. The structure elements of the gene regulatory network are scattered throughout membrane-bound compartments and organelles, showing dynamic architecture [4].

Proteins (regulatory enzymes, protein transcription factors) play central role in a cell by responding to the various stimuli in the external environment and regulating the expression of specific set of genes. Our goal is to present a new view of the gene functional organization and dynamics in genome activity that provides a direct, quantitative assessment of changes in gene expression as a function of input signals. No matter which type of cell we are considering, all cells can be modelled by means of P transducers:

a) the cellular structure may be represented by cell-like membrane system;

b) the cellular tasks and functions may be modelled by the P transducer rules (Fig. 2).

Gene regulatory networks that govern cellular functions can be modelled by continuous-time P systems of elementary membranes, arranged into chains.

Definition 2 *A P transducer of the living cell is the construct:*

$$\Pi = (O_{g_p}, V, \mu_{g_p}, \mu_{cell}, w_{g_p(1)}, w_{g_p(2)}, \ldots, w_{g_p(l_p)}, w_1, \ldots, w_k,$$

$$R_{g_p(1)}, R_{g_p(2)}, \ldots, R_{g_p(l_p)}, E, R_1, \ldots, R_k), 1 \leq p \leq n,$$

where:

- n is the number of the g_p genes linked as a gene regulatory network;
- $k \geq 1$ is the degree of the hierarchical cell-like membrane system (number of cell-like membranes). The components of the Π_{g_p} systems, which map the structural-functional organization of the g_p genes, are described in the previous section;

- V is the alphabet of objects of the cell-like membrane system. $O_{g_p} \subseteq V$;
- μ_{cell} is the membrane structure of the cell-like membrane system with the *skin* membrane labelled by k;
- $w_1, \ldots, w_k$ are multisets of objects over V^* initially present in each region of the membrane structure μ_{cell}. w_1 is the multiset of objects initially present in *the environment of the* Π_{g_p} *systems*;
- $E \subseteq V$ is the set of objects which are supposed to appear in an arbitrarily large number of copies in *the environment of the P transducer*;
- $R_1, \ldots, R_k$ are finite sets of rules associated with the each membrane of the cell-like membrane system. Rules of different types, such as: evolution, symport, antiport rules, as well as rules of membrane division, dissolving and creation [6] can be used for modelling cellular functions (connected with protein networks, membrane networks, etc.). The rates of biological processes, modelled by the rules of the P transducer, are indicated as superscripts $r_m, m \in N$ associated with the rules. The rules of the P transducer are used in a non-deterministic maximally parallel manner.

  The rules associated with the *skin* membrane might contain both "unmarked" and "marked" objects; the *marked* objects are of the form $(a, read)$ or $(a, write)$, $a \in V$. $(a, read)$ can be used being associate with the command *in*, while $(a, write)$ can be used having associate the command *out*. As input, the P transducer takes input "marked" objects from the "input tape" to be translated, as well as "unmarked" objects from the environment. A *symport rule* of the form (x, in) associated with the skin membrane means that "unmarked" objects $x \in V$ enter from the environment of the P transducer into the cell-like system. Once the *marked* objects pass the skin membrane they become *unmarked*, and they can be used without restrictions by the other rules. The P transducer can provide a result in the form of the *marked* objects $a \in V$ which are written to the "output tape" [3].

  The multisets of objects u and v can evolve according to given *evolution rules* of the form $(u \to v)|^{r_m}$, meaning that u is replaced by v. Rules of the form $(u, in)|^{r_m}$ or $(u, out)|^{r_m}$, associated with a membrane, and stating that the object u can enter, respectively, exit the membrane, are *symport* rules. Some rules can be used only in the presence of certain objects and are named *conditional rules*. A conditional rule is

allowed only if the condition is realized. Rules without conditions are applied in the usual way. *Creation rules* of the form $R : (u \to v)|^{r_m}/r$, where r is a set of labels of possible rules can be used; when using the rule R (one copy of it is consumed), the rules specified by r become available for the next step [6].

3 P transducer model of the SOS response

In this section a P transducer model of conversion of environmental changes (toxicity dose) into gene expression changes and of producing of output signals (fluorescence) is described. The structure-functional organization of the *lexA recA* gene regulatory network that control SOS repair system of *E. coli* cell is mapped by means of continuous-time P systems of elementary membranes, arranged into chains. The execution of the gene machine is seen as a continuous process both in time and in concentration levels.

A scheme of transcriptional regulation of the *lexA recA* genetic system of *E.coli* cells with the negative type of gene regulation is represented in Fig. 3. The *lexA* regulatory gene codes for the *LexA* repressor proteins. When the *LexA* repressor molecules bind to the O_1 and O_2 operators, RNA polymerase is unable to begin the transcription of the *lexA* and *recA* genes (i.e., the genes are repressed). In the absence of the $LexA$ repressors the RNA polymerases, bound to the P_1 or P_2 promoter, start to move along genes, synthesizing $mRNA$ copies of the *lexA* or *recA* gene. The transcription termination sites of these genes are denoted by T_1 and T_2, respectively [5].

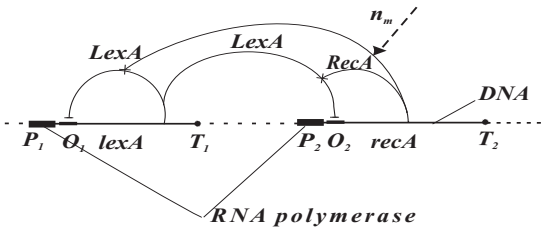


Figure 3: The scheme of the transcriptional regulation process of the *lexA recA* genetic system.

The *recA* gene codes for the *RecA* regulatory enzyme, which is a multi-functional protein, involved in the recombinational repair of damaged DNA (i.e., DNA breaks) and SOS repair [7]. It activates the synthesis of many proteins, including DNA repair proteins, that help a bacterium adapt to a

variety of metabolic stresses. DNA damages can be induced by UV irradiation (the most studied case) or various other agents, such as chemical mutagens, addition of drugs, etc. In a healthy bacterium cell there is a high $LexA$ protein concentration. $LexA$ does not completely abolish the transcription of the *lexA* and *recA* genes, these genes are expressed at low levels. The trigger that activates the SOS repair system after damage is thought to be single-strand DNA breaks ($ssDNA$) that activate the *RecA* regulatory enzymes. Activated *RecA* enzymes split *LexA* proteins. Intracellular levels of the *LexA* decrease removing the *LexA* barrier to *recA* gene transcription. Thus, one of the two alternative functioning regimes (corresponding to the activated/repressed states of genes) can be established in the molecular genetic trigger system as a result of the influence of the environmental circumstances on the regulatory enzymes.

Using methods of genetic engineering, a g gene that might code for fluorescent proteins can be closely integrated to the *recA* gene. The fluorescence, denoted by G, as indicator of gene expression, can be considered as output signal. A biosensor, used for detection of mutagenic agents, can be implemented on the basis of this genetic trigger system [8].

The P transducer of the SOS response to environmental stimuli is the construct:

$$\Pi = (O_{lexA}, O_{recA}, V, \mu_{lexA}, \mu_{recA}, \mu_{cell}, w_{lexA(1)}, w_{lexA(2)}, w_{lexA(3)}, w_{lexA(4)},$$
$$w_{recA(1)}, w_{recA(2)}, \ldots, w_{recA(5)}, w_1, R_{lexA(1)}, R_{lexA(2)}, R_{lexA(3)}, R_{lexA(4)},$$
$$R_{recA(1)}, R_{recA(2)}, \ldots, R_{recA(5)}, E, R_1),$$

where:

- O_{lexA}, O_{recA} are alphabets of objects associated with the continuous-time P systems of elementary membranes that map the structure-functional organization of the $lexA$ and $recA\&g$ genes.

  $O_{lexA} = \{L, p, p_l, n_l\}, O_{recA} = \{L, p, p_r, n_r\}$, where:

  L represents the $LexA$ repressor protein, encoded by the $lexA$ gene;

  p - RNA polymerase molecule (holoenzyme);

  p_l and p_r - RNA polymerases (without sigma subunit, which is responsible for the recognition of the promoter by RNA polimerase) transcribing the $lexA$ and $recA$ genes into $mRNA$ copies, respectively;

  n_l, n_r - number of the RNA polymerases transcribing the $lexA$ and $recA$ genes into $mRNA$.

- V is the alphabet of objects of the cell-like membrane system.
  $O_{lexA} \cup O_{recA} \subseteq V$.

  $V = \{L, R, G, p, p_l, p_r, n_l, n_r, u, v, n_m, n, b\}$, where:

  R represents $RecA$ regulatory enzymes, encoded by the $recA$ gene;

  G - output signals, encoded by the g gene;

  u - $mRNA$ copies of the $lexA$ gene;

  v - $mRNA$ copies of the $recA$ and g genes;

  n_m - input signals;

  n - $ssDNA$ breaks;

  b - objects in the environment (for instance, the temperature) which influence the biochemical reactions rates and characteristic constants of protein interaction processes with DNA loci;

- μ_{lexA}, μ_{recA} are the membrane structures of the P systems of elementary membranes that map the structure-functional organisation of the $lexA$ and $recA$ genes.

  $$\mu_{lexA} = [_{lexA(1)} \quad {}_{lexA(1)}]_{lexA(2)} \quad {}_{lexA(2)}]_{lexA(3)} \quad {}_{lexA(3)}]_{lexA(4)} \quad {}_{lexA(4)}]$$

  is the elementary membrane structure, which represent the $lexA$ gene with its regulatory region. The elementary membrane labelled by $lexA(1)$ corresponds to the regulatory region ($P1/O1$ promoter/operator site) of the $lexA$ gene. The membranes labelled by $lexA(2)$ and $lexA(3)$ represent the coding region of the $lexA$ gene. The membrane labelled by $lexA(4)$ corresponds to the transcription termination site of this gene.

  $$\mu_{recA} = [_{recA(1)} \quad {}_{recA(1)}]_{recA(2)} \quad {}_{recA(2)}]_{recA(3)} \quad \cdots \quad {}_{recA(4)}]_{recA(5)} \quad {}_{recA(5)}]$$

  is the membrane structure, modelling the $recA$ and g genes with their regulatory region. The membrane labelled by $recA(1)$ corresponds to the $P2/O2$ regulatory site of the $recA$ and g genes. The membranes labelled by $recA(2)$, $recA(3)$, $recA(4)$ represent the gene–coding region. The membrane labelled by $recA(5)$ corresponds to the transcription termination site of these genes;

- $\mu_{cell} = [_1 \quad {}_1]$ is the membrane structure of the cell-like membrane system with the *skin* membrane labelled by 1. The P transducer membrane structure consisting of the elementary membrane structures embedded into the cell-like membrane labelled by 1 is:

  $$\mu = [_1 \quad [_{lexA(1)} \quad {}_{lexA(1)}]_{lexA(2)} \quad {}_{lexA(2)}]_{lexA(3)} \quad {}_{lexA(3)}]_{lexA(4)} \quad {}_{lexA(4)}]$$

$$\left[_{recA(1)} \quad _{recA(1)}\right|_{recA(2)} \quad _{recA(2)}\left|_{recA(3)} \quad \cdots \quad _{recA(4)}\right|_{recA(5)} \quad _{recA(5)}\left.\right] \quad _1];$$

- $w_{lexA(1)}, w_{lexA(2)}, w_{lexA(3)}, w_{lexA(4)}$ and $w_{recA(1)}, w_{recA(2)}, \ldots, w_{recA(5)}$ represent the initial sets of objects over O_{lexA} and O_{recA}, associated with each membrane of the elementary membrane structures μ_{lexA} and μ_{recA}, respectively;

- w_1 is the initial multiset over V associated with the region delimited by the skin membrane (cell cytoplasm). The initial configuration of the P transducer (corresponding to the healthy cell in the absence of mutagenic agents, $n_m = 0$) can be represented as following:

  $w_{lexA(1)} = \{L\}$, $w_{lexA(2)} = \{p_l\}$,
  $w_{recA(1)} = \{L\}$, $w_{recA(3)} = \{p_r\}$,
  $w_1 = \{L, R, G, p, p_l, p_r, n_l, n_r, u, v\}$.

- $E \subseteq V$ is the set of objects which are supposed to appear in an arbitrarily large number of copies in *the environment of the P transducer*. The schematic representation of the initial state of the P transducer model of the SOS response to environmental stimuli is shown in Diagram 1.

- $R_{lexA(1)}, R_{lexA(2)}, R_{lexA(3)}, R_{lexA(4)}$ and $R_{recA(1)}, R_{recA(2)}, \ldots, R_{recA(5)}$ are finite sets of rules associated with each membrane of the membrane structures μ_{lexA} and μ_{recA}, respectively. They illustrate the communication between the elementary membranes of the chains, and between the elementary membranes and their environment. Only one object is specified in these rules. All the relevant DNA–protein interactions are modelled by these rules. In this way we model in detail the three stages of the gene transcription process: initiation of transcription, RNA polymerase movement along the DNA, and termination of transcription;

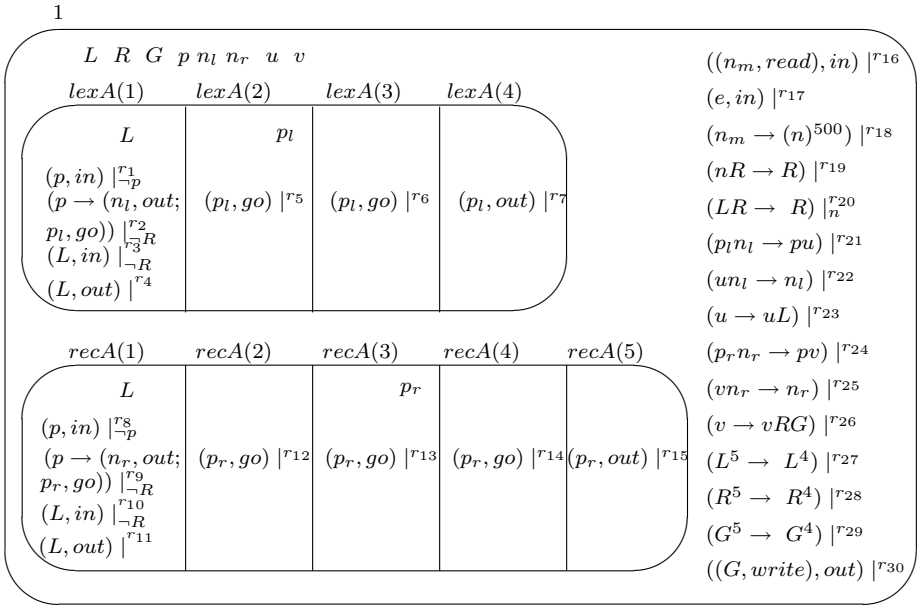


Diagram 1. Schematic representation of the initial state of the P transducer model of the *SOS* response of the *E.coli* cell to environmental stimuli. The *lexA* and *recA* genes with their regulatory regions are mapped by systems of elementary membranes. The sets of rules associated with each membrane of the P transducer model are shown.

- R_1 is a finite set of rules associated with the region delimitated by the skin membrane of the cell-like membrane system (cell cytoplasm). The rates of biological processes, modelled by the P transducer rules, are indicated as superscripts r_m , $m \in N$ associated with the rules. All available rules are applied according to their rates in a non-deterministic maximally parallel manner. Biochemical reactions (a part of cellular protein network) that evolve in the cell cytoplasm are modelled by evolution rules. The action of input signals on the living cell and the cellular response (a part of cellular membrane network) are modelled by the symport rules associated with the skin membrane of the P transducer.

To get a clearer picture of the regulatory network that control *SOS* repair and to monitoring small changes of gene activity profiles as response to external stimuli, a predictive computational Visual Hybrid Petri Nets ($VHPN$) [9] model of the P transducer of the living cell has been built.

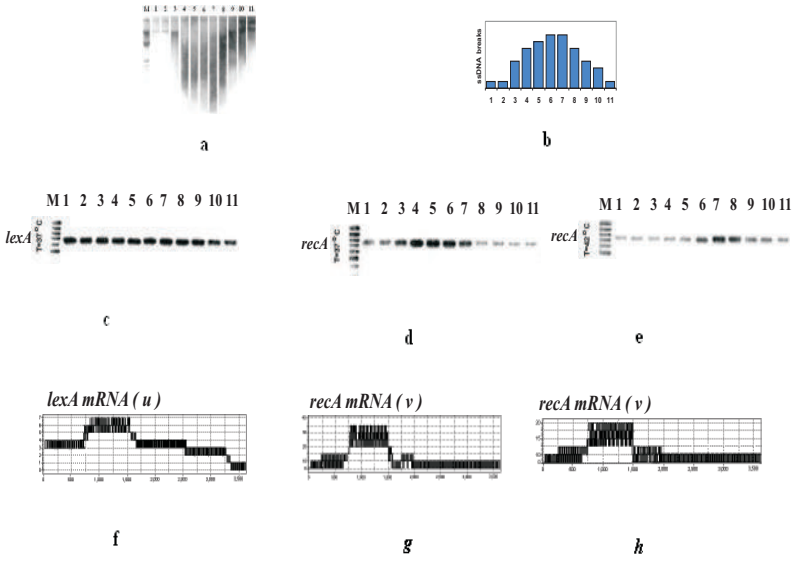


Figure 4: Electrophoretic analysis of DNA breaks in different samples of the $HB101$ strain of *E.coli* bacterial cells, conditioned by UV irradiation treatment for: 10, 15, 20, 25, 30, 35 min., denoted by $n_m = 1, \ldots 6$, (a): M – molecular markers; 1 – control (without damage treatment); 2 – 7 – DNA from bacterial cells after UV irradiation (10 min. to 35 min.); 8 – 11 – DNA from bacterial cells during DNA repair process (5 min. to 20 min.). *RT-PCR* analysis of the *lexA* and *recA* gene expression at 37^0C (c, d) and after treatment with heat shock at 42^0C (e). The simulation results (b, f, g, h) denote direct correlation between degree of cell response and the environmental changes (dose of the UV radiation)

The advantages of the Petri Nets modelling of complex and distributed systems consist in the possibility to compose models step by step from components (subsystems). The simulation of objects processing shows the dynamic evolution of the system during computation directly on the Graphical User Interface representation. The $VHPN$ model comprises discrete locations, discrete transitions, discrete test arcs, flow arcs, inhibitory arcs and guard functions. The simulation of the P transducer models permits us to analyze the processing of objects (molecules) during the computation. For every set of objects placed within each membrane and for every rule of the P transducer model we put into correspondence a discrete location and a discrete transition of the $VHPN$ model, respectively. The number of tokens

corresponds to the number of object copies. The transition firing rates are functions of the model's parameters, reflecting actual values of the rates of biochemical reactions.

In our experiments we study the coordinative transcription of the *recA* and *lexA* genes to demonstrate the differential expression of these genes as a response to low–level UV irradiation treatment (Fig. 4).

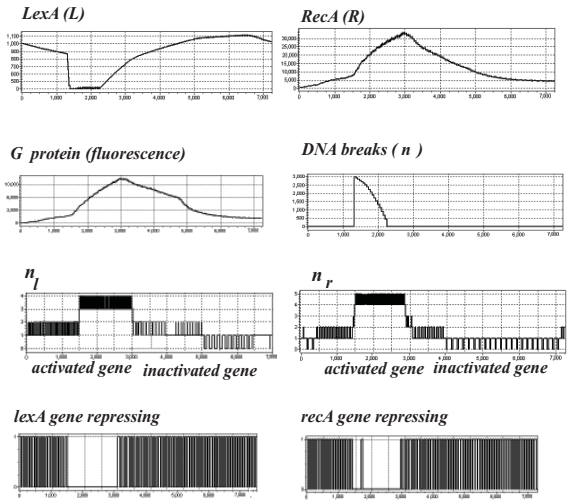


Figure 5: The simulation results of the P transducer model in the presence of mutagen agents at the 37^0C. It has been simulated the case when DNA breaks are generated in a cell as a result of DNA damaging treatment ($n_m = 6$).

Some simulation results of the P transducer model, using the $VHPN$ software tool on the basis of a special class of Timed Hybrid Petri Nets are represented in Fig. 5.

It is known that by simulating a network we can make predictions and compare them to experimental data. The previous graphics depict quantitative assessment of changes in gene expression levels as a function of input signals (toxicity dose). These results agree quite well with experimental data.

The P systems models of gene expression regulation present great interest in more efficient controlling ways of how cells self-regulate the expression of their genes *in vivo* and for adequate description of the cellular processes, and implicitly for solving different problems related to genetic medicine and

environmental diagnostics, using cellular biosensors [8].

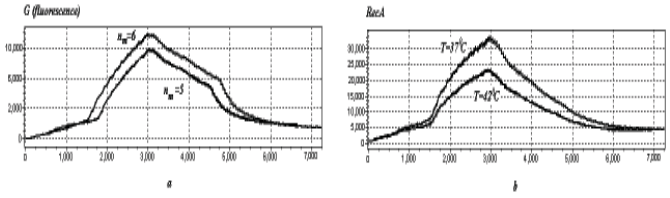


Figure 6: Correlation between the G fluorescent protein concentration (cell response) and different degrees of DNA damaging treatment (a) and dependence of the $RecA$ enzyme concentration on temperature (b).

The P transducer model of the living cell permits us to develop efficient *in vivo* computational models for monitoring and prediction gene expression and protein profiles, and for determining protein interactions. Our study is focused on elucidation of the molecular mechanisms underlying fundamental cellular processes, to decipher gene function. It is shown that the continuous-time P systems of elementary membranes, are appropriate for quantitative modelling of real gene regulatory networks.

A predictive computational $VHPN$ model of the P transducer of the living cell that allows us to measure small and gradual changes of gene expression profiles, differential gene expression, and to elucidate all the relevant molecular interaction one-by-one with DNA, all functional aspects of temporal behavior of gene expression regulation has been built. This model allows us to obtain a broader view of complex cellular response under specific conditions and to measure delicate changes of gene expression levels that are indispensable in disease diagnosis.

4 Final remarks

No conceptual framework to integrate all interacting components of the living cell has been proposed [2]. We shown that the P transducer model of the living cell is a suitable tool for unifying the main aspects of biological systems of high architectural complexity: a) structure-functional organization both at DNA and cellular levels; b) integration of all cellular networks (gene regulatory network, protein network, and membrane network) into an entire system; c) interaction of all interacting components of the main hierarchical levels; d) discrete-continuous aspects of multi-scale organization.

References

[1] D. E. Zak, R. K. Pearson, R. Vadigepalli, G. E. Gonye, J. S. Schwaber, F. J. Doyle III: Continuous-Time Identification of Gene. Expression Models. *Journal of Integrative Biology* **7** (4) (2003), 373–386.

[2] L. Cardelli: Languages and notations for systems biology. *Unconventional Programming Paradigms*. Invited talk, Le Mount St.Michel (2004).

[3] G. Ciobanu, Gh. Păun, Gh. Ştefănescu: P Transducers. *New Generation Computing* (2005).

[4] N.A. Kolchanov, O.A. Podkolodnaya, E.A. Ananko, D.A. Afonnikov, O.V. Vishnevsky, D.G. Vorobiev, E.V. Ignatieva, V.G. Levitskii, V.A. Likhoshvai, N.A. Omelyanchuk, N.L. Podkolodny, A.V. Ratushny, V.V. Suslov: An Integrated Computer System for Studying the Regulation of Eukaryotic Gene Expression. *Molecular Biology* **38** (1) (2004), 58–69.

[5] B. Lewin: *Genes*. Oxford University Press (1997).

[6] Gh. Păun: *Membrane Computing: An Introduction.* Springer-Verlag, Berlin (2002).

[7] T. Yamanaka, H. Toyoshiba, H.Sone, F.M. Parham, C.J. Portier: The TAO-Gen Algorithm for Identifying Gene Interaction Networks with Application to SOS Repair in E. Coli. *Environmental Health Perspectives* **112** (16) (2004), 1614–1621.

[8] A.W. Knight: Using yeast to shed light on DNA damaging toxins and radiation. *Analyst* **129** (2004), 866–869.

[9] E. Guţuleac, A. Răilean, C. Boşneaga: VPNP - software tool for modeling and performance evaluation using generalized stochastic Petri nets. In: *Proc. of 6th International Conference on DAS2002*, Suceava, România (2002), 243–248.

LP Colonies for Language Evolution. A Preview*

Gemma BEL ENGUIX, M. Dolores JIMÉNEZ LÓPEZ

Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
Pl. Imperial Tarraco, 1, 43005, Tarragona, Spain
E-mail: {gemma.bel,mariadolores.jimenez}@urv.net

Abstract

LP colonies, a new theoretical mechanism that integrates *colonies* and *linguistic P systems*, are introduced. The objective of the new integrative framework is to demonstrate that it is possible to give a complete description of linguistic change by means of formal tools. In this model each membrane system that form the LP colony is understood as a specialized module for each part of language that evolve in parallel. A very simple example that shows how to model a lexical transformation from Latin to Spanish by means of the interaction of two linguistic modules (phonetics and semantics) is given.

1 Introduction

In this paper we 'play' with two frameworks from the field of formal language theory in order to model a linguistic problem traditionally far away from any formalization. The two frameworks are: *membrane systems* and *grammar systems*. The linguistic problem is: *language evolution.*

Membrane systems –introduced in [35]– are models of computation inspired by some basic features of biological membranes. They can be viewed as a new paradigm in the field of natural computing based on the functioning of membranes inside the cell. Briefly, in a membrane system multisets of objects are placed in the compartments defined by the membrane structure

*This research has been supported by a Marie Curie European Reintegration Grant (ERG) under contract number MERG-CT-2004-510644.

–a hierarchical arrangement of membranes, all of them placed in a main membrane called the *skin membrane*– that delimits the system from its environment. Each membrane identifies a *region*, the space between it and all the directly inner membranes, if any exists. Objects evolve by means of reaction rules also associated with the compartments, and applied in a maximally parallel, nondeterministic manner. Objects can pass through membranes, membranes can change their permeability, dissolve and divide. Membrane systems can be used as generative, computing or decidability devices. A membrane structure is shown in Figure 1.

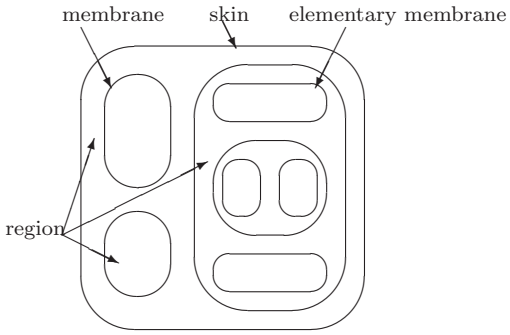


Figure 1: A Membrane Structure.

From the field of grammar systems we focus on the so-called *colonies*. Colonies as well-formalized language generating devices have been proposed in [25], and developed during the nineties in several directions in several papers, e.g. [13, 27, 33, 1, 38, 29, 30, 26, 37, 15]. A colony, that consists of a finite number of simple modules, behaves in a symbolic environment and changes its states only through acts performed by the simple agents –the components of the colony. Each state of the environment is modelled by a (finite) string of symbols. The set of all possible states of the environment is considered the language generated by the agents that form the colony. The environment itself does not change its state autonomously, but only through the activities of agents. Therefore, the global behaviour of the whole colony emerges from the strictly individual behaviours of components. The scheme of a colony is given in Figure 2.

Membrane systems and grammar systems theory offer features that seem to be very appropriate in order to model natural languages. In fact, both fields have been already applied to many natural languages issues. A general application of membrane systems to linguistics was introduced in [2]. More

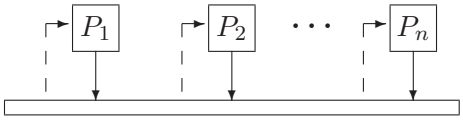


Figure 2: A Colony.

specific applications of membrane systems can be found in [3, 4, 7, 5, 6]. The most important intuition for translating this natural computing model to natural languages is that membranes can be understood as *contexts*. Contexts may be different words, persons, social groups, historical periods, languages. They can accept, reject, or produce changes in elements they have inside. At the same time, contexts/membranes and their rules evolve, that is, change, appear, vanish, etc. All these features make of membrane systems an attractive model to deal with natural language.

Defining features of grammar systems such as easy generation of *non-context-free structures* using context-free rules, *modularity*, *parallelism*, *interaction*, *cooperation*, *communication*, *distribution*, *interdependence among parts* and *emergent behaviour* have led to the idea that these systems might be applied to a large range of topics in linguistics, from pragmatics to language change. Applications of grammar systems to different linguistic issues can be found in [10, 11, 17, 18, 19, 20, 21, 22]

Both membrane systems and grammar systems have been applied to the topic of *language evolution* (cf. [2, 3, 10, 17, 20]). If we analyse such applications we realize that while the former answer in a very simple way to the question *what does it happen when a linguistic change takes place?*, the latter can formally define *how* a language innovation spreads, *who* change languages and *why* a linguistic change takes place. So, we think that there exists a complementarity between membrane systems and grammar systems. Taking into account this and considering the fact that in grammar systems we can postulate any mechanism as component of the system, in this paper we propose to define grammar systems -specifically, colonies– with membrane systems as components in order to give a complete formal-language-model of linguistic change. We are fully convinced that collaboration between both models may provide a useful formalism that, due to its naturalness and simplicity, might offer interesting results in a discipline traditionally far away from any formalization as linguistic evolution is.

2 LP Colonies

Grammar systems theory and membrane systems have been already related in papers like [16] and [9] where the notions of a P-colony and an EP-colony are introduced, respectively. Other papers in which the relation between these two frameworks is emphasized are [14, 12]. In our model, LP colonies are understood as colonies of *Linguistic P systems* (as introduced in [7]), specially defined for the modelling of linguistic generation and evolution. Both processes are usually considered as complex operations caused by the interaction of several modules: *phonetics*, *semantics*, *morphology* and *syntax*. Intuitively, each one of these parameters could be linked to different linguistic P systems, which collaborate in a modular way in the generation of linguistic objects as well as in the evolution of languages.

The P systems that form an LP colony do not need to have the same features. The semantic module, for example, may be a Dynamic Meaning Membrane System (cf. [7]), while modules devoted to phonetics and morphology may be Linguistic Membrane Systems (cf. [2, 7]).

LP systems in an LP colony cooperate by performing evolutionary steps on linguistic objects, that may be strings, symbols (form + meaning) or sentences. For an LP system to modify the linguistic object, two conditions are necessary:

- the object has to belong to the LP system's domain,
- some rule in the input membrane must allow to play with the object.

In order to modify an object, the LP system takes it from the tape. When computation in the LP system finishes, the output is replaced in the tape, and it is ready to be taken by another membrane if conditions are fulfilled. Therefore, parallelism –understood as the same object being modified by more than one LP system at the same time– is not allowed in the system. However, several LP systems can be working in parallel on different linguistic objects. So, parallelism is allowed when it is understood as different LP systems modifying different objects simultaneously, e.g. LP system 1 modifies object 1 while LP system 2 modifies object 2, LP system 3 modifies object 3, etc.

Linguistic P systems integrated in LP colonies have a special feature: the existence of *input* and *output* membranes. The *input membrane* is the one where the object goes when it is taken by the LP system. The *output*

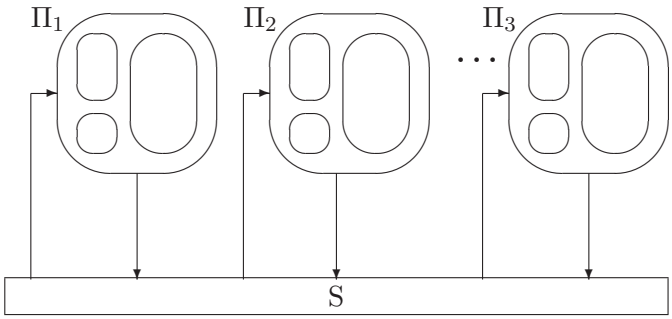


Figure 3: An LP-Colony.

membrane is the one whose objects go to the tape when the computation finishes in the LP system. It can exist more than one input and output membrane. Every object placed in an output membrane goes to the tape, therefore, the system can generate new objects.

Operations applied by membrane systems in linguistic objects may be *point mutations, duplication, transposition, splicing, rewriting, assignment of meaning*. Some modules can be specialized in some of these operations. For example, phonetic modules, which work with linguistic strings, use point mutations as a main method of evolution. Syntax, on the other hand, usually needs splicing or rewriting to explain changes in the configuration of sentences.

Before introducing the formal definition of an LP-colony we want to emphasize that what is proposed here is not at all a colony in the standard meaning of the term, i.e. a system composed of *very simple components* (regular grammars generating a finite language). Components of an LP-colony are not simple, they are rather complicated mechanisms with an intricate inner structure that perform different linguistic tasks.

2.1 Formal Definition

Definition 1 *An LP colony (LPC) is a 3-tuple* $PC = (\Pi, \mathcal{V}, S)$ *where:*

(i) $\Pi = \{\Pi_i | 1 \leq i \leq n\}$ *is a finite set of LP systems*
$\Pi_i = (\mu, \mathcal{V}_i, R_i, I_i, O_i, \mathcal{D}_i)$ *where:*

- μ_i *is the membrane structure,*

– $\mathcal{V}_i = \{V_1 \dots V_i | 1 \leq i \leq n\}$ *is the alphabet of the LP system,*

– R_i *is the set of rules,*

– I_i *is the input membrane,*

– O_i *is the output membrane,*

– $\mathcal{D}_i = \{D_i \cup E_i\}$ *is the domain of the LP system.*

This finite set of LP systems perform changes either in the form or in the meaning of a string or object S. M_i will be referred to as a component of LPC.

The domain is to be understood as the set of words, statements, ideas and other linguistic units, a membrane is able to work with in a given state of the computation. The domain is an active context.

(ii) $V = \bigcup_{i=1}^{n}(V_i)$ *is the alphabet of the LP-colony.*

(iii) $S \in \mathcal{V}$ *is the linguistic object.*

3 An LP Colony for Phonetic/Semantic Evolution

We introduce in this section a very simple system for dealing with phonetic and semantic lexical evolution. Phonetics and semantics, as independent modules, have assigned different LP systems. Therefore, we formalize an LP colony with two LP systems that interact for modelling the evolution of single words. A general scheme for this LP colony is shown in Figure 4.

Evolution described with these systems is a historical process. We are dealing, thus, with diachrony, which usually implies the transformation of a language into another one, by the transformation of lexical items. In the LP colony we describe here, we model a small aspect of the evolution from Latin to Spanish. The example aims to describe the transformation of the Latin term *regŭlam* into two different Spanish words:

- *regla* (*'rule'*), which is considered to be a *cultism* because there are only a few changes from Latin to Spanish.
- *reja* (*'plough'*), which is the *popular derivation* that has undergone a deep transformation from Latin to Spanish.

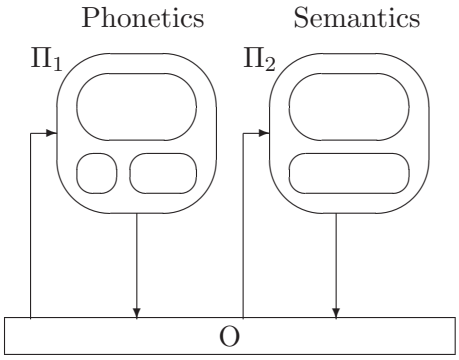


Figure 4: An LP Colony for the Interaction Phonetics-Semantics.

These Spanish words which come from the same Latin lexical item, have two different meanings, given by the semantic module of the system.

To explain the double evolution of the word and the double final output for Spanish, an LP colony is introduced, $PC = (\Pi, \mathcal{V}, O)$, where:

(i) $\Pi = \{\Pi_1, \Pi_2\}$

- $\Pi_1 = (\mu_1, \mathcal{V}_1, R_1, I_1, O_1, \mathcal{D}_1)$ where:
  - μ_1 is $[\ [\ ]_1[\ ]_2[\ ]_3]_0$
  - $\mathcal{V}_1 = \{$regŭla, regla, reja$\}$
  - R_1 is:
    - $\rho_0 =$ regŭlam $\rightarrow$ regŭla, in M_1
    - $\rho_1 =$ regŭla $\rightarrow$ regla, in M_2, M_2
    - $\rho_2 =$ regla $\rightarrow$ reja, in M_3
  - $I_1 = M_0$
  - $O_1 = M_3$
  - $\mathcal{D}_1 = V_1$
- $\Pi_2 = (\mu_2, V_2, R_2, I_2, O_2, \mathcal{D}_2)$ where:
  - μ_2 is $[\ [\ ]_1[\ ]_2]_0$
  - $\mathcal{V}_2 = V \cup \Sigma$, being $V = \{$regla, reja$\}$, $\Sigma = \{$'rule', 'plough'$\}$
  - R_2 is:
    - $\rho_0 =$ reja $\rightarrow$ reja, in M_1, M_2
    - $\rho_1 =$ 'rule' $\rightarrow$ regla

· ρ_2 = 'plough' → reja

* $I_2 = M_0$
* $O_2 = M_1$, M_2
* $\mathcal{D}_2 = D \cup E$, being $D = V$, $E = \Sigma$

(ii) $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$

(iii) *Regŭlam* is a linguistic string.

The initial configuration of the described system is shown in Figure 5.

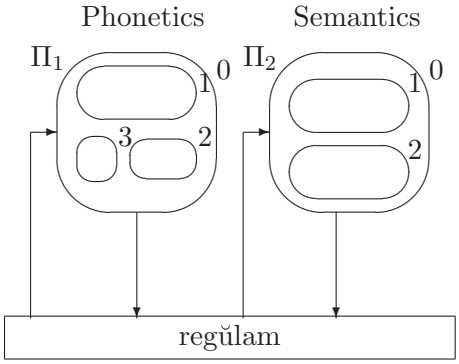


Figure 5: Initial Configuration.

In this LP colony, LP systems have nothing inside, while the object *regŭla* is placed in the tape. Π_1 starts the computations because it is the only module whose domain contains that Latin word. The first step is placing the object in the input membrane M_0. Then, the phonetic module, can start modifying the object.

Figure 6 shows the rules applied in Π_1. First, the last consonant *-m*, is deleted by ρ_0, and the result, *regŭla*, is sent to the membrane 1. In M_1 the word becomes *regla*, and it is sent to the membranes 2 and 3. M_3, the output membrane, has no rule to apply, in a way that the word goes to the tape. On the other hand, M_2 applies the rule $gl \rightarrow j$, and sends the result to M_3. From here, the word goes also to the tape.

At this moment no more rules can be applied, so phonetic module Π_1 can not play any more. The configuration of the system in this step is the one shown in Figure 7.

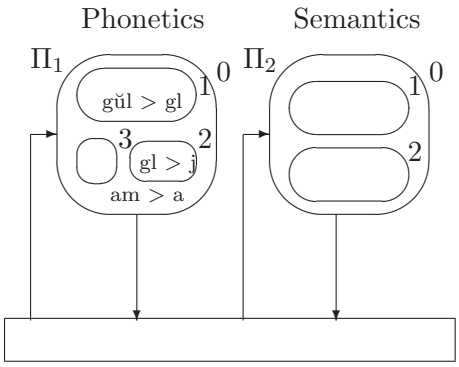


Figure 6: Work Performed by the Phonetic Module.

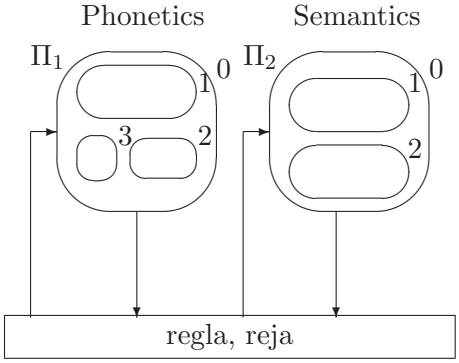


Figure 7: Output after Phonetic Evolution.

As we have seen, only one input string (*regŭla*) has generated two output strings (*regla* and *reja*). This fact demonstrates that multiple rules can be applied for the evolution of only one object. But this can be a temporary step in the pass from Latin to Spanish, since the evolution of meaning has not been tackled yet. If only one of the objects is in the domain of the second module, probably the other one will disappear because no meaning will be assigned to it.

In this case, both *regla* and *reja* are elements in the domain of Π_2, therefore, both of them are taken by the LP system and placed in its input membrane M_0. From here, *regla* is sent to M_1 and *reja* to M_2. In these membranes, the application of a semanteme to these linguemes is performed, as shown in Figure 8.

After the modifications that occur in Π_2 (the semantic module) the objects placed in the output membranes M_1 and M_2 go to the tape. No more

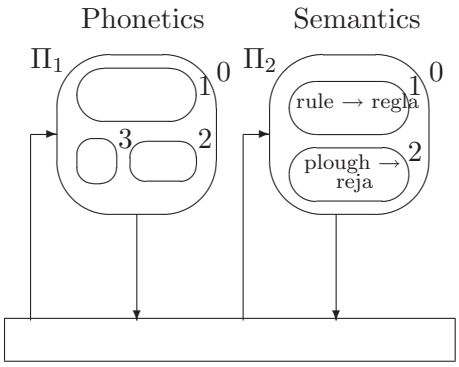


Figure 8: Assignation of Meaning by the Semantic Module.

operations can be done in the system. Therefore, the configuration shown in Figure 9 is the final one.

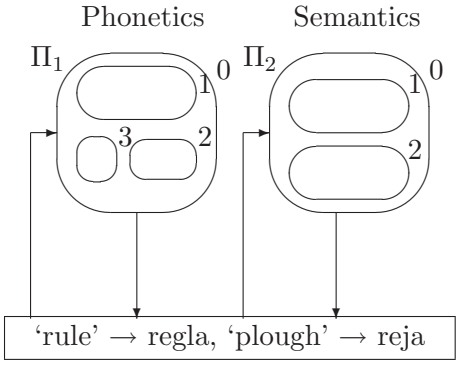


Figure 9: Final Configuration.

Therefore, by means of this simple formal system an integrative description of lexical evolution that may be implemented by formal mechanisms is given. This can be a good way to start strategies for the prediction of language evolution by the characterization of the rules, taking into account every modular aspect of linguistics.

4 Final Remarks

The paper shows how the interaction between different theories in theoretical computer science can be very fruitful in the field of linguistics. In this case, we have modelled linguistic evolution by combining P systems and colonies.

The improvement and development of this ideas could give rise to a first complete formal description of linguistic evolution.

The formalization and examples given in the paper are only a small example of the possibilities of LP colonies. In the future, in order to give account of the processes that take place in linguistic evolution, more complex devices can be formalized, introducing variations in membrane systems for different languages, dialects, periods, etc.

The practical application of the model can be the simulation and prediction of the evolution of languages. Indeed, given certain rules and contexts, probably the future shape of a given language can be foreseen, if formal concepts have been correctly described.

References

[1] I. Baník: Colonies with Position. *Computers and Artificial Intelligence* **15** (1996), 141–154.

[2] G. Bel Enguix:. Preliminaries about Some Possible Applications of P Systems in Linguistics. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing. *Lecture Notes in Computer Science* **2597**, Springer-Verlag, Berlin (2003), 74–89.

[3] G. Bel Enguix, M.D. Jiménez López: A Membrane Systems Explanation of Semantic Change. In: E. Csuhaj-Varjú, Gy. Vaszil (Eds.): *Proceedings of Grammar Systems Week 2004*, MTA Sztaki, Budapest (2004), 30–49.

[4] G. Bel Enguix, R. Gramatovici: Parsing with Active P Automata. In: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. *Lecture Notes in Computer Science* **2933**, Springer-Verlag, Berlin (2004), 31–42.

[5] G. Bel Enguix, M.D. Jiménez López: Computing Dialogues with Membranes. To appear in *LCMAS 2005*.

[6] G. Bel Enguix, M.D. Jiménez López: Computing Speech Acts. In: Ch. Bussler, D. Fensel (Eds.): Artificial Intelligence: Methodology, Systems and Applications. *Lecture Notes in Artificial Intelligence* **3192**, Springer-Verlag, Berlin (2004), 236–245.

[7] G. Bel Enguix, M.D. Jiménez López: Linguistic Membrane Systems and Applications. In: G. Ciobanu, Gh. Păun, M.J. Pérez Jiménez (Eds.): *Applications of Membrane Computing*. Springer-Verlag, Berlin (2005), 347–388.

[8] C. Calude, Gh. Păun: *Computing with Cells and Atoms*, London, Taylor and Francis (2001).

[9] E. Csuhaj-Varjú: EP-Colonies: Micro-organisms in a cell-like environment. In: *Proceedings of the Third Brainstorming Week on Membrane Computing*, Sevilla (2005), 123–130.

[10] E. Csuhaj-Varjú, M.D. Jiménez López: Cultural Eco-Grammar Systems: A Multiagent System for Cultural Change. In A. Kelemenová (Ed.), *Proceedings of the MFCS'98 Satellite Workshop on Grammar Systems*, Silesian University, Opava (1998), 165–182.

[11] E. Csuhaj-Varjú, M.D. Jiménez López, C. Martín Vide: "Pragmatic Eco-Rewriting Systems": Pragmatics and Eco-Rewriting Systems. In: Gh. Păun, A. Salomaa (Eds.): *Grammatical Models of Multi-Agent Systems*, Gordon and Breach, London (1999), 262–283.

[12] E. Csuhaj-Varjú, Gy. Vaszil: Grammar Systems vs. Membrane Computing: The case of CD Grammar Systems. Submitted.

[13] J. Dassow, J. Kelemen, Gh. Păun: On Parallelism in Colonies. *Cybernetics and Systems* **14** (1993), 37–49.

[14] R. Freund, M. Oswald: Modelling Grammar Systems by Tissue P Systems. In: E. Csuhaj-Varjú, Gy. Vaszil (Eds.): *Proceedings of the Workshop on Grammar Systems*. MTA Sztaki, Budapest (2004), 162–179.

[15] J. Gašo: Unreliable Colonies as Systems of Stochastic Grammars. *Journal of Automata, Languages, and Combinatorics* **5** (2000).

[16] J. Kelemen, A. Kelemenová, Gh. Păun: P Colonies. In: M. Bedan M. et al. (Eds.), *Workshop on Artificial Chemistry, ALIFE9*, Boston (2004), 82–86.

[17] M.D. Jiménez López: Cultural Eco-Grammar Systems: Agents between Choice and Imposition. A Preview. In: G. Tatai, L. Gulyás (Eds.): *Agents Everywhere*. Springer, Budapest (1999), 181–187.

[18] M.D. Jiménez López: Dialogue Modeling with Formal Languages. In: F. Spoto, G. Scollo, A. Nijholt (Eds.): *Algebraic Methods in Language Processing*. TWLT 21, University of Twente (2003), 207–221.

[19] M.D. Jiménez López: Linguistic Grammar Systems: A Grammar Systems Approach to Natural Languages. In: C. Martín Vide, V. Mitrana (Eds.): *Grammars and Automata for String Processing: from Mathematics and Computer Science to Biology, and Back*. Taylor and Francis, London (2003), 55–65.

[20] M.D. Jiménez López: *Using Grammar Systems*. GRLMC Report, Rovira i Virgili University, Tarragona (2002).

[21] M.D. Jiménez López, C. Martín Vide: Grammar Systems and Autolexical Syntax: Two Theories, One Single Idea. In: R. Freund, A. Kelemenová (Eds.): *Grammar Systems 2000*, Silesian University, Opava (2000), 283–296.

[22] M.D. Jiménez López, C. Martín Vide: Grammar Systems for the Description of Certain Natural Language Facts. In: Gh. Păun, A. Salomaa (Eds.): New Trends in Formal Languages. *Lecture Notes Computer Science* **1218**, Springer-Verlag, Berlin (1997), 288–298.

[23] J.J. Katz: *Propositional Structure and Illocutionary Force*. Crowell, New York (1977).

[24] J.J. Katz, J.A. Fodor: The Structure of a Semantic Theory. *Language* **39** (1963), 170–210.

[25] J. Kelemen, A. Kelemenová: A Grammar-Theoretic Treatment of Multiagent Systems. *Cybernetics and Systems* **23** (1992), 621–633.

[26] A. Kelemenová: Timing in colonies. In: Gh. Păun, A. Salomaa (Eds.): *Grammatical Models of Multi-Agent Systems*. Gordon and Breach, London (1999).

[27] A. Kelemenová, E. Csuhaj-Varjú: Languages of Colonies. *Theoretical Computer Science* **134** (1994), 119–130.

[28] S. Levinson: *Pragmatics*. Cambridge University Press, London (1983).

[29] C. Martín-Vide, Gh. Păun: PM-colonies. *Computers and Artificial Intelligence* **17** (1998), 553–582.

[30] C. Martín-Vide, Gh. Păun: New Topics in Colonies Theory. *Grammars* **1** (1999), 209–223.

[31] C.W. Morris: Foundations of the Theory of Signs. In: O. Neurath, R. Carnap, C. Morris (Eds.): *International Encyclopaedia of Unified Science*, University of Chicago Press (1938), 77–138.

[32] B. Moulin, D. Rousseau, G. Lapalme: A Multi-Agent Approach for Modelling Conversations. In: AI'94, *Natural Language Processing. Proceedings of the Fourtheenth International Avignon Conference*. Paris, vol. 3 (1994), 35–50.

[33] Gh. Păun: On the Generative Power of Colonies. *Kybernetika* **31** (1995), 83–97.

[34] Gh. Păun: Computing with membranes: An introduction. *Bulletin of the EATCS* **67** (1999), 139–152.

[35] Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (2000), 108–143.

[36] Gh. Păun: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin (2002).

[37] P. Sosík: Parallel Accepting Colonies and Neural Networks. In: Gh. Păun, A. Salomaa (Eds.): *Grammatical Models of Multi-Agent Systems*. Gordon and Breach, London (1999).

[38] P. Sosík, L. Štýbnar: Grammatical Inference of Colonies. In: Gh. Păun, A. Salomaa (Eds.): *New Trends in Formal Languages*. Springer-Verlag, Berlin (1997).

[39] C. Tagliavini: *Orígenes de las Lenguas Latinas*. Fondo de Cultura Económica, Mexico (1973).

On P Systems as a Modelling Tool for Biological Systems

Francesco BERNARDINI[1], **Marian GHEORGHE**[1], **Natalio KRASNOGOR**[2], **Ravie C. MUNIYANDI**[1], **Mario J. PÉREZ-JIMÉNEZ**[3], **Francisco J. ROMERO-CAMPERO**[3]

[1]Department of Computer Science
The University of Sheffield
Regent Court, Portobello Street
Sheffield, S1 4DP, UK
E-mail: {F.Bernardini,M.Gheorghe,R.Muniyandi}@dcs.shef.ac.uk

[2]Automated Scheduling, Optimisation and Planning Research Group
School of Computer Science and Information Technology
University of Nottingham, Jubilee Campus
Nottingham NG8 1BB, UK
Email: Natalio.Krasnogor@nottingham.ac.uk

[3]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Seville
Avda. Reina Mercedes
41012 Sevilla, Spain
Email: {Mario.Perez,Francisco-Jose.Romero}@cs.us.es

Abstract

A specific type of P systems is considered and its behaviour defined in terms of mass action law strategy. Such an approach is applied to the modelling of the quorum sensing regulatory networks of the bacterium *Vibrio fischeri*. In this respect, a formalisation of the network in terms of P systems is provided and some simulation results concerning the behaviour of a colony of such bacterium are reported. We also briefly describe the implementation technique adopted by pointing out the generality of our approach which appears to be fairly independent from

the particular choice of P system variant and the language used to implement it.

1 Introduction

Membrane computing represents a new and rapidly growing research area which is part of the natural computing paradigm. Already a monograph has been dedicated to this subject [10] and some fairly recent results can be found in [11],[6],[7]. Membrane computing has been introduced with the aim of defining a computing device, called P system, which abstracts from the structure and the functioning of living cells [9]. The main results in this area show that P systems are a very powerful computational model mostly equivalent to Turing machines. Recent researches have been instead dedicated to the study of P systems as a modelling tool for biological systems [3], [4], [12] and system biology appears now to be a promising field of application for membrane computing. In this domain of application, P systems are not used as a computing paradigm, but rather as a formalism for describing the behaviour of the system to be modelled. Therefore, there is a growing interest in developing implementations for the membrane computing paradigm in order to be able to execute P system model and run simulations of biological phenomena of various interest. In this respect, a number of tools have already been produced (some of them are available from [14]) but yet correct implementation techniques need to be identified, especially when the quantitative aspects featuring the "reality" of a biological phenomenon are considered in the model.

In this paper, we present a variant of P systems (Section 2) where rules are generalised boundary rules which allow us to express transformations affecting simultaneously the objects placed on both sides of a membrane, that is, both the objects placed inside that compartment and the objects placed into the surrounding region. As well as this, each rule has associated a real number providing a measure for the "intrinsic reactivity"of the rule and roughly corresponding the kinetic coefficient which, in bio-chemistry, is usually associated to each molecular reaction [12]. A mass action law will be used to randomly pick up the next rule to be applied. The main difference with respect to the usual approach adopted in membrane computing is that, in our approach, there is no parallelism in the application of the rules as the system evolves only by means of a rule at a time. Next, in Section 3, we present, as a case-study, a P system model for the quorum sensing system of the marine bacterium *Vibrio fischeri* together with some simulation results

obtained by implementing the model in Scilab, a freely available software package [16]. The novelty of our approach consists in the fact that we do not only provide a description for the reactions involved in the quorum sensing regulatory network but we are also able to provide a model for an arbitrarily large colony of bacteria. In this respect, we can say our simulations provides a snapshot of the behaviour of the colony as a whole complex system. Finally, the last section describes implementation techniques for our P system model by presenting the data structures and the code that are necessary to support its execution. Moreover, by following [8], we advocate the use of the mark-up language SBML as a "machine-interpretable" language for defining executable specifications of P systems and the corresponding code that can be automatically generated from. In this respect, we want to stress the generality of our approach which appears to be fairly independent with the particular choice of a P system variant, the language used to implement it, and flexible with the strategy for applying the rules.

2 The P System Model

The P system model used in this paper is a variant of cell-like P systems with the rules being a generalisation of the boundary rules introduced in [2] and having associated a certain rate of application. Cell-like P systems means systems with a structure which is a tree of nested compartments delimited by a corresponding number of membranes. Specifically, each membrane separates the inside of a compartment from its outside represented by the region internal to the compartment associated with its parent node. In other words, inside of a compartment is the space between its delimiting membranes and the membranes delimiting its directly inner compartments (i.e., its directed descendants in the tree structure). Thus, the objects contained in a compartment are supposed to be placed into this region and rules associated with the compartment can be applied only to these specific objects. Such a tree of nested compartments delimited by membranes is called a membrane structure.

Membrane structures are represented as usual by means of strings of matching pairs of square-brackets, with each pair of square-brackets representing a membrane and each one of them being labeled with a different value in $\{1, 2, \ldots, n\}$, for n the number of membranes in the structure. For instance, the string $[\,[\,[\,]_3\,]_2\,[\,[\,[\,]_6\,]_5\,]_4\,[\,]_7\,]_1$ represents a membrane structure with seven membranes and seven compartments. Notice that the order of the membranes placed at same level does not matter. We will use the ex-

pression "compartment i" (or "membrane i"), with $1 \leq i \leq n$, to refer to a certain compartment (or a certain membrane).

A P system is then defined in the following way.

Definition 2.1. *A P system is a construct*

$$\Pi = (O, L, \mu, C_1, C_2, \ldots, C_n, R)$$

where:

- *O is a finite alphabet of symbols representing objects;*
- *L is a finite alphabet of symbols representing labels for the compartments;*
- *μ is a membrane structure consisting of $n \geq 1$ membranes labeled in an one-to-one manner with values in $\{1, 2, \ldots, n\}$;*
- *$C_i = (l_i, w_i)$, for each $1 \leq i \leq n$, is the initial configuration of the compartment i with $l_i \in L$ and $w_i \in O^*$ a finite multiset of objects;*
- *R is a finite set containing $m \geq 1$ rules that are labelled in one-to-one manner with values in $\{1, 2, \ldots, m\}$ and that are of the form*

  $$j : u\, [\, v\,]_l \overset{k_j}{\rightarrow} u' [\, v'\,]_l$$

  with $1 \leq j \leq m$, $u, v, u', v' \in V^*$ some finite multisets of objects, $l \in L$ a label for the compartment, and k_i a real number.

Thus, a P system is characterised by a finite alphabet O for the objects placed into the compartments, a finite alphabet L for labelling the compartments, a membrane structure μ, an initial configuration for each compartment in the system, and a finite set R containing rules describing transformations that can occur to the objects placed inside the compartments.

Specifically, the initial configuration of a compartment consists of a label from the alphabet L and a finite multiset of objects from O represented as a string in O^*; these objects are those which are initially placed inside that compartment. The rules R instead are generalized boundary rules where, with respect to the original definition proposed in [2], transformation can occur on both sides of the membrane without distinguishing between transformation rules and communication rules. Thus, rules of this form allow us to capture generic interactions occurring at the level of membranes between

the internal region and the surrounding region of a given compartment. Moreover, each rule in R has associated a real constant which is meant to provide a measure of the "reactivity" of the rule and which, in general, affect the "mass action law" associated to the rule to be applied in the next step of evolution.

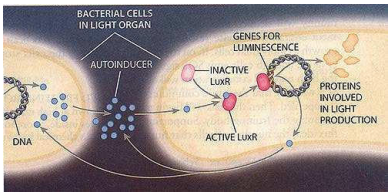
Here, as opposite to the maximal parallel approach usually considered in membrane computing, a strategy for the application of the rules is adopted that makes the system evolve only by means of a rule at a time; that is, in each step, only one rule is selected to be applied inside a specific compartment, which is the unique one to evolve in that step of computation.

3 Modelling Quorum Sensing in *Vibrio fischeri*

Bacteria are generally considered to be independent organisms. However it has been observed that certain bacteria, like the marine bacterium *Vibrio fischeri*, exhibit coordinated behaviour which allows an entire population of bacteria to regulate the expression of certain or specific genes in a coordinated way depending on the size of the population. This cell density dependent gene regulation system is referred to as *Quorum Sensing* [13],[5].

This phenomenon was first investigated in the marine bacterium *Vibrio fischeri*. This bacterium exists naturally either in a free-living planktonic state or as a symbiont of certain luminescent squid. The bacteria colonise specialised light organs in the squid, which cause it to luminesce. Luminescence in the squid is thought to be involved in the attraction of prey, camouflage and communication between different individuals. The source of the luminescence is the bacteria themselves. The bacteria only luminesce when colonising the light organs and do not emit light when in the free-living state. The Quorum Sensing System in *Vibrio fischeri* relies on the synthesis, accumulation and subsequent sensing of a signal molecule, 3-oxo-C6-HSL, an N-acyl homoserine lactone or AHL, we will call it OHHL. When only a small number of bacteria are present these proteins are produced at a low level. OHHL diffuses out of the bacterial cells and into the surrounding environment. At high cell density the signal accumulates in the area surrounding the bacteria and can also diffuse to the inside of the bacterial cells. The signal is able to interact with the LuxR protein to form the complex LuxR-OHHL. This complex binds to a region of DNA called the Lux Box causing the transcription of the luminescence genes, a small cluster of 5 genes, luxCDABE. as well as the transcription of LuxR and OHHL, which are therefore called autoinducers as they activate their own synthesis.

In this way, bacteria can effectively communicate each other by responding to changes in the concentration of signal molecules in their inside and in the surrounding environment. A comprehensive literature about quorum sensing in bacteria can be found in [13].



Next, a model for quorum sensing in *Vibrio fischeri* is obtained by considering a P system consisting of a number of distinct compartments placed inside an unique main membrane, which represents the environment, and where each one of these compartments represents a bacterium and contains rules describing the reactions involved in the regulation of the luminescence genes. Compartments representing bacteria interact each other by sending objects into the environment and receiving some others from it. Specifically, given a population of $m \geq 1$ bacteria, we define the P system $\Pi(m)$ such that

$$\Pi(m) = (O, \{e, b\}, \mu, C_1, C_2, \ldots, C_m, C_{m+1}, R)$$

and where:

- $O = \{OHHL, LuxR, LuxR\text{-}OHHL, LuxBox\} \cup$
  $\cup \{LuxBox\text{-}LuxR\text{-}OHHL\}$,
- $\mu = [\,[\,]_1\,[\,]_2 \ldots [\,]_m\,]_{m+1}$,
- $C_i = (LuxBox, b)$, for each $1 \leq i \leq m$,
- $C_{m+1} = (\lambda, e)$,
- $R = R_b \cup R_e$ with R_b the set of rules to be used inside compartments labeled by b and R_e the set of rules to be used inside the compartment labeled by e. Each compartment labeled by b represents a bacterium whereas the unique compartment labeled by e represents the environment.

Notice that the P system $\Pi(m)$ is a parametric one as its exact definition depends on the value m, the number of bacteria in the colony, as well as on the choice of the real number associated with each rule in $R = R_b \cup R_e$.

The set R_b contains the following rules.

An unstressed bacterium produces the signal OHHL and the protein LuxR at basal rates, very low rates that compensate with the degradation rates:

$$1 : [\ LuxBox\ ]_b \xrightarrow{k_1} [\ LuxBox,\ OHHL\ ]_b,$$
$$2 : [\ LuxBox\ ]_b \xrightarrow{k_2} [\ LuxBox,\ LuxR\ ]_b.$$

The protein LuxR acts as a receptor and OHHL as its ligand. Both together form the complex LuxR-OHHL which in turn can dissociate into OHHL and LuxR again:

$$3 : [\ LuxR,\ OHHL\ ]_b \xrightarrow{k_3} [\ LuxR\text{-}OHHL\ ]_b,$$
$$4 : [\ LuxR\text{-}OHHL\ ]_b \xrightarrow{k_4} [\ LuxR,\ OHHL\ ]_b.$$

The complex LuxR-OHHL acts as a transcription factor or as a promoter binding to a region of the bacterium DNA called LuxBox and starting the transcription of different proteins involved in the production of light. The complex LuxR-OHHL can also dissociate from the LuxBox:

$$5 : [\ LuxBox,\ LuxR\text{-}OHHL\ ]_b \xrightarrow{k_5} [\ LuxBox\text{-}LuxR\text{-}OHHL\ ]_b,$$
$$6 : [\ LuxBox\text{-}LuxR\text{-}OHHL\ ]_b \xrightarrow{k_6} [\ LuxBox,\ LuxR\text{-}OHHL\ ]_b.$$

The binding of the complex LuxR-OHHL to the LuxBox produces a massive increase of the production of the signal OHHL and of the protein LuxR. In this sense OHHL and LuxR are autoinducers:

$$7 : [\ LuxBox\text{-}LuxR\text{-}OHHL\ ]_b \xrightarrow{k_7} [\ LuxBox\text{-}LuxR\text{-}OHHL,\ OHHL\ ]_b,$$
$$8 : [\ LuxBox\text{-}LuxR\text{-}OHHL\ ]_b \xrightarrow{k_8} [\ LuxBox\text{-}LuxR\text{-}OHHL,\ LuxR\ ]_b.$$

OHHL is a small molecule that diffuses outside the bacterium and so it can accumulate in the environment:

$$9 : [\ OHHL\ ]_b \xrightarrow{k_9} OHHL\ [\ ]_b.$$

Due to the presence of proteases and other chemical substances OHHL, LuxR and the complex LuxR-OHHL undergo a process of degradation in the bacterium:

$$10 : [\ OHHL\ ]_b \xrightarrow{k_{10}} [\ ]_b,$$
$$11 : [\ LuxR\ ]_b \xrightarrow{k_{11}} [\ ]_b,$$
$$12 : [\ LuxR\text{-}OHHL\ ]_b \xrightarrow{k_{12}} [\ ]_b.$$

The set R_e contains the following rules. When the signal OHHL accumulates in the environment it can diffuse inside the bacteria. OHHL also undergoes a process of degradation in the environment

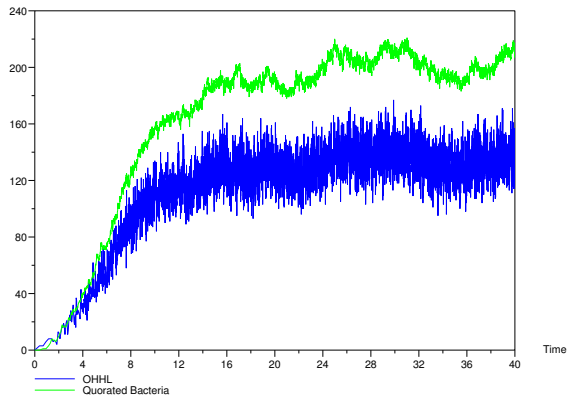
$$13 : OHHL\,[\ ]_b \overset{k_{13}}{\rightarrow} [\,OHHL\,]_b,$$
$$14 : [\,OHHL\,]_e \overset{k_{14}}{\rightarrow} [\ ]_e.$$

3.1 Simulation Results and Discussion

The quorum sensing model has been implemented using SciLab, a scientific software package for numerical computations providing a powerful open computing environment for engineering and scientific applications [16].

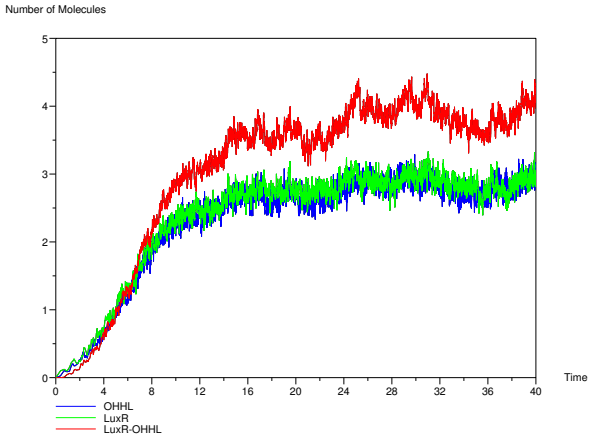
In order to implement our model we have chosen the following kinetic constants $k_1 = 2$, $k_2 = 2$, $k_3 = 9$, $k_4 = 1$, $k_5 = 10$, $k_6 = 2$, $k_7 = 250$, $k_8 = 200$, $k_9 = 1$, $k_{10} = 50$, $k_{11} = 30$, $k_{12} = 15$, $k_{13} = 20$, $k_{14} = 20$. These values have been set such that the degradation rates $(k_{11}, k_{12}, k_{13}, k_{14})$ compensate the basal production of the signal and the protein (k_1, k_2) and such that the production rates when the regulatory region is occupied (k_7, k_8) produce a massive increase in the transcription of the signal and the protein. In particular, when this region is occupied, we say the bacterium is quorated. We have studied the behaviour of the system for populations of different sizes to examine how bacteria can sense the number of bacteria in the population and produce light only when the number of individuals is big enough.

First we have considered a population of 300 bacteria. Next we show the evolution over time of the number of quorated bacteria and the number of signal (OHHL) in the environment.



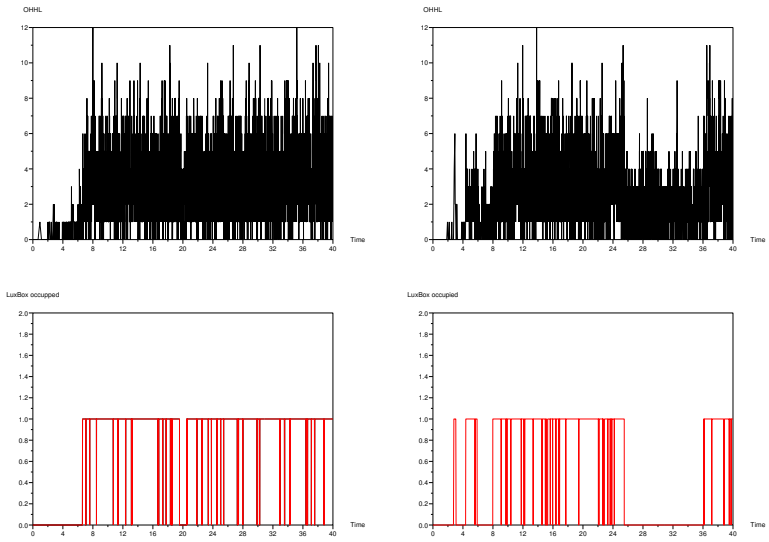
It may be observed that the signal, OHHL, accumulates in the environment until saturation and then, when this threshold is reached, bacteria are able to detect that the size of the population is big enough. At the beginning, a few bacteria get quorated and then they accelerate a process of recruitment that makes the whole population behave in a coordinated way. There exists a correlation between the number of signals in the environment and the number of quorated bacteria such that, when the number of signals in the environment drops, so does the number of quorated bacteria and when the signal goes up it produces a recruitment of more bacteria.

Now we show the evolution over time of the average bacterium across the population of the number of signal (OHHL), protein (LuxR) and the complex (LuxR-OHHL).



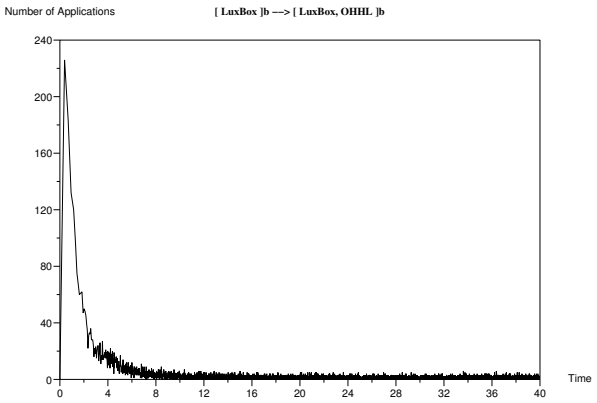
Note that in the average bacterium there is also a correlation among the signal OHHL, the protein LuxR and the complex LuxR-OHHL. Besides the patterns in the evolution of the average number of complexes across the population and the number of quorated bacteria are similar.

In our approach the behaviour of each individual in the colony can be tracked. We have taken a sample of two bacteria and have studied the correlation between the amount of signal inside each bacterium (first row) and the occupation of the LuxBox by the complex (second row) which represents that the bacterium has been quorated.

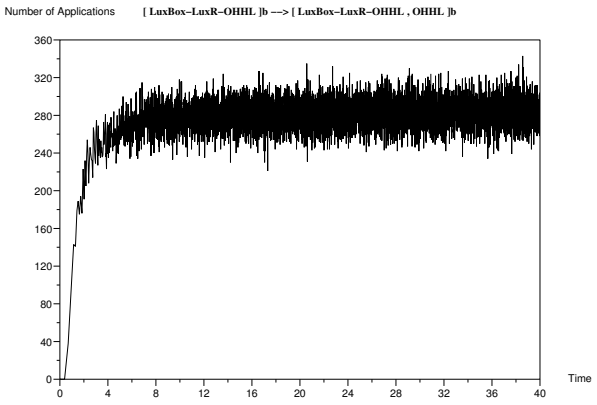


Above it is shown that the number of signal molecules inside the bacterium has to exceed a threshold in order to recruit the bacterium. It may be observed that when the number of molecules is greater than the threshold the bacterium gets quorated or up-regulated, but when there are less signals molecules the bacterium switches off the system and goes down-regulated.

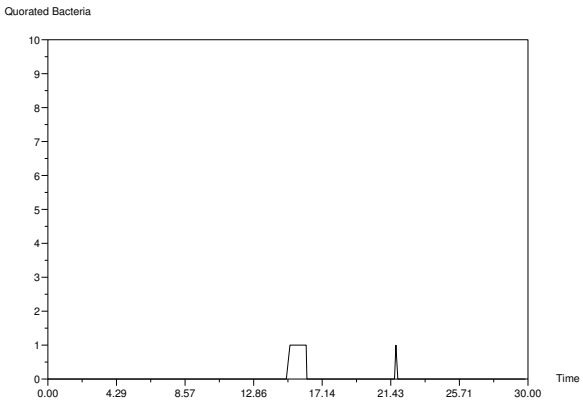
We can also study how rules are applied across the evolution of the system. For instance, we can show the evolution of the number of applications of the rule representing the basal production of the signal OHHL. and the number of applications of the rules representing the production of the signal OHHL after the binding of the complex to the LuxBox.

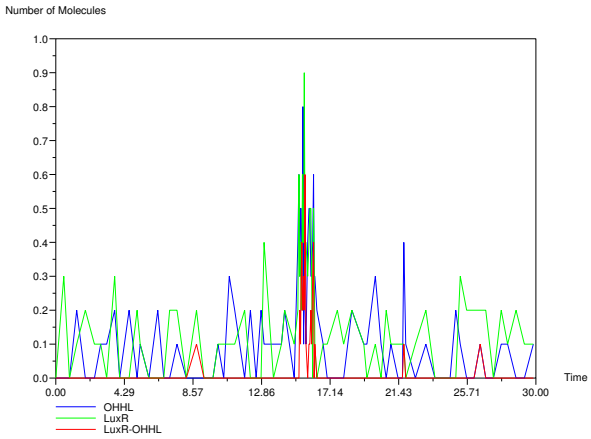


This can be compared with the number of applications of the rules representing the production of the signal OHHL after the binding of the complex to the LuxBox. In this way, we can show how at the beginning the basal production rule is the most applied rule while the other one is seldom applied. But then, as a result of the recruitment process the bacteria sense the size of the population and they behave in a coordinate way applying massively the third rule. So the system moves from a down-regulated state to an up-regulated one where the bacteria are luminescence. Specifically, this can be clearly seen if you compare the last graph above with the next one. Two similar graphs can be obtained for the rules producing the protein LuxR



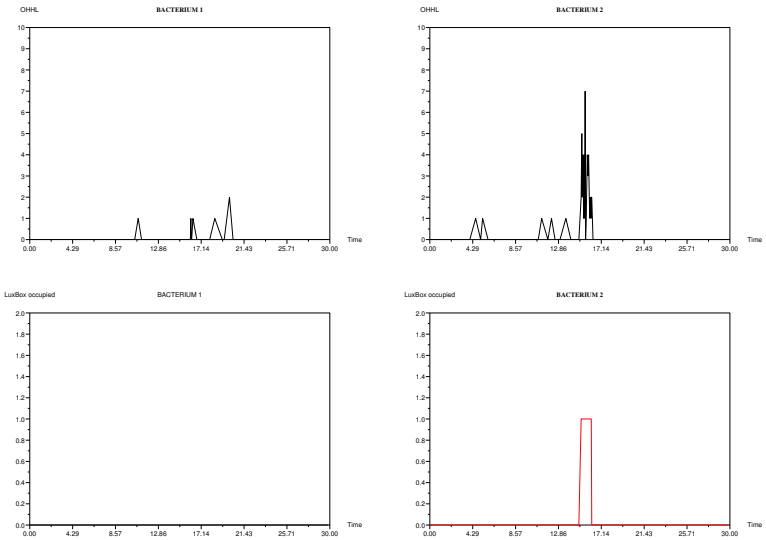
Finally, in order to study how bacteria can sense the number of individuals in the colony and get quorated only when the size of the colony is big enough, we have examined the behaviour of a population of only 10 bacteria.



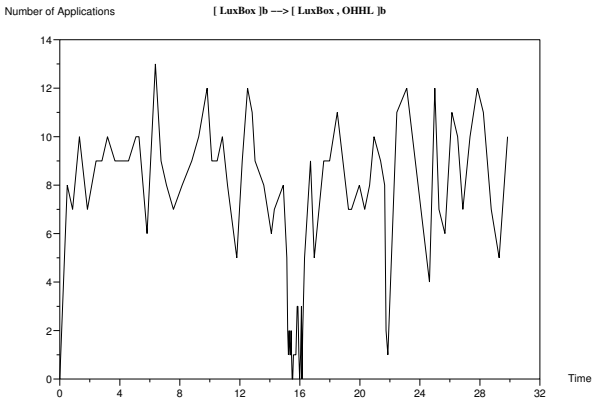


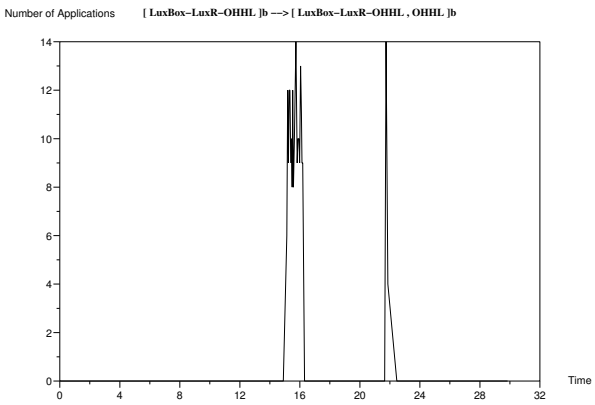
For the case of only 10 bacteria the recruitment process does not take place. Only one of the bacteria guessed wrong the size of the population and got up-regulated but then it switches off after sensing that the signal does not accumulate in the environment. The average number of molecules shows no pattern which means that the colony is not coordinating its behaviour.

Below it is depicted the behaviour of two bacteria in the population; one that never got quorated and the one that got quorated. Observe that this bacterium got quorated because the number of signal inside it exceeded the threshold of 7 signals.



Finally, observe that in this situation the system remains in a down-regulated state only applying the rules representing the basal productions while the rules associated with the production of light are seldom applied. This is illustrated in the next two graphs, which report the number of application of the rules producing the signal OHHL.





Summing up our simulations shows that *Vibrio fischeri* has a quorum sensing System where a single bacterium can guess that the size of the population is big enough and start to produce light. Then this bacterium starts to massively produce signals, if the signal does not accumulate in the environment meaning that the guess was wrong it switches off. On the other hand if the signal does accumulate in the environment meaning that the number of bacteria in the colony is big a recruitment process takes place that makes the whole population of bacteria to luminescence. These results agree well with in vitro experiments and with results obtained using differential equations [5].

4 Implementation of the P System Model

We implemented the P system model of Definition 2.1 by following the approach proposed in [8] that is based on an initial specification in SBML of the model and a subsequent automatic generation of the executable code. In this section, we describe the data structures necessary to support the execution of our variant of P systems and we provide some details about the implementation of the procedure for the application of the rules. The language chosen is Scilab but similar considerations may apply to other commonly-used programming languages, such as C, Java, MatLab. Moreover, our approach appears to be fairly independent from the particular choice of P system variant. An SBML specification of the P system modelling quorum sensing in *Vibrio fischeri* is instead reported in appendix.

The data structure used to represent the different components of P systems are the following:

• **Rules:**

Recall that we are using rules of the form:

$$j : u\ [\ v\ ]_l \overset{k_j}{\rightarrow} u'[\ v'\ ]_l$$

Which will be represented as:

$$Comp \quad father(l) \quad l \quad k_j \quad multisets$$

with $multisets = length(u)\ u\ length(v)\ v\ length(u')\ u'\ length(v')\ v'$ and where: $Comp$ represents the compartment where the rule j can be applied, $father(l)$ represents the father of the membrane with label l in the membrane structure, l is the label of the compartment involved in the rule and k_j is the kinetic constant. $length(u)$, $length(v)$, $length(u')$ and $length(v')$ tell us the size of u, v, u' and v' respectively for the rule j. And u, v, u' and v' are the string of objects representing the reactants and products of the rule j.

• **Compartments:**

Each compartment is represented by:

$$label \quad n\text{-}copies \quad multiplicity\text{-}of\text{-}o_1 \quad \cdots \quad multiplicity\text{-}of\text{-}o_n$$

The first component represents the label associated with the compartment, the second component is the number of instances of the compartment in the initial configuration; the content is instead represented by reporting for each object $o_i \in O$ its corresponding multiplicity inside that compartment.

• **Configurations of the system:**

A configuration of the system is made up of compartments; each compartment is represented:

$$identifier \quad label \quad multiplicity\text{-}of\text{-}o_1 \quad \cdots \quad multiplicity\text{-}of\text{-}o_n$$

$identifier$ is an index associated in a one-to-one manner with each compartment, $label$ is the label of the compartment and the last n components of the row are the multiplicities of the objects in the compartment in the current configuration of the system.

In what follows we briefly describe the implementation using the data structure specify above:

1. **Initialisation**

   - Set the time of the simulation $t = 0$ and the number of applied rules $r = 0$.
   - Load the rules.
   - Load the initial configuration.
   - Create a stack of rules to be apply according to the semantics of the application of the rules.

2. **Iteration**

   - Take the first rule in the stack of rules to be applied.
   - Move to the next configuration of the system by applying the rule, that is, increase the multiplicity of the products (right hand side) and decrease the multiplicity of the reactants (left hand side).
   - Update the time of the simulation, the number of applied rules and the stack of rules to be applied.

3. **Termination**

   - Terminate the simulation when time of the simulation t reaches or exceeds a preset maximal time of simulation or alternatively when the number of applied rules r reaches or exceeds a preset maximal number of rules applications.

5 Conclusions and Future Work

There is a growing interest in membrane computing in using P systems for modelling biological systems. This often requires the introduction into the model of quantitative aspects featuring the "reality" of the biological phenomenon to be modelled which are not usually considered in the abstract model of P systems. In this paper, these quantitative aspects have been considered for P systems by associating to each rule a real number (i.e., a kinetic constant), and by defining a mass law action strategy for the application of the rules. This approach has been used to model the quorum sensing process in a colony of *Vibrio fischeri* bacteria by obtaining some simulation results which show the transition from a population of down-regulated cells to a population of up-regulated cells.

Our interest for the future is in developing a flexible software platform for running *in silico* experiments that integrates tools for the specification, execution and verification/validation of P system models. The details of the implementation provided in this paper can be viewed as a first step in this direction. A model checking approach is now being investigated that is based on Maude term rewriting tool [1]. In this framework, a central issue is the integration of the specification at individual level (e.g., a bacterium) with the specification at population level (e.g., the colony) such us to allow us to model more complex and larger biological systems. In this respect, a number of case studies need to be identified together with appropriate simulation/validation/verification techniques.

Acknowledgements

The research of F.B. and M.G. has been supported by the Engineering and Physical Sciences Research Council of United Kingdom (EPSRC), grant GR/R84221/01. N.K. acknowledges the support of the Biotechnological and Biological Sciences Research Council (BBSRC) for supporting his grant BB/C511764/1 and also to the EPSRC for the grant GR/T07534/01. The research of M.P. and F.R. has been supported by the Ministero de Ciencia y Tecnología of Spain, by the Plan Nacional de I+D+I (2002-2003) (TIC-2002-04220-C03-01), co-financed by FEDER funds, and by a FPI fellowship from the Universidad de Seville, Spain.

References

[1] O. Andrei, G. Ciobanu, D. Lucanu: Executable Specifications of P Systems. In [7], 126–145.

[2] F. Bernardini, V. Manca: P Systems with Boundary Rules. In [11], 107–118.

[3] D. Besozzi: *Computational and Modelling Power of P systems*. Ph.D. Thesis, Università degli Studi di Milano, Milan, Italy (2004).

[4] L. Bianco, F. Fontana, G. Franco, V. Manca: P Systems for Biological Dynamics. In: G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez (Eds.): *Applications of Membrane Computing*. Springer-Verlag, Berlin (2005), 81–126.

[5] T. Fargerström, G. James, S. James, S. Kjelleberg, S. Nilsson: Luminescence Control in the Marine Bacterium. *Vibrio fischeri*: An Analysis of the Dynamics of lux Regulation. *J. Mol. Biol.* **296** (2000), 1127–1137.

[6] C. Martin-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Revised Papers. Membrane Computing. International Workshop, WMC 2003, Tarragona, Spain, July 2003. *Lecture Notes in Computer Science* **2933**, Springer-Verlag, Berlin (2004).

[7] G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.): Revised and Invited Papers. Membrane Computing. International Workshop, WMC 2004, Milan, Italy, June 2004. *Lecture Notes in Computer Science* **3365**, Springer-Verlag, Berlin (2005).

[8] I. Nepomuceno, J.A. Nepocumeno, F. Romero-Campero: A Tool for Using the SBML Format to Represent P System which Model Biological Reaction Networks. In: *Proceeding of the Third Brainstorming Week in Membrane Computing, Seville, Spain, January 31st-February 4th, 2005*, University of Seville, Seville, Spain (2005).

[9] Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, **61** (1) (2000), 108–143.

[10] Gh. Păun: *Membrane Computing: An Introduction.* Springer-Verlag, Berlin (2002).

[11] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Revised Papers. Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Argeş, Romania, (2002). *Lecture Notes in Computer Science* **2597**, Springer-Verlag, Berlin, Heidelberg, New York (2003)..

[12] M.J. Pérez-Jiménez, F.J. Romero-Campero: Modelling EGFR Signalling Cascade Using Continuous Membrane Systems. In: G. Plotkin(Ed.): *Proceedings of the Third International Workshop on Computational Methods in Systems Biology 2005 (CMSB 2005)*. University of Edinburgh, Edinburgh, UK (2005).

[13] Nottingham QS Web Page `http://www.nottingham.ac.uk/quorum/`

[14] The P System Web Page `http://psystems.disco.unimib.it/`

[15] SBML Web Pages `http://sbml.org/index.psp`

[16] Scilab Web Pages `http://scilabsoft.inria.fr/`

A An SBML Specification

Consider the P system $\Pi(m)$, with $m = 100$, defined in Section 3.We start by specifying the structure of the system by listing the compartments present in the system and the relationships of inclusion between them.

```
<listOfCompartments>
  <compartment id="e" />
  <compartment id="b" outside="b"/>
</listOfCompartments>
```

There are two different "types" of compartments: compartments labeled by e and compartment labeled by b; all the compartments labeled by b, the bacteria, are included in a compartment with label e, the environment. Specifically, this is just a shorthand for a membrane structure consisting of a number of membranes, each one associated with a compartment labeled by b, contained inside an unique main membrane associated with a compartment labeled by e. The actual number of bacteria in the system is specified as a parameter of the system together with the constants k_i, $1 \leq i \leq 14$.

```
<listOfParameters>
   <parameter id="k1" value="2''constant="true"/>
   <parameter id="k2" value="2" constant="ture"/>
   <parameter id="k3" value="9" constant="true"/>
   <parameter id="k4" value="1" constant="true"/>
   <parameter id="k5" value="10" constant="true"/>
   <parameter id="k6" value="2" constant="true"/>
   <parameter id="k7" value="250" constant="true"/>
   <parameter id="k8" value="200" constant="true"/>
   <parameter id="k9" value="1"  constant="true"/>
   <parameter id="k10" value="50" constant="true"/>
   <parameter id="k11" value="30" constant="true"/>
   <parameter id="k12" value="15" constant="true"/>
   <parameter id="k13" value="20" constant="true"/>
   <parameter id="k14" value="20" constant="true"/>
    <parameter id="m" value="100" constant="true"/>
</listOfParameters>
```

Next, we specify the initial distribution of objects inside the system by listing out the species and their initial concentration inside each compartment.

```
<listOfSpecies>
```

```
  <specie id="OHHL_e"
  initialConcentration="0" compartment="e" />
  <specie id="OHHL_b"
  initialConcentration="0" compartment="b" />
  <specie id="LuxR_b"
  initialConcentration="0" compartment="b" />
  <specie id="LuxR_OHHL_b"
  initialConcentration="0" compartment="b" />
  <specie id="Lux_Box_b"
  initialConcentration="1" compartment="b" />
  <specie id="Lux_Box_LuxR_OHHL_b"
  initialConcentration="0" compartment="b" />
</listOfSpecies>
```

The objects that can be contained inside the environment are labeled by e whereas the objects that can appear inside a bacterium are labeled by b.

Finally we specify the rules as a list of SBML reactions. We just report here two of them as an example.

```
  <reaction name="Reaction1" reversible="false">
    <listOfReactants>
     <specieReference specie="Lux_Box_b" />
    </listOfReactants>
    <listOfProducts>
     <specieReference specie="Lux_Box_b" />
     <specieReference specie="OHHL_b" />
    </listOfProducts>
    <kineticLaw>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <ci>k1</ci>
          <ci>Lux_Box_b</ci>
        </apply>
     </math>
    </kineticLaw>
  </reaction>

  <reaction name="Reaction9" reversible="false">
    <listOfReactants>
     <specieReference specie="OHHL_b" />
```

```
    </listOfReactants>
    <listOfProducts>
     <specieReference specie="OHHL_e" />
    </listOfProducts>
    <kineticLaw>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
          <ci>k9</ci>
          <ci>OHHL_b</ci>
        </apply>
      </math>
    </kineticLaw>
  </reaction>
```

The movement of objects is specified by changing the labels of the products according to the labels of the reactants.

P Systems and the Modeling of Biochemical Oscillations

Luca BIANCO, Federico FONTANA, Vincenzo MANCA

University of Verona
Department of Computer Science
15 strada Le Grazie - 37134 Verona, Italy
E-mail: {fontana,bianco,manca}@sci.univr.it

Abstract

In this paper we discuss the role that P systems have in the description of oscillatory biochemical processes once the membrane system evolution depends on the process parameters. This discussion focuses on a specific application example, meanwhile it includes a general definition of oscillation based on which we want to explore the meaning of oscillatory behaviors more deeply. The symbolic-based approach to biochemical processes such as that provided by P systems has recently resulted in insightful model descriptions. For this reason we expect it to turn useful in computational systems biology, whose models must deal with the twofold nature of the cell that is a *continuous* biochemical reactor ruled by *discrete* information contained in the DNA.

1 Introduction

Originally conceived to assess the expressive power of grammars and to classify formal languages [20], rewriting systems more recently have been applied to the analysis of biological structures—for instance, they have demonstrated capability to represent the development of living species such as the growing of some simple organisms [12, 19]. Inspired by these investigations, P systems [16, 17, 15] have been concerned particularly with the dynamic aspect of rewriting and its application to biology and biochemistry [22, 1]. Dynamic rewriting systems have led to alternative representations of several biological phenomena [21] and to exploratory models of known pathological processes [13, 18].

By our side, we have recently presented a symbolic rewriting algorithm [2] in which production rules are given along with *reaction maps*, each one specifying the "strength" of a rule in modifying a population of symbols (denoting concentrations of chemical reactants, individuals, molecules, and so on) in the system. This algorithm has successfully simulated some well-known biochemical models: the Lotka-Volterra population dynamics [24], and the Brusselator model of the BZ chemical reaction [8]. These early results, along with the inherent advantages that rewriting systems offer in terms of modeling flexibility, ask for doing further tests on more elaborate biochemical models such as those presented in this paper.

After a brief description of the algorithm, we show results obtained simulating an extensive model of circadian rhythms in *Drosophila melanogaster* [11], whose clarity and richness of quantitative data allows to make effective comparisons between the numerical solutions of the differential equations found in that model and the solutions coming out from our rewriting system. Although still partial in front of the huge amount of simulations of circadian rhythms that have been carried out through differential equation system models, these results, along with those achieved in the simulation of the aforementioned dynamics, confirm the effectiveness of our approach in modeling elaborate biochemical behaviors such as those emerging in circadian rhythms.

Inspired by the flexibility and power of this model we have started thinking about how to investigate on the meaning of oscillation taken as a phenomenon *per se*. We have in fact discovered that no clear definition of this phenomenon exists. This lack of a definition reflects an inherent difficulty to characterize oscillation in formal (and consequently quantitative) terms, as opposite to periodicity for which a huge amount of theoretical results have been found, the Fourier analysis being of top of everything [9].

Unlike ideal oscillatory systems, biochemical processes never show exact periodic behavior [5]. In the meantime it is crucial to find if, and how, they oscillate. Gaining insight on the ultimate meaning of oscillation may be useful to define formal tools that help giving an answer to these two questions.

2 Algorithm Quick Overview

A detailed description of the algorithm we use to control the evolution of a system has been previously given [2], furthermore a comprehensive formalization of this algorithm in specific P system-based constructs is ongoing.

Here, we briefly recall the concepts that are necessary to set up a representation of the circadian model.

Let a single-membrane system be made of a set $R = \{r_1, \ldots, r_k\}$ of rewriting rules working over strings on an alphabet $\mathcal{A} = \{X, Y, \ldots\}$ containing k symbols:

$$r_1 : \alpha_{r_1} \to \beta_{r_1}\,, \quad r_2 : \alpha_{r_2} \to \beta_{r_2}\,, \quad \ldots \quad r_k : \alpha_{r_k} \to \beta_{r_k}\,, \tag{1}$$

in which α_ρ and β_ρ are strings respectively denoting consumed and produced objects for each rule $\rho \in R$.

Let the *state* of our system be a k-tuple $\langle q(X), q(Y), \ldots\rangle$ containing the number of objects $X, Y, \ldots$ in the system at every temporal step (here we will make every step correspond to a system transition, though in general this is not necessarily true). To every rule we associate a corresponding *reaction map* $F_{r_1}, F_{r_2}, \ldots, F_{r_k}$, i.e., a real function of the state of the system affecting the rule in the way we explain below.

By denoting with $\alpha(i)$ the ith symbol in a string α, with $|\alpha|$ the length of the same string, and with $|\alpha|_X$ the number of occurrences of the symbol X in α, then we define the *reaction weight* $W_r\big(\alpha_r(i)\big)$ for $r : \alpha_r \to \beta_r$ with respect to the symbol $\alpha_r(i)$:

$$W_r\big(\alpha_r(i)\big) = \frac{F_r}{\displaystyle\sum_{\rho\in R\,|\,\alpha_r(i)\in\alpha_\rho} F_\rho} \quad , \quad i = 1, \ldots, |\alpha_r|\,. \tag{2}$$

Note that at the denominator we sum only over the rules containing the symbol $\alpha_r(i)$ in their left part.

If we, at this point, consider that every rule r cannot consume more than the amount of the symbol (called also *reactant*) whose availability in the system is lowest, then for every rule we have to minimize among all reactants—each one taken with its own multiplicity in α_r—participating to the reaction. In this way for every symbol we find the population a rule applies to during a transition of the system:

$$\Lambda_r = \min_{i=1,\ldots,|\alpha_r|} \left\{ W_r\big(\alpha_r(i)\big)\, \frac{q\big(\alpha_r(i)\big)}{\big|\alpha_r\big|_{\alpha_r(i)}} \right\}. \tag{3}$$

In the end for every symbol $X \in \mathcal{A}$ the change in the number of objects due to r is equal to $|\beta_r|_X - |\alpha_r|_X$ times Λ_r:

$$\Delta_r(X) = \Lambda_r \left(|\beta_r|_X - |\alpha_r|_X\right). \tag{4}$$

A detailed explanation of the algorithm structure, in particular the way it works with discrete populations rather than concentrations, and its extension to multiple reaction environments made using membrane systems, is given in [2].

3 Application to Circadian Rhythms

We have applied the algorithm discussed in Section 2 to the simulation of a known model of circadian cycles (or *rhythms*) in *Drosophila melanogaster*, involving the oscillation of the Period (PER) and Timeless (TIM) proteins [4]. Existing in every living organism, circadian rhythms are biochemical cycles evoked by variations in the expression level of specific genes. Such variations give rise to a surprisingly robust biological clock, synchronized with daylight and performing a complete cycle about every 24 hours.

According to this model the genes involved in the process code for PER and TIM proteins, meanwhile their expression is inhibited by the presence of a PER-TIM protein complex, in its own made of PER and TIM. This complex forms in the cytosol under certain conditions, then migrates inside the nucleus where it behaves as a PER and TIM suppressor. Taken together, gene expression and suppression result in a negative feedback network of signal transduction that has been formalized by a non-trivial system made of several nonlinear differential equations [5, 11].

A graphical scheme of the model is depicted in Figure 1. Details of its functioning can be found in [11]. At least it is interesting to note that the formation of the PER-TIM complex is regulated by the degradation induced on mature TIM (denoted as T_2) by light. Though, in our study we do not include the effects of light.

In spite of its complexity, the PER and TIM model results in emergent oscillatory concentrations of the biochemical elements considered. The temporal evolution of such concentrations exhibits clear mutual relationships between concentration onsets and decays. These relationships disclose the causality existing between gene expression and the consequent change in concentration of the transcribed mRNA and, hence, of the coded proteins.

Symbolic rewriting allows to describe this model by means of a set of rules, avoiding the classical approach based on differential equations. By looking at Figure 1 it is not difficult to figure out the following rewriting

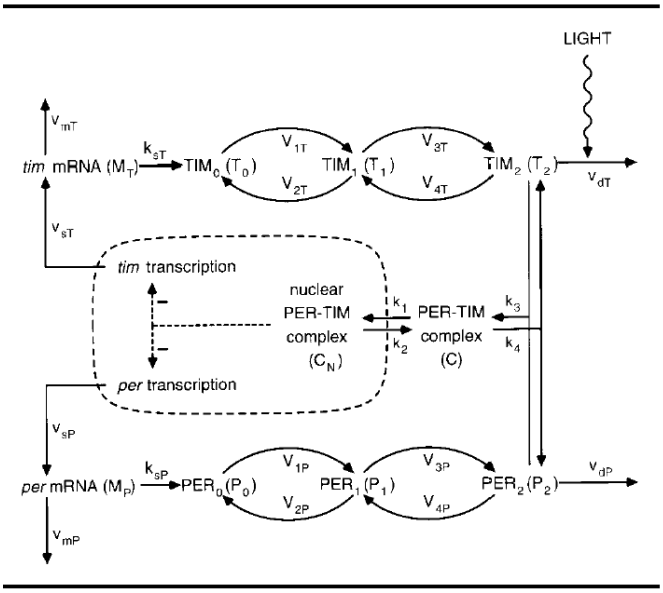


Figure 1: Model for circadian rhythms in *Drosophila* (from Leloup and Goldbeter [11]).

rules:

$$
\begin{array}{llcllcl}
r1: & \lambda & \rightarrow & M_P, & r2: & \lambda & \rightarrow & M_T, \\
r3: & M_P & \rightarrow & \lambda, & r4: & M_T & \rightarrow & \lambda, \\
r5: & M_P & \rightarrow & M_P P_0, & r6: & M_P & \rightarrow & M_T T_0, \\
r7: & P_0 & \rightarrow & P_1, & r8: & T_0 & \rightarrow & T_1, \\
r9: & P_1 & \rightarrow & P_0, & r10: & T_1 & \rightarrow & T_0, \\
r11: & P_0 & \rightarrow & \lambda, & r12: & T_0 & \rightarrow & \lambda, \\
r13: & P_1 & \rightarrow & \lambda, & r14: & T_1 & \rightarrow & \lambda, \\
r15: & P_1 & \rightarrow & P_2, & r16: & T_1 & \rightarrow & T_2, \\
r17: & P_2 & \rightarrow & P_1, & r18: & T_2 & \rightarrow & T_1, \\
r19: & P_2 & \rightarrow & \lambda, & r20: & T_2 & \rightarrow & \lambda, \\
r21: & P_2 T_2 & \rightarrow & C, & r22: & C & \rightarrow & P_2 T_2, \\
r23: & C & \rightarrow & C_N, & r24: & C_N & \rightarrow & C, \\
r25: & C & \rightarrow & \lambda, & r26: & C_N & \rightarrow & \lambda.
\end{array}
\tag{5}
$$

In these rules the symbol λ as usual represents the null string. In this way rules in the form $\lambda \rightarrow X$ are production rules, conversely rules in the form $X \rightarrow \lambda$ are degradation rules.

Furthermore, for each element X that is present in the system we introduce a *transparent rule* in the form $X \to X$. These rules do not cause any change in the system. Rather, they are needed to model elements that do not take part in a reaction (for example, reactants that are spatially far from each other) [2].

Note that, besides the radical differences existing between the differential and the rewriting system, our model differs from the continuous one especially for what concerns the formation of the PER-TIM complex—expressed in our system by rule $r21$. This rule is, in fact, cooperative and in this case we use the *limiter* Λ_{r21}, discussed in Section 2, in order to calculate the variation of P_2, T_2, and C.

As we have previously seen each rule is coupled with a reaction map. According to the formulas proposed in the original model, we have come up with the following maps:

$$
\begin{aligned}
&F_{r1} = v_{sP}\frac{K_{IP}^n}{K_{1P}^n + C_N^n}, && F_{r2} = v_{sT}\frac{K_{IT}^n}{K_{1T}^n + C_N^n},\\
&F_{r3} = \frac{v_{mP}}{K_{mP} + M_P} + K_d, && F_{r4} = \frac{v_{mT}}{K_{mT} + M_T} + K_d,\\
&F_{r5} = K_{sP}, && F_{r6} = K_{sT},\\
&F_{r7} = \frac{v_{1P}}{K_{1P} + P_0}, && F_{r8} = \frac{v_{1T}}{K_{1T} + T_0},\\
&F_{r9} = \frac{v_{2P}}{K_{2P} + P_1}, && F_{r10} = \frac{v_{2T}}{K_{2T} + T_1},\\
&F_{r11} = K_d, && F_{r12} = K_d,\\
&F_{r13} = K_d, && F_{r14} = K_d,\\
&F_{r15} = \frac{v_{3P}}{K_{3P} + P_1}, && F_{r16} = \frac{v_{3T}}{K_{3T} + T_1},\\
&F_{r17} = \frac{v_{4P}}{K_{4P} + P_2}, && F_{r18} = \frac{v_{4T}}{K_{4T} + T_2},\\
&F_{r19} = K_d + \frac{v_{dT}}{K_{dT} + T_2}, && F_{r20} = K_d + \frac{v_{dP}}{K_{dP} + P_2},\\
&F_{r21} = K_3, && F_{r22} = K_4,\\
&F_{r23} = K_1, && F_{r24} = K_2,\\
&F_{r25} = K_{dC}, && F_{r26} = K_{dN}.
\end{aligned}
\tag{6}
$$

Moreover, in agreement with [11], we choose the following set of parameters (reported here dimensionless): $v_{sP} = v_{sT} = 1$, $v_{mP} = v_{mT} = 0.7$, $K_{mP} = K_{mT} = 0.2$, $K_{sP} = K_{sT} = 0.9$, $v_{dP} = v_{dT} = 2$, $K_1 = 0.6$, $K_2 = 0.2$, $K_3 = 0.5$, $K_4 = 0.2$, $K_{IP} = K_{IT} = 1$, $K_{dP} = K_{dT} = 0.2$, $n = 4$ $K_{1P} = K_{1T} = K_{2P} = K_{2T} = K_{3P} = K_{3T} = K_{4P} = K_{4T} = 2$, $K_d = K_{dC} = K_{dN} = 0.01$, $v_{1P} = v_{1T} = v_{1P} = v_{1T} = 8$, $v_{2P} = v_{2T} = v_{4P} =$

$v_{4T} = 1$. Note that the different interpretation given by $r21$ to the formation of the PER-TIM compound, compared to that formalized by a numerical equation (typically as the product of two reactants weighted by a proper kinetic constant rate), suggested to employ different values for the variables K_3 and K_4 as opposite to the values chosen in the continuous model, respectively set to 1.2 and 0.6. In addition to that we have coupled a constant reaction map $F_r = 1$ to every transparent rule r.

Figure 2 (above) shows the salient result we have obtained by simulating circadian rhythms using the membrane model. Plots figure out the state along 130 transitions of the system, i.e., every plot describes the evolution along discrete time of the corresponding element in the k-tuple forming the state. It can be seen that a stable oscillatory dynamics is achieved using the symbolic approach.

Such plots are compared to the numerical solution of the corresponding differential equation model, reported in Figure 2 (below). It can be noted that the relative temporal shifts between concentration peaks are preserved by our simulation. This means that comparable dynamic behaviors exist for the two models. In particular, the membrane system correctly models the sequence of concentration peaks of the phosphorilating PER protein (P_0, P_1 and P_2), followed by the peak in the concentration of the cytosolic PER-TIM complex C and, finally, by its nuclear counterpart C_N. This dynamic behavior corresponds to results obtained by Leloup and Goldbeter, which, in their turn, match with experimental observations [25].

4 Toward a Characterization of Oscillations

For what we have seen in the previous example, oscillation is perhaps the most important emergent property featured by a biochemical system. Investigating its onsets, temporal extension, robustness against parameter changes, characteristic evolution along time, deviation from an ideal periodic track, is crucial for extracting many properties inherently present in the system structure. These properties range from the topology of the signal transduction network underlying the communication flows that are active in the system, to its distinctive parameters which determine the modalities by which this evolution develops along time.

Curiously, oscillation is not yet well defined. On one hand this depends on the generality of the phenomenon. Oscillation in fact includes concepts such as quasi-periodicity, recurrence, periodic chaotic attraction [7]. On the other hand it is precisely that generality that most biochemical systems

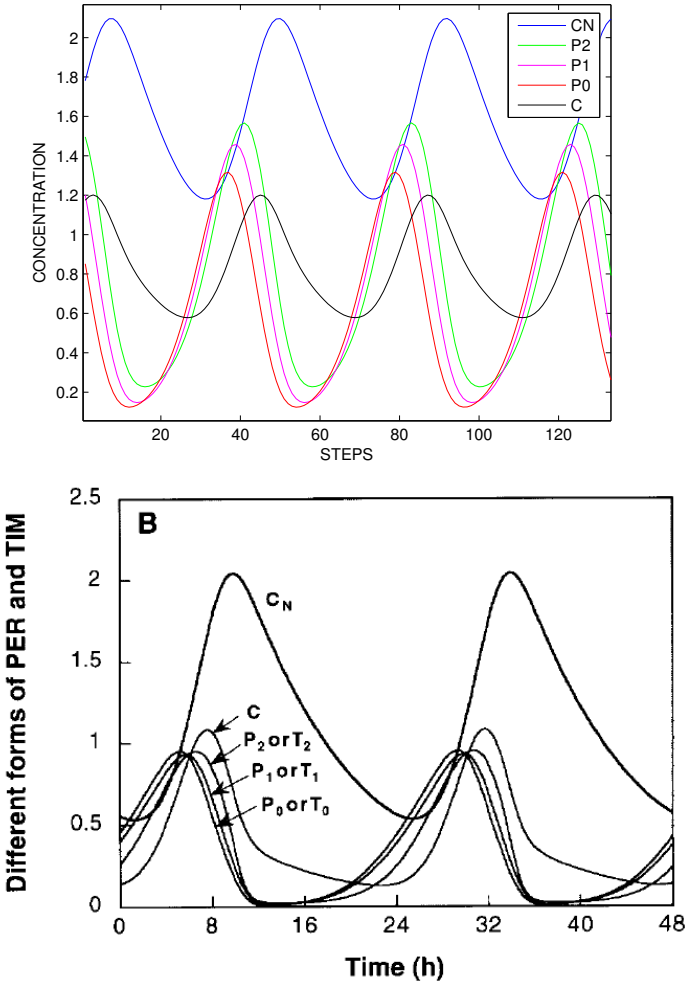


Figure 2: Above: plots for C_N, P_2, P_1, P_0, and C obtained using the metabolic algorithm (elements ordered starting from the highest to the lowest maximum peak value, as in the legend at the top-right corner). Below: plots for C_N, P_2, P_1, P_0, and C (from Leloup and Goldbeter [11]).

exhibit: in some sense, oscillation is a weak but, at the same time, one of the strongest and most distinctive properties shown by nonlinear systems. It is with these questions in mind that we try to formalize oscillation.

Let us consider a state transition dynamics $\mathcal{S} = (S, q)$, in which q maps states into sets of states: $q : S \longrightarrow \mathcal{P}(S)$ [13, 14]. In $\mathcal{S}$, let us consider a

local trajectory T made of states $X_0, X_1, \ldots, X_n$ such that X_i is obtained by repeatedly applying m_i times the transition function to X_{i-1} for each $i = 1, \ldots, m$: (note that we conveniently extend q to work over sets, i.e., $q(X) = \bigcup_{s \in X} q(s)$)

$$T: \quad X_0 \overset{q^{(m_1)}}{\longrightarrow} X_1 \overset{q^{(m_2)}}{\longrightarrow} \cdots \overset{q^{(m_i)}}{\longrightarrow} X_i \overset{q^{(m_{i+1})}}{\longrightarrow} \cdots \overset{q^{(m_{n-1})}}{\longrightarrow} X_{n-1} \overset{q^{(m_n)}}{\longrightarrow} X_n\,, \quad (7)$$

with

$$X_i \subseteq q^{(m_i)}(X_{i-1}) = \underbrace{\big(q \circ q \circ \cdots \circ q\big)}_{m_i \text{ times}}(X_{i-1}), \quad i = 1, 2, \ldots \quad (8)$$

Definition 4.1. *A local trajectory T in $\mathcal{S}$ oscillates around x_0 with respect to a (state observation) function $\mu : S \longrightarrow \mathbb{R}$ if T exists such that*

- $\mu(X_i) \geq x_0$ for i even (odd)
- $\mu(X_i) < x_0$ for i odd (even).

Clearly, this definition does not prevent from several oscillations to exist in one single sequence. In particular it does not exclude that inner oscillations are present in between adjacent states in T. For instance, it may be likely that T oscillates also around x_1, and that this oscillation appears within states traced by the local trajectory between X_k and X_{k-1}. And so on.

Whether this definition can form an initial basis to a future theoretical development, enriching the wide amount of literature already existing on Fourier and, more in general, spectral analysis, will be a matter of forthcoming research. We are now working on this definition in an attempt to find a more insightful interpretation of the oscillation *per se*.

5 Concluding Remarks

Our experience with the representation of several biochemical phenomena, including the circadian model we have presented here, suggests that membrane systems are promising candidates for providing accurate models of such phenomena provided their versatility in dealing with discrete (that is, symbolic) representations of the information and its transmission along peculiar communication channels such as cell ports and signal transduction networks.

This first attempt of symbolic modeling of the circadian cycle in *Drosophila* yet has not considered the effect of light on the degradation of the

phosphorilated TIM. We want to include this effect in a forthcoming session of further tests of our algorithm, still relying on the well-documented figures proposed in [11]. Even more interesting will be comparing our symbolic algorithm to some well-known stochastic simulation methods that are used when the molecules involved in a biochemical process are few, in a way that the deterministic approach turns out to be no longer suitable. Surprising analogies exist in fact between the symbolic and the stochastic approach to the simulation of circadian rhythms when our algorithm is set to work over populations rather than concentrations, i.e., over discrete rather than continuous domains [6, 3].

In parallel, an analysis focusing on the ultimate meaning of the behaviors we observe in a dynamical system is needed since, if successful, this analysis will become a useful way to extract important structural information from a system, even independently of its physical (*viz.* biological for us) nature: such kind of analyses have already provided powerful conceptual frameworks based on control theory [23] that have found fertile applications, for instance, in the identification of "black-box" systems—as biological systems still are, at least to some extent [10].

A possible roadmap to follow along this research perhaps starts from properly defining basic dynamic concepts, oscillation *in primis* for its major importance in any dynamic phenomenon. We will try to move along this roadmap in the next few months.

References

[1] L. Bianco, F. Fontana, G. Franco, V. Manca: P systems for biological dynamics. In: G. Ciobanu, G. Păun, M. J. Pérez-Jiménez (Eds.): *Applications of Membrane Computing*. Springer (2005), 81–126.

[2] L. Bianco, F. Fontana, V. Manca: Metabolic algorithm with time-varying reaction maps. *Proc. of the Third Brainstorming Week on Membrane Computing (BWMC 2005)*. Sevilla, Spain (2005).

[3] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. of Phys. Chem.* **81** (25) (1977), 2340–2361.

[4] A. Goldbeter: Computational approaches to cellular rhythms. *Nature* **420** (2002), 238–244.

[5] A. Goldbeter: Biochemical Oscillations and Cellular Rhythms. The molecular bases of periodic and chaotic behaviour. *Cambridge University Press*, New York (2004).

[6] D. Gonze, J. Halloy, A. Goldbeter: Stochastic model for circadian oscillations: Emergence of a biological rhythm. *Int. J. of Quantum Chemistry* **98** (2) (2004), 228–238.

[7] R. C. Hilborn: *Chaos and Nonlinear Dynamics. Oxford University Press*, Oxford, UK (2000).

[8] D. S. Jones and B. D. Sleeman: *Differential equations and mathematical biology.* Chapman & Hall/CRC, London, UK (2003).

[9] T. Kailath: *Linear Systems.* Prentice-Hall, Englewood Cliffs (1980).

[10] H. Kitano: Computational systems biology. *Nature* **420** (2002), 206–210.

[11] J.-C. Leloup, A. Goldbeter: A model for circadian rhythms in *Drosophila* incorporating the formation of a complex between the PER and TIM proteins. *J. of Biological Rhythms* **13** (1) (1998), 70–87.

[12] A. Lindenmayer: Mathematical models for cellular interaction in development. *J. of Theoretical Biology* **18,** Part I and II (1968), 280–315.

[13] V. Manca, L. Bianco, F. Fontana: Evolutions and oscillations of P systems: Applications to biological phenomena. In: G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa (Eds.): Membrane Computing, 5th International Workshop, WMC 2004. *Lecture Notes in Computer Science* **3365**, Springer (2005), 63–84.

[14] V. Manca, G. Franco, and G. Scollo: State transition dynamics: Basic concepts and molecular computing perspectives. In: M. Gheorghe (Ed.): *Molecular Computational Models - Unconventional Approaches,* chapter 2. Idea Group (2004).

[15] G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, 5th International Workshop, WMC 2004, Milan, Italy, June 14-16, 2004, Revised Selected and Invited Papers. *Lecture Notes in Computer Science* **3365**, Springer (2005).

[16] G. Păun: Computing with membranes. *J. Comput. System Sci.* **61** (1) (2000), 108–143.

[17] G. Păun: *Membrane Computing: An Introduction*. Springer, Berlin, (2002).

[18] M. J. Pérez-Jiménez, F. J. Romero-Campero: Modelling EGFR signalling network using continuous membrane systems. Submitted (2005).

[19] P. Prusinkiewicz, M. Hammel, J. Hanan, R. Mech: Visual models of plant development. In: G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages,* volume III: Beyond Words. Springer-Verlag, Berlin (1997), 535–597.

[20] G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*. Springer-Verlag, Berlin, Germany (1997).

[21] I. Stamatopoulou, M. Gheorghe, P. Kefalas: Modelling dynamic organization of biology-inspired multi-agent systems with communicating X-machines and population P systems. In: G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing, 5th International Workshop, WMC 2004, *Lecture Notes in Computer Science* **3365**, Springer (2005), 389–403.

[22] Y. Suzuki and H. Tanaka: Chemical oscillation in symbolic chemical systems and its behavioral pattern. In: Y. Bar-Yam (Ed.): *Proc. International Conference on Complex Systems*. Nashua, NH, September (1997).

[23] T. L. Vincent, W. J. Grantham: *Nonlinear and Optimal Control Systems*. Wiley, New York (1997).

[24] V. Volterra: Fluctuations in the abundance of a species considered mathematically. *Nature* **118** (1926) 558–560.

[25] H. Zeng: Constitutive overexpression of the drosophila period protein inhibits period mRNA cycling. *The EMBO Journal* **13** (1994), 3590–3598.

Encoding-Decoding Classes of P Systems for the Metabolic Algorithm

Luca BIANCO, Vincenzo MANCA

University of Verona
Department of Computer Science
strada Le Grazie, 15
37134 Verona, Italy
E-mail: bianco@sci.univr.it,
vincenzo.manca@univr.it

1 Introduction

A fundamental aspect in the investigation of computation systems, and especially of P systems, is the use of many translation methods in order to pass from a certain kind of systems into another one. In this way it is possible to perform all computations in the translated system, rather than in the original one. For example, many computational universality and equivalence results on P systems [6, 5, 8] are based on such a technique.

The aim of this extended abstract is to define a notion of *computational encoding*, which allows us to extend to complex membrane structures the *metabolic algorithm* [2, 4, 3] that was developed for basic membrane systems and has proved to be very useful in the simulation of many biological phenomena. In this preliminary work, we give fundamental definitions and the main framework of this approach.

Our notion of *computational encoding* resembles, in some aspects, the notion of bisimulation developed in concurrency theory [7]. However, computational encoding does not intend to cope with the operational semantics of systems or processes "at the same level", but rather it deals with the reduction of a computation from a "machine" to another one at a different (simpler or more complex) descriptional level. In fact, this notion of encoding is related to the interpretation of a computational system into another

one. This aspect is apparent in Figure 1, if we compare the upper computation with the lower one, where in general one step at the upper level corresponds to many steps at the lower level. The ratio between the computation lengths is a parameter related to the different descriptional levels of the two corresponding transitional systems.

2 Transitional Systems

Let us start introducing formally the notion of *Transitional system*:

Definition 2.1 *A* transitional system *is a 7-tuple* $\Pi = (A, C, R, \sigma, \Rightarrow, G, F)$ *where*

(i) *A is the alphabet;*

(ii) *C is a set of strings defined over A, representing configurations of the system;*

(iii) *R is a set of rewriting rules, defined over C;*

(iv) *σ is the* transitional *or* program function*, where $\forall \mu \in C$, $\sigma(\mu)$ is a set of sets of rules, that is, every element $Q \in \sigma(\mu)$ contains the rules that are simultaneously applicable to the configuration $\mu \in C$;*

(v) *$\Rightarrow$ is the* transition*, a ternary relation $C \times \wp(R) \times C$. Given two configurations $\mu, \mu' \in C$ and a subset of rules $Q \in \sigma(\mu)$, we denote by $\mu \Rightarrow_Q \mu'$ the transition from μ to μ' by means of the application of the set of rules Q;*

(vi) *G is a set of sequences of transitions and its elements are called* computations*. Given an initial configuration μ_1, a computation $\Gamma \in G$ is denoted by:*

$$\Gamma = \mu_1 \Rightarrow_{Q_1} \mu_2 \Rightarrow_{Q_2} \cdots \Rightarrow_{Q_{n-1}} \mu_n$$

where for each i:

(a) $\mu_i \in C$,
(b) $\sigma(\mu_i) = Q_i$,
(c) $\mu_i \Rightarrow_{Q_i} \mu_{i+1}$
(d) $\mu_n \in F$.

(vii) *$F \subseteq C$ is the set of final configurations;*

Note that if the set of initial configurations is fully specified, then G is completely defined by the other elements of the transitional system. As it is apparent from the notation, the computation $\Gamma = \mu_1 \Rightarrow_{Q_1} \mu_2 \Rightarrow_{Q_2} \cdots \Rightarrow_{Q_{n-1}} \mu_n$ is basically a sequence of transitions between configurations, originated from the initial configuration μ_1 by means of the application of the rules of the system.

It is useful to denote with $\Gamma(i)$ the ith configuration of the computation Γ and with $l_\Gamma = |\Gamma|$ the length of the computation Γ.

In the following, $\mu \Rightarrow_{*Q} \mu'$ is a compact representation meaning the existence of a computation $\mu \Rightarrow_{Q_1} \mu_2 \Rightarrow_{Q_2} \cdots \Rightarrow_{Q_{n-1}} \mu_n$ for some $n \in \mathbb{N}$ where, for every $1 \leq i \leq n$, $Q_i \subseteq Q$.

3 Computational Encodings

We introduce two distinct notions of *computational encodings* between different transitional systems.

Definition 3.1 *A* computational encoding E *from a transitional system* $\Pi = (A, C, R, \sigma, \Rightarrow, G, F)$ *to another transitional system* $\Pi' = (A', C', R', \sigma', \Rightarrow', G', F')$ *is a triple of functions* $E = (\gamma_{conf}, \gamma_{rule}, \gamma_{comp})$ *with:*

- $\gamma_{conf} : C \to C'$ *is an injective function used to encode configurations of* Π *into configurations of* Π'*,*
- $\gamma_{rule} : R \to \wp(R')$ *is an injective function encoding rules of* Π *into a set of rules of* Π'*, where* $\wp(R')$ *is the power set of* R'*,*
- $\gamma_{comp} : G \to G'$ *is an injective function used to encode computations of* Π *into computations of* Π'*.*

in which, for every $\Gamma \in G$*, the following conditions are satisfied:*

(i) $\gamma_{conf}(\Gamma(1)) = \gamma_{comp}(\Gamma)(1)$

(ii) $\gamma_{conf}(\Gamma(l_\Gamma)) = \gamma_{comp}(\Gamma)(l_\Gamma)$

Given a computational encoding E that encodes a transitional system Π into another one Π', we write $\Pi' = E(\Pi)$.

It is interesting to point out that, due to the injectivity of γ_{conf}, a *mirror principle* holds. In fact, starting from an initial configuration μ of Π, we can

encode it in $\mu' = \gamma_{conf}(\mu)$ of $E(\Pi)$ and in this encoded transitional system we can execute the computation Γ' until we reach its last configuration $\Gamma'(l_{\Gamma'})$. After this, we can obtain the final configuration $\Gamma(l_\Gamma) = \gamma^{-1}_{conf}(\Gamma'(l_{\Gamma'}))$ of the computation Γ in the transitional system Π. The mirror principle becomes interesting when we can encode a transitional system into another one, that is more efficient, according to some computational perspective.

Let us introduce a more strict notion of encoding, in which we require a step-by-step correspondence between configurations of a transitional system Π and their corresponding configurations in $E(\Pi)$:

Definition 3.2 *A computational encoding $E = (\gamma_{comp}, \gamma_{conf}, \gamma_{rule})$ from a transitional system $\Pi = (A, C, R, \sigma, \Rightarrow, G, F)$ to another one $\Pi' = (A', C', R', \sigma', \Rightarrow', G', F')$ is* strict *(or 1-1 step) if the step commutativity holds for every $\Gamma \in G$:*

(i) $\gamma_{comp}(\Gamma)(i) = \gamma_{conf}(\Gamma(i)),\ \forall\ i, 1 \leq i \leq l_\Gamma$

(ii) $\Gamma(i) \Rightarrow_Q \Gamma(i+1) \Leftrightarrow \gamma_{conf}(\Gamma(i)) \Rightarrow'_{\gamma_{rule}(Q)} \gamma_{conf}(\Gamma(i+1))$

Note that, when a computational encoding E is **strict**, γ_{comp} is **completely determined** by the couple $(\gamma_{conf}, \gamma_{rule})$.

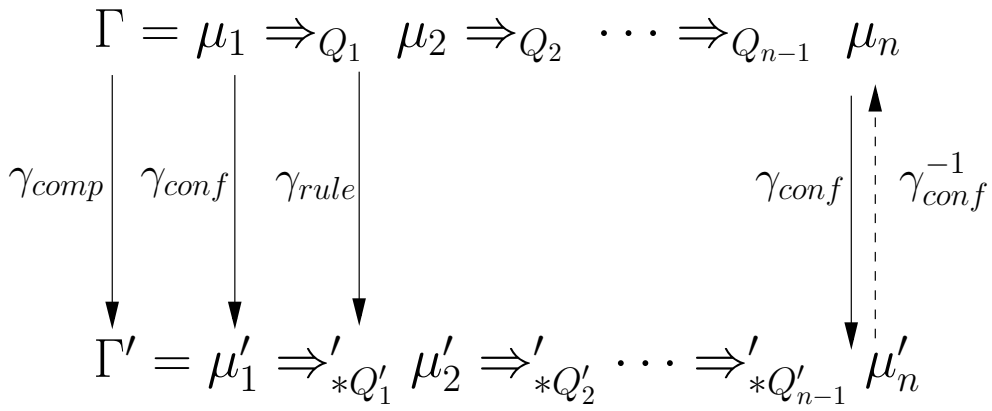


Figure 1: Schematic representation of a computational encoding.

Figure 1 depicts a schematic representation of a computational encoding. The three encoding functions $(\gamma_{conf}, \gamma_{rule}, \gamma_{comp})$ are represented as arrows which connect elements of the computation Γ to the corresponding ones in computation Γ'. In the case of strict encoding the relationship between elements of Γ and the corresponding ones in Γ' can be extended to all elements

of computations. This means that in the previous picture we have to remove the symbols * from rules and add arrows going from all up configurations and rules to the corresponding elements on the lower path.

4 Encoding n-PBR Systems into 0-PBR Systems

The definitions of transitional systems and of computational encoding, expressed in previous sections, allow us to use the general schema of Figure 2 to compare the computational dynamics of different systems. In fact, starting from two distinct systems S_1 and S_2 of different nature, we can represent them in terms of transitional systems (respectively Π_1 and Π_2) and then compare their dynamics in the common and homogeneous environment of the transitional systems.

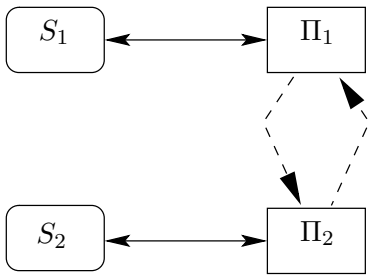


Figure 2: General framework for systems encodings.

Now we apply a similar schema (refer to Figure 3) in order to determinate the dynamics of an n-PBR system by means of a 0-PBR system. We have defined the metabolic algorithm [2, 4, 3], only in the case of 0-PBR systems, as a method to compute the dynamics of many interesting biological phenomena. Now we can extend its applicabily to the case of n-PBR systems (i.e. PBR systems with $n > 0$ membranes) by using a strict computational encoding.

PBR Systems [4, 3] are an extension and generalization of PB systems [1]. They introduce reaction maps needed to describe their time-varying dynamics and generalize PB Systems rules in such a way to obtain forms allowing us to perform an easy translation from multi-membrane (n-PBR) to zero-membrane (0-PBR) systems.

A n-PBR system (i.e., a PBR system with n membranes) is

$$\Pi = (\mathcal{A}, \mu_0, R, F, E) \tag{1}$$

where:

- $\mathcal{A}$ is the alphabet of *symbols*;
- μ_0, is the initial *configuration*, a string in which alphabet symbols contained in n nested parentheses, labelled $0, \ldots, n-1$, denote the objects contained in corresponding membranes. For instance, the configuration $[_1 a [_2 bc]_2 [_3 d]_3]_1$, where $a, b, c, d \in \mathcal{A}$, says a belongs to membrane 1, bc to membrane 2, and d to membrane 3, moreover that membrane 1 contains membranes 2 and 3.
- R is a finite set of *rules* of the following three possible forms, with $\alpha, \beta, \delta, \gamma \in \mathcal{A}^*$:

  (a) $[_h [_j \alpha [_i \beta \rightarrow [_h [_j \delta [_i \gamma$ with $1 \leq j, i \leq n-1$ and $0 \leq h \leq n-1$, telling that α is transformed into δ in membrane j and β is transformed into γ in membrane i, moreover that membrane i is contained into membrane j and they are both placed inside membrane h;

  (b) $[_0 \alpha [_i \beta \rightarrow [_0 \delta [_i \gamma$ with $1 \leq i \leq n-1$, telling that α is transformed into δ in membrane 0 (e.g., the skin membrane) and β is transformed into γ in membrane i, moreover that membrane i is contained in the skin membrane;

  (c) $[_0 \beta \rightarrow [_0 \alpha\beta$, telling that α is created inside the skin membrane in presence of β.

  Note that $[_i \alpha [_j \beta$ means that α is a substring of the string representing a multiset of objects contained in membrane i, which in turn contains membrane j comprising β.
- F is a finite set of functions called *reaction maps*, each associated to a rule in a one-to-one manner;
- E is the *environment*, a set of rules of the type (c).

Note that when $n = 0$ there are no membranes in the system. In this case a configuration is simply a string over the alphabet $\mathcal{A}$ (i.e., $\mu = \gamma$), rules of type (a) and (b) have the form $\alpha \rightarrow \beta$, while rules of type (c) have the form $\beta \rightarrow \alpha\beta$, with $\alpha, \beta, \gamma \in \mathcal{A}^*$.

We refer the reader to [4, 3] for more details on reaction maps, their relationship with rules and for an accurate description on how these elements are used by the metabolic algorithm.

The encoding strategy from n-PBR into 0-PBR systems is made of two parts: the former managing configurations, the latter dealing with rules.

The following set of rewriting rules defines an encoding γ_{conf} of n-PBR configurations into 0-PBR configurations:

$$\begin{array}{lcl} [_i]_i & \rightarrow & \lambda \\ [_i X & \rightarrow & X_i[_i \\ [_j X_i & \rightarrow & X_{j,i}[_j \\ [_h X_{j,i} & \rightarrow & X_{j,i}[_h \\ X_0 & \rightarrow & X_{0,0} \end{array} \tag{2}$$

in which $X \in \mathcal{A}$, $0 \leq i, j, h \leq n-1$ with $i \neq j \neq h$. When these rules are applied to configurations of the n-PBR system they provide configurations of the 0-PBR system. The idea behind this encoding is to get rid of membranes, by indexing objects with the identifiers of membranes containing them. To keep track of the whole membrane structure it is sufficient to mark every object with the label j of the membrane containing it in combination with the label i of the immediately outer membrane. We encode an object X within the skin membrane (that is conventionally labeled with 0) as $X_{0,0}$.

Before proceeding any further, we see an encoding example for the following configuration of a 3 membrane system:

$$[_0\ A\ [_1\ BC]_1\ [_2\ A\ ]_2\ B\ ]_0$$

that, according to rules (2), applied in a maximal parallel way, originate the following sequence of strings:

$$\begin{array}{lcl} & [_0\ A\ [_1\ BC\ ]_1 [_2\ A\ ]_2\ B\ ]_0 & \rightarrow \quad A_0\ [_0\ B_1\ [_1\ C\ ]_1\ A_2\ [_2]_2\ B\ ]_0 \\ \rightarrow & A_{0,0}\ [_0\ B_1 C_1\ [_1]_1\ A_2 B\ ]_0 & \rightarrow \quad A_{0,0}\ B_{0,1}\ [_0\ C_1 A_2 B\ ]_0 \\ \rightarrow & A_{0,0} B_{0,1} C_{0,1} [_0\ A_2 B\ ]_0 & \rightarrow \quad A_{0,0} B_{0,1} C_{0,1} A_{0,2}\ [_0\ B\ ]_0 \\ \rightarrow & A_{0,0} B_{0,1} C_{0,1} A_{0,2} B_0\ [_0]_0 & \rightarrow \quad A_{0,0} B_{0,1} C_{0,1} A_{0,2} B_{0,0} \end{array} \tag{3}$$

where only the first and the last strings represent admissible configurations, respectively for a 3-PBR system and for a 0-PBR system.

The second part of the encoding deals with metarules which establish how to transform n-PBR rules into 0-PBR rules. As defined in (1), an n-PBR System has three types of rules. So, in the 0-PBR system:

(i) rules of type (a) must be substituted by rules in the form $\alpha_{h,j}\beta_{j,i} \rightarrow \delta_{h,j}\gamma_{j,i}$;

(ii) rules of type (b) must be substituted by rules in the form $\alpha_{0,0}\beta_{0,i} \rightarrow \delta_{0,0}\gamma_{0,i}$;

(iii) rules of type (c) must be substituted with rules in the form $\beta_{0,0} \rightarrow \alpha_{0,0}\beta_{0,0}$.

To summarize, the encoding of an n-PBR into a 0-PBR System changes the configuration and the rules by removing all parentheses (that represent membranes). Localization is now encoded into symbols, this making the alphabet different: every symbol in the 0-PBR system is indexed with the two innermost membranes containing it (in the n-PBR system). Obviously, the alphabet $\mathcal{A}_1$ of a 0-PBR system that has been derived by an n-PBR system can have larger cardinality than $\mathcal{A}$.

It is important to notice that in this case the computational encoding $E = (\gamma_{conf}, \gamma_{rule}, \gamma_{comp})$ is strict, for this reason γ_{comp} is fully specified once we give γ_{conf} and γ_{rule}. Therefore from the mirror principle we can calculate the dynamics of an n-PBR system by means of a 0-PBR system, and this construction can be used to extend the applicability of the metabolic algorithm to the case of n-PBR systems. The following picture illustrates the underlying schema of the method.

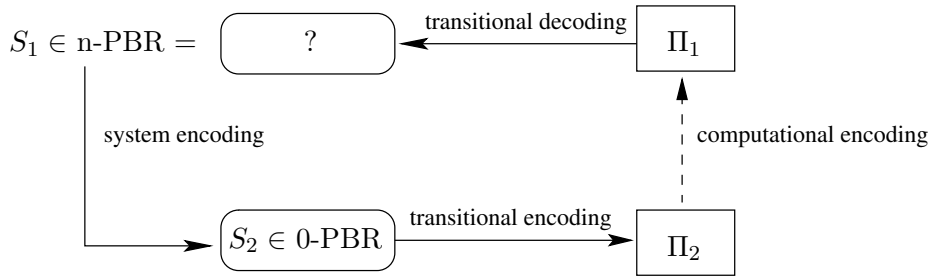


Figure 3: Simulation of a n-PBR system by means of a 0-PBR system.

References

[1] F. Bernardini, V. Manca: Dynamical aspects of P systems. *BioSystems* **70** (2002), 85–93.

[2] L. Bianco, F. Fontana, G. Franco, V. Manca: P systems for biological dynamics. In [6].

[3] L. Bianco, F. Fontana, V. Manca: P systems with reaction maps. *International J. of Fundations of Computer Science*,To appear (2005).

[4] L. Bianco, F. Fontana, V. Manca: Reaction-driven Membrane Systems. *ICNC'05*, Changsha, China (2005).

[5] Cristian S. Calude, Gheorghe Păun: Bio-steps beyond Turing. *BioSystems* **77** (1-3) (2004), 175–194.

[6] G. Ciobanu, G. Păun, M. J. Pérez-Jiménez (Eds.): *Applications of Membrane Computing*. Springer, Berlin (2005). To appear.

[7] R. Milner: *Communicating and Mobile Systems: the π-calculus*. Prentice-Hall (1999).

[8] G. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, Germany (2002).

On the Computational Power of the Mate/Bud/Drip Brane Calculus: Interleaving vs. Maximal Parallelism

Nadia BUSI

Dipartimento di Scienze dell'Informazione
Università di Bologna
Mura A. Zamboni 7, I-40127 Bologna, Italy.
E-mail: busi@cs.unibo.it

Abstract

Brane calculi are a family of biologically inspired process calculi proposed in [3] for modeling the interactions of dynamically nested membranes. In [3] two basic calculi are proposed. Mate/Bud/Drip (MBD) is one of such basic calculi, and its primitives are inspired by membrane fusion and fission.

In this paper we investigate the expressiveness of MBD w.r.t. its ability to act as a computational device. In particular, we compare the expressiveness of two different semantics for MBD: the standard interleaving semantics – where a single interaction is executed at each computational step – and the maximal parallelism semantics – according to which a computational step is composed of a maximal set of independent interactions.

For the interleaving semantics, we show a nondeterministic encoding of Register Machines in MBD, that preserves the existence of a terminating computation, but that could introduce additional divergent (i.e., infinite) computations.

For the maximal parallelism semantics, we provide a deterministic encoding of Register Machines, which preserves both the existence of a terminating computation and the existence of a divergent computation.

The impossibility of providing a deterministic encoding under the interleaving semantics is a consequence of the decidability of the existence of a divergent computation proved in [1].

1 Introduction

Brane calculi [3] are a family of process calculi proposed for modeling the behaviour of biological membranes. In a process algebraic setting, brane calculi represent an evolution of BioAmbients [10], a variant of Mobile Ambients [4] based on a set of biologically inspired primitives of interaction. The main novelty of brane calculi consists in the fact that the active entities reside on membranes, and not inside membranes.

However, the formal investigation of biological membranes has been initiated by G. Păun with membrane computing [8], in the field of automata and formal language theory. Quoting from [5], the objectives of brane calculi and membrane computing [9] are different: "While membrane computing is a branch of natural computing, which tries to abstract computing models, in the Turing sense, from the structure and the functioning of the cell, making use especially of automata, languages, and complexity theoretic tools, brane calculi pay more attention to the fidelity to the biological reality, have as primary target systems biology, and use especially the framework of process algebra." Another difference is concerned with the semantics of the two formalism: whereas brane calculi are usually equipped with an interleaving, sequential semantics (each computational step consists of the execution of a single instruction), the usual semantics in membrane computing[1] is based on maximal parallelism (a computational step is composed of a maximal set of independent interactions).

Despite such differences, some recent papers try to establish some contact point between the two areas. A very preliminary step in this direction is represented by [1], where the computational power of two variants of basic brane calculi is investigated. A more relevant step is [5], where a variant of P systems (the formalism of membrane computing) is defined, inspired by the interaction primitives of the brane calculi, and its computational power is investigated. The present paper goes in the same direction, as it continues the investigation of the computational power of brane calculi started in [1], and investigates an alternative semantics for brane calculi, inspired by the maximal parallelism semantics usually adopted for P systems.

The focus in this paper is on the Mate/Bud/Drip calculus (MBD), a variant of basic brane calculus whose primitives are inspired by membrane fusion (mate) and fission (mito). Because membrane fission can split a membrane at an arbitrary place, it turns out to be a rather uncontrollable process. Hence, it is replaced by two simpler operations: *budding*, that is splitting

[1] With the notable exception of, e.g., [6].

off one internal membrane, and *dripping*, that consists in splitting off zero internal membranes. This paper originates from an open problem raised in [1], where the expressiveness of two basic brane calculi of [3], namely, MBD and PEP (a basic Brane Calculus with interaction primitives inspired by endocytosis and exocytosis) was investigated.

In [1] an encoding of RAMs in PEP is defined. Such an encoding provides a very faithful representation of the behaviour of RAMs. In fact the encoding of RAMs in PEP is deterministic. As RAMs are a deterministic computing device, we have that the RAM can either terminate or diverge, but cannot have both a divergent and a terminated computation. As the encoding has the same property, and the encoding respects the terminating behaviour of the RAM (i.e., the encoding terminates iff the RAM terminates), we obtain the undecidability of both the existential termination (there exists a terminating computation) and the universal termination (all computations terminate) for PEP. In [1] we also prove the decidability of universal termination for MBD, and the decidability of existential termination for MBD was left as an open problem. In this paper we answer to the above question by providing a nondeterministic encoding of RAMs in MBD, which preservers the existence of a terminating computation. The encoding is nondeterministic because it introduces additional computations which do not follow the expected behaviour of the modeled RAM. However, all these computations are infinite. This ensures that, given a RAM, its modeling has a terminating computation if and only if the RAM terminates. A direct consequence of this result is the undecidability of existential termination for MBD.

The decidability of universal termination for MBD in [1] ensures that we cannot do better, namely, it is impossible to provide a deterministic encoding of RAMs in MBD. It is also impossible to provide a (nondeterministic) encoding of RAMs in MBD that preserves the existence of a divergent computation, or satisfying the following property: the RAM terminates iff all the computations of the encoding terminate.

The computational power of MBD is increased if we move to the maximal parallelism semantics typical of Membrane Computing [9]. According to the maximal parallelism semantics, at each computational step a maximal set of independent reductions is simultaneously executed. Hence, all the membranes that can evolve have to do it. By exploiting such maximal progress hypothesis, we provide a deterministic encoding of RAMs in MBD with maximal parallelism that preserves the existence of a terminated computation (hence also the existence of a divergent computation). Thus we obtain the undecidability of both existential and universal termination for

MBD with maximal parallelism. This result confirms the intuition emerging from [6], where the interleaving (sequential) and the maximal parallelism semantics of many variants of P systems are compared: in most cases, the computational power increases when moving from interleaving to maximal parallelism.

The paper is organized as follows: in Section 2 we present the syntax of MBD, and equip MBD with both a standard, interleaving semantics and a maximal parallelism semantics. Section 3 contains the nondeterministic encoding of RAMs in MBD with interleaving semantics, and the deterministic encoding of RAMs in MBD with maximal parallelism semantics. Section 4 reports some conclusive remarks.

2 MBD Calculus: Syntax and Semantics

In this section we recall the syntax and the standard, interleaving semantics of Brane Calculi, and specialize it to MBD [3]. Then, we define an alternative semantics that enforces the execution, at each computational step, of a maximal set of independent operations.

2.1 Syntax and Structural Congruence of Brane Calculi

A system consists of nested membranes, and a process is associated to each membrane.

Definition 1 *The set of systems is defined by the following grammar:*

$$P, Q \quad ::= \quad \diamond \mid P \circ Q \mid !P \mid \sigma(\!| P |\!)$$

The set of membrane processes is defined by the following grammar:

$$\sigma, \tau \quad ::= \quad 0 \mid \sigma|\tau \mid !\sigma \mid a.\sigma$$

Variables a, b range over actions, that will be detailed later.

The term $\diamond$ represents the empty system; the parallel composition operator on systems is $\circ$. The replication operator ! denotes the parallel composition of an unbounded number of instances of a system. The term $\sigma(\!| P |\!)$ denotes the membrane that performs process σ and contains system P.

The term 0 denotes the empty process, whereas | is the parallel composition of processes; with $!\sigma$ we denote the parallel composition of an unbounded number of instances of process σ. Term $a.\sigma$ is a guarded process: after performing the action a, the process behaves as σ.

We adopt the following abbreviations: with a we denote $a.0$, with $(\!|\, P \,|\!)$ we denote $0(\!|\, P \,|\!)$, and with $\sigma(\!|\ |\!)$ we denote $\sigma(\!|\, \diamond \,|\!)$.

The structural congruence relations on systems and processes is defined as follows:[2]

Definition 2 *The structural congruence $\equiv$ is the least congruence relation satisfying the following axioms:*

$$
\begin{array}{ll}
P \circ Q \equiv Q \circ P & \sigma \mid \tau \equiv \tau \mid \sigma \\
P \circ (Q \circ R) \equiv (P \circ Q) \circ R & \sigma \mid (\tau \mid \rho) \equiv (\sigma \mid \tau) \mid \rho \\
P \circ \diamond \equiv P & \sigma \mid 0 \equiv \sigma \\
 & \\
!\diamond \equiv \diamond & !0 \equiv 0 \\
!(P \circ Q) \equiv !P \circ !Q & !(\sigma \mid \tau) \equiv !\sigma \mid !\tau \\
!!P \equiv !P & !!\sigma \equiv !\sigma \\
P \circ !P \equiv !P & \sigma \mid !\sigma \equiv !\sigma \\
 & \\
0(\!|\, \diamond \,|\!) \equiv \diamond &
\end{array}
$$

2.2 Interleaving Semantics of Brane Calculi

We recall the standard, interleaving semantics. At each computational step, a single reaction is chosen and executed. The next definition provides the set of generic reaction rules that are valid for all brane calculi, while the reaction axioms are specific for each brane calculus; the reaction axioms for MBD will be provided in Definition 5.

Definition 3 *The basic reaction rules are the following:*

$$
(\texttt{par}) \quad \frac{P \ \rightarrow \ Q}{P \circ R \ \rightarrow \ Q \circ R} \qquad\qquad (\texttt{brane}) \quad \frac{P \ \rightarrow \ Q}{\sigma(\!|\, P \,|\!) \ \rightarrow \ \sigma(\!|\, Q \,|\!)}
$$

$$
(\texttt{strucong}) \quad \frac{P' \equiv P \qquad P \ \rightarrow \ Q \qquad Q \equiv Q'}{P' \ \rightarrow \ Q'}
$$

Rules `(par)` and `(brane)` are the contextual rules that respectively permit to a system to execute also if it is in parallel with another process or if it is inside a membrane, respectively. Rule `(strucong)` ensures that two structurally congruent systems have the same reactions.

[2]With abuse of notation we use $\equiv$ to denote both structural congruence on systems and structural congruence on processes.

With $\rightarrow^*$ we denote the reflexive and transitive closure of a relation $\rightarrow$. Given a reduction relation $\rightarrow$, we say that the system P' is a *derivative* of the system P if $P \rightarrow^* P'$; the set of *derivatives* of a system P is denoted by $Deriv(P)$.

We say that a system P *has a divergent computation* (or infinite computation) if there exist an infinite sequence of systems $P_0, P_1, \ldots, P_i, \ldots$ such that $P = P_0$ and $\forall i \geq 0 : P_i \rightarrow P_{i+1}$. We say that a system P *has a terminating computation* if there exists $Q \in Deriv(P)$ such that $Q \not\rightarrow$. We say that *all computations of a system P terminate* if P has no divergent computations.

We use $\prod$ (resp. $\bigcirc$) to denote the parallel composition of a set of processes (resp. systems), i.e., $\prod_{i\in\{1,\ldots,n\}} \sigma_i = \sigma_1 \mid \ldots \mid \sigma_n$ and $\bigcirc_{i\in\{1,\ldots,n\}} P_i = P_1 \circ \ldots \circ P_n$. Moreover, $\prod_{i\in\emptyset} \sigma_i = 0$ and $\bigcirc_{i\in\emptyset} P_i = \diamond$. Finally, $\prod_n \sigma$ (resp. $\bigcirc_n P$) denotes the parallel composition of n copies of process σ (resp. system P).

2.3 Syntax and Interleaving Semantics of MBD

The actions of the MBD calculus, proposed in [3], are inspired by membrane fusion and splitting. To make membrane splitting more controllable, in [3] two more basic operations are used: *budding*, consisting in splitting off one internal membrane, and *dripping*, consisting in splitting off zero internal membranes. Membrane fusion, or merging, is called *mating*.

Definition 4 *Let $Name$ be a denumerable set of ambient names, ranged over by $n, m, \ldots$. The set of actions of MBD is defined by the following grammar:*

$$a \quad ::= \quad mate_n \mid mate_n^{\perp} \mid bud_n \mid bud_n^{\perp}(\sigma) \mid drip(\sigma)$$

Actions $mate_n$ and $mate_n^{\perp}$ will synchonize to obtain membrane fusion. Action bud_n permits to split one internal membrane, and synchronizes with the co-action $bud_n^{\perp}$. Action *drip* permits to split off zero internal membranes. Actions $bud^{\perp}$ and *drip* are equipped with a process σ, that will be associated to the new membrane created by the membrane performing the action.

Definition 5 *The reaction relation for MBD is the least relation containing*

the following axioms, and satisfying the rules in Definition 3:

(`mate`) $mate_n.\sigma|\sigma_0(\!|\, P \,|\!) \;\circ\; mate_n^{\perp}.\tau|\tau_0(\!|\, Q \,|\!) \to \sigma|\sigma_0|\tau|\tau_0(\!|\, P \circ Q \,|\!)$

(`bud`) $bud_n^{\perp}(\rho).\tau|\tau_0(\!|\, bud_n.\sigma|\sigma_0(\!|\, P \,|\!) \circ Q \,|\!) \to \rho(\!|\, \sigma|\sigma_0(\!|\, P \,|\!) \,|\!) \;\circ\; \tau|\tau_0(\!|\, Q \,|\!)$

(`drip`) $drip(\rho).\sigma|\sigma_0(\!|\, P \,|\!) \to \rho(\!|\;|\!) \;\circ\; \sigma|\sigma_0(\!|\, P \,|\!)$

2.4 Maximal Parallelism Semantics of MBD

In this section we introduce a semantics based on maximal progress, and inspired by the standard semantics of Membrane Computing [9]. The idea is that at each computational step, a maximal set of independent reductions is simultaneously executed. Hence, all the membranes that can evolve have to do it. For example, the system

$$mate_a(\!|P|\!) \;\circ\; drip(0)(\!|Q|\!) \;\circ\; mate_a^{\perp}(\!|R|\!)$$

performs the maximal progress move

$$mate_a(\!|P|\!) \;\circ\; drip(0)(\!|Q|\!) \;\circ\; mate_a^{\perp}(\!|R|\!) \Rightarrow 0(\!|P|\!) \;\circ\; 0(\!|Q|\!) \;\circ\; 0(\!|\;|\!) \;\circ\; 0(\!|R|\!)$$

On the other hand, the following move does not involve all the membranes that can evolve, hence it is not allowed:

$$mate_a(\!|P|\!) \;\circ\; drip(0)(\!|Q|\!) \;\circ\; mate_a^{\perp}(\!|R|\!) \not\Rightarrow 0(\!|P|\!) \;\circ\; drip(0)(\!|Q|\!) \;\circ\; 0(\!|R|\!)$$

At each computational step, a membrane can be involved in at most one reduction rule. Hence, also the following move, where three membranes are simultaneously fused, is not allowed:

$$mate_a|mate_b(\!|P|\!) \;\circ\; mate_a^{\perp}(\!|Q|\!) \;\circ\; mate_b^{\perp}(\!|R|\!) \not\Rightarrow 0(\!|P \;\circ\; Q \;\circ\; R|\!)$$

In such case, one of the following computational steps can be performed:

$$mate_a|mate_b(\!|P|\!) \;\circ\; mate_a^{\perp}(\!|Q|\!) \;\circ\; mate_b^{\perp}(\!|R|\!) \Rightarrow \\ mate_b(\!|P \;\circ\; Q|\!) \;\circ\; mate_b^{\perp}(\!|R|\!)$$

$$mate_a|mate_b(\!|P|\!) \;\circ\; mate_a^{\perp}(\!|Q|\!) \;\circ\; mate_b^{\perp}(\!|R|\!) \Rightarrow \\ mate_a(\!|P \;\circ\; R|\!) \;\circ\; mate_a^{\perp}(\!|Q|\!)$$

A maximal parallelism computational step is obtained as a maximal sequence of independent reductions. To formalize this notion, we take a

modified reduction semantics, obtained by "freezing" all the processes associated to a membrane, after that such a membrane has been involved in a reduction. After the execution of a maximal parallelism computational step, the frozen processes are "heated" and can be involved in the next computational step.

To this aim, we extend the grammar of systems with a new term, denoting a membrane whose process is frozen:

$$P, Q \quad ::= \quad \ldots \mid \langle \sigma \rangle (\!| P |\!)$$

The reaction relation is modified as follows:

Definition 6 *The reaction relation $\mapsto$ for MBD is the least relation containing the following axioms, and satisfying the rules in Definition 3 (obtained by replacing $\rightarrow$ with $\mapsto$):*

(`mate`) $mate_n.\sigma|\sigma_0(\!| P |\!) \ \circ \ mate_n^{\perp}.\tau|\tau_0(\!| Q |\!) \mapsto \langle \sigma|\sigma_0|\tau|\tau_0 \rangle (\!| P \circ Q |\!)$

(`bud`) $bud_n^{\perp}(\rho).\tau|\tau_0(\!| bud_n.\sigma|\sigma_0(\!| P |\!) \circ Q |\!) \mapsto$
$\langle \rho \rangle (\!| \langle \sigma|\sigma_0 \rangle (\!| P |\!) |\!) \ \circ \ \langle \tau|\tau_0 \rangle (\!| Q |\!)$

(`drip`) $drip(\rho).\sigma|\sigma_0(\!| P |\!) \mapsto \langle \rho \rangle (\!|\ |\!) \ \circ \ \langle \sigma|\sigma_0 \rangle (\!| P |\!)$

The heating function *heated*( ) transforms the frozen processes of a system in active processes.

Definition 7 *The heating function, called heated(P), is defined inductively on the structure of (the extended set of) systems:*

$$\begin{array}{l} heated(\diamond) = \diamond \\ heated(P \circ Q) = heated(P) \circ heated(Q) \\ heated(!P) = !heated(P) \\ heated(\sigma(\!|P|\!)) = \sigma(\!|P|\!) \\ heated(\langle \sigma \rangle (\!|P|\!)) = \sigma(\!|P|\!) \end{array}$$

Now we are ready to define the maximal parallelism computational step $\Rightarrow$, consisting of a maximal (not extendable) sequence of reductions $\mapsto$.

Definition 8 *Let P, Q be MBD systems (not containing frozen processes). $P \Rightarrow Q$ iff there exists a system Q' such that $P \mapsto^{+} Q'$, $Q' \not\mapsto$ and $Q = heated(Q')$.*

3 Computing with MBD

In this section we investigate the computational power of MBD. We show how to model Register Machines (RAMs) [12], a well known Turing powerful formalism. We start by recalling what RAMs are.

Then, we provide a nondeterministic encoding of RAMs in MBD (with interleaving semantics), which preservers the existence of a terminating computation. The encoding is nondeterministic because it introduces additional computations which do not follow the expected behaviour of the modeled RAM. However, all these computations are infinite. This ensures that, given a RAM, its modeling has a terminating computation if and only if the RAM terminates. A direct consequence of this result is the undecidability of existential termination for MBD.

Finally, we provide a deterministic encoding of RAMs in MBD with maximal parallelism that preserves the existence of a terminated computation (hence also the existence of a divergent computation). Thus we obtain the undecidability of both existential and universal termination for MBD with maximal parallelism.

3.1 Register Machines

RAMs are a computational model based on finite programs acting on a finite set of registers. More precisely, a RAM R is composed of the registers $r_1, \ldots, r_n$, that can hold arbitrary large natural numbers, and by a sequence of indexed instructions $(1 : I_1), \ldots, (m : I_m)$. In [7] it is shown that the following two instructions are sufficient to model every recursive function:

- $(i : Succ(r_j))$: adds 1 to the contents of register r_j and goes to the next instruction;
- $(i : DecJump(r_j, s))$: if the contents of the register r_j is not zero, then decreases it by 1 and goes to the next instruction, otherwise jumps to the instruction s.

The computation starts from the first instruction and it continues by executing the other instructions in sequence, unless a jump instruction is encountered. The execution stops when an instruction number higher than the length of the program is reached.

A state of a RAM is modelled by $(i, c_1, \ldots, c_n)$, where i is the program counter indicating the next instruction to be executed, and $c_1, \ldots, c_n$ are the current contents of the registers $r_1, \ldots, r_n$, respectively.

A state $(i, c_1, \ldots, c_n)$ is *terminated* if the program counter i is strictly greater than the number of instructions m. We say that a RAM R *terminates* if its computation reaches a terminated state.

3.2 A Nondeterministic Encoding of RAMs in MBD with Interleaving Semantics

In this section we show how to obtain a nondeterministic encoding of RAMs. The encoding satisfies the following property. If the RAM terminates, then the encoding has at least one terminating computation; otherwise, no computation of the encoding terminates. Hence, even if the RAM terminates, it may happen that a run of the encoding diverges. This is due to the fact that it is not possible to perform a test for zero on the (representation of the) contents of registers. When a $DecJump$ instruction is performed, one of the two branches (decrement or jump) is chosen nondeterministically. If the right branch is taken, then the encoding behaves correctly. On the other hand, if the wrong branch is taken, then a system is reached such that any computation starting from such a system will diverge.

The modelling of RAMs is based on an encoding function, which transforms instructions and registers independently.

The basic idea for modelling the natural numbers contained in the registers is the following: the natural number n contained in register r_j is represented by n copies of a system R_j collected inside a register membrane. The increment is performed by fusing the register membrane with a membrane containing one copy of R_j, thus obtaining $n + 1$ copies of R_j inside the register membrane. The decrement is performed by mating the register membrane with a membrane whose process permits to perform a budding of one of the systems R_j contained inside the register membrane, thus leaving $n - 1$ copies of R_j inside the register membrane.

Consider a RAM R with instructions $(1 : I_1)$, ..., $(m : I_m)$ and registers r_1, ..., r_n; the encoding of an initial state $(1, c_1, \ldots, c_n)$ is defined as follows:

$$\begin{array}{rcl} [\![(1, c_1, \ldots, c_n)]\!] & = & [\![PC = 1]\!] \circ !\, [\![(1 : I_1)]\!] \circ \ldots \circ !\, [\![(m : I_m)]\!] \circ \\ & & [\![r_1 = c_1]\!] \circ \ldots \circ [\![r_n = c_n]\!] \circ LOOP(\!|\ |\!) \end{array}$$

where $LOOP =! \, mate^{\perp}_{loop}.drip(mate_{loop})$ is the process on the loop membrane, ensuring that the system will diverge if the wrong branch of the encoding of a $DecJump$ instruction is taken. If a membrane $mate_{loop}(\!| \ldots |\!)$ is produced, then such a membrane may fuse with the loop membrane, and another similar membrane is dripped, that may fuse with the loop membrane, and so on, thus preventing the system to terminate.

The encoding of an initial state of the RAM is composed by the following parts: the program counter, (an unbounded number of occurrences of) the encodings of each instruction, the encodings of the initial contents of registers, and the loop membrane.

The encoding of the contents of the program counter is defined as follows:

$$[\![PC = i]\!] = mate_{p_i}(\!|\ |\!)$$

The presence of such a program counter membrane denotes the fact that the next instruction to be executed is I_i. The encoding of the program counter membrane $[\![PC = i]\!]$ will fuse with the encoding of the i-th instruction to activate the execution of such instruction.

The encoding of the contents of register r_j is

$$[\![r_j = c_j]\!] = mate^{\perp}_{opr_j}(\!|\bigcirc_{c_i} R_j|\!)$$

where $R_j = (bud_{decr_j} \mid bud_{loopr_j})(\!|\ |\!)$.

If an increment operation on r_j is executed, then a membrane, containing one copy of R_j, is fused with $[\![r_j = c_j]\!]$, thus obtaining a representation of $[\![r_j = c_j + 1]\!]$.

If a decrement operation on r_j is executed, then a membrane – decorated with a budding instruction on name $decr_j$ – is fused with $[\![r_j = c_j]\!]$. At this point, the only operation that can be performed by the register membrane is such a budding. If $c_j > 0$, then at least one copy of R_j is present in the register membrane; by performing action bud_{decr_j}, one copy of R_j is "expelled" from the register membrane. Such an expelled copy is surrounded by a membrane with an empty program, hence becoming an innocuous garbage that can neither perform reductions nor interact with the other membranes. If $c_j = 0$, then the register membrane contains no membranes and no further operation can be performed by the register membrane.

If the zero branch is selected, then a membrane – decorated with a budding instruction – is fused with $[\![r_j = c_j]\!]$, and a new system $[\![r_j = 0]\!]$ is produced. If $c_j = 0$, then the old register membrane contains no membranes inside; as the only instruction that the old register membrane can perform is a budding, it becomes innocuous garbage. If $c_j > 0$, then the old register membrane contains at least one copy of R_j; such R_j can be expelled, and surrounded by a membrane that can activate the loop membrane, thus starting a divergent computation.

The encoding for the instruction $(i : I_i)$ is as follows:

$$[\![(i : Succ(r_j))]\!] = mate^{\perp}_{p_i}.mate_{opr_j}.drip(mate_{p_{i+1}}).mate^{\perp}_{opr_j}(\!|R_j|\!)$$

$$[\![(i : DecJump(r_j, s))]\!] = DECR_{i,j,s}(\!|\ |\!) \mid ZERO_{i,j,s}(\!|\ |\!)$$

where

$$\begin{aligned} DECR_{i,j,s} &= mate^{\perp}_{p_i}.mate_{opr_j}.drip(mate_{loop}).bud^{\perp}_{decr_j}(0). \\ &\quad mate^{\perp}_{loop}.drip(mate_{p_{i+1}}).mate^{\perp}_{opr_j} \\ ZERO_{i,j,s} &= mate^{\perp}_{p_i}.mate_{opr_j}.drip(mate^{\perp}_{opr_j}).drip(mate_{p_s}). \\ &\quad bud^{\perp}_{loopr_j}(mate_{loop}) \end{aligned}$$

The encoding of each instruction consists of a membrane, and the encoding of a RAM contains an unbounded number of copies of the encoding of each instruction.

When a program counter system $mate_{p_i}$ appears at top-level, an (occurrence of) instruction $(i : I_i)$ is activated by fusing it with the program counter.

If the i-th instruction is an increment of register r_j, and the actual contents of r_j is k, then the instruction membrane is fused with the register membrane by performing $mate_{opr_j}$. As the instruction membrane for increment, $[\![(i : Succ(r_j))]\!]$, contains one copy of system R_j, now the register+instruction membrane (the result of the fusion of register membrane and instruction membrane) contains $k + 1$ copies of R_j. At this point, the program counter membrane corresponding to instruction $i + 1$ is dripped, and the register+instruction membrane becomes the register membrane corresponding to $[\![r_j = k + 1]\!]$, and is ready to accept the execution of new operations on the register.

Suppose that the i-th instruction is a decrement of register r_j, or jump to instruction s if the contents of r_j is zero. Independently of the actual contents of register r_j, the program counter membrane is fused with either the decrement part or the zero part of the instruction, thus selecting nondeterministically one of the two branches of the $DecJump$ instruction.

Suppose that the decrement part is selected. The instruction membrane is fused with the register membrane by performing $mate_{opr_j}$, and a loop activator membrane $mate_{loop}(\!|\ |\!)$ is dripped. Now the register+instruction membrane is ready to perform a budding of a copy of R_j. Two cases can happen:

- If the contents of r_j is not zero, e.g., $r_j = k + 1$, the right branch has been chosen. Moreover, the register+instruction membrane contains at least one copy of R_j. Hence, the budding operation is performed, and the expelled copy of R_j is surrounded by a membrane with an empty program, thus producing innocuous garbage. Now the register+instruction membrane contains k copies of R_j. The loop activator membrane is removed (by fusing it with the register+instruction

membrane by operation $mate^{\perp}_{loop}$) and the program counter membrane corresponding to instruction $i+1$ is dripped. At this point, the register+instruction membrane becomes the register membrane $[\![r_j = k]\!]$, and is ready to accept the execution of new operations on the register.

- If $r_j = 0$, then the wrong branch has been chosen. Moreover, the register+instruction membrane contains no membranes. As the only instruction that can be performed by the register+instruction membrane is a budding, no other reduction or interaction can be performed by such a membrane. No other computation is possible, but the fusion of the loop activator membrane with the loop membrane. At this point, the computation can only diverge.

Suppose that the zero part is selected. The instruction membrane is fused with the register membrane by performing $mate_{opr_j}$. A new register membrane $[\![r_j = 0]\!]$ and a program counter $mate_{p_s}$ are produced, thus the computation continues from instruction s. Now the old register+instruction membrane can only perform a budding $bud_{loopr_j}(mate_{loop})$:

- If $r_j = 0$, then the right branch has been chosen. Moreover, the old register+instruction membrane contains no membranes. As the only operation the old register+instruction membrane can perform is a budding, it has become innocuous garbage.

- If the contents of r_j is not zero, e.g., $r_j = k+1$, the wrong branch has been chosen. Moreover, the old register+instruction membrane contains at least one copy of R_j. Hence, the budding operation is performed, and the expelled copy of R_j is surrounded by a membrane with program $mate_{loop}$, that can fuse with the loop membrane, thus preventing the computation to terminate.

We can now conclude with the Theorem which states that our modelling of RAMs preserves existential termination.

Theorem 9 *Let R be a RAM with program $(1 : I_1), \ldots, (m : I_m)$ and initial state $(1, c_1, \ldots, c_n)$. Then we have that the RAM R terminates if and only if the system $[\![(1, c_1, \ldots, c_n)]\!]$ has a terminating computation.*

3.3 A Deterministic Encoding of RAMs in MBD with Maximal Parallelism Semantics

In this section we show how to obtain an encoding that behaves deterministically under the maximal parallelism hypothesis.

The modeling of the RAM is quite similar to the one of the previous section. The key idea is to use the maximal progress hypothesis to ensure that the right branch of a *DecJump* instruction is taken. Both the decrement and the zero branches of the instruction are activated in parallel, but the execution of the relevant part of the zero branch is delayed by innocuous *drip*(0) operations, so that the zero branch will be executed only if the decrement branch fails.

The modelling of the contents of registers and of the increment instruction is the same as for the previous encoding, but in the present encoding all the components are surrounded by an external membrane. Such an external membrane permits to bud the garbage membranes that are not innocuous but could interfere with the correct components.

For completeness, here we report the whole encoding, and we highlight the differences.

Consider a RAM R with instructions $(1 : I_1)$, ..., $(m : I_m)$ and registers r_1, ..., r_n; the encoding of an initial state $(1, c_1, \ldots, c_n)$ is defined as follows:

$$\begin{array}{rcl} [\![(1, c_1, \ldots, c_n)]\!] & = & EXT(\!| \; [\![PC = 1]\!] \circ \\ & & ! \; [\![(1 : I_1)]\!] \; \circ \; \ldots \; \circ \; ! \; [\![(m : I_m)]\!] \; \circ \\ & & [\![r_1 = c_1]\!] \; \circ \; \ldots \; \circ \; [\![r_n = c_n]\!] \; |\!) \end{array}$$

where $EXT =! \, bud^{\perp}_{ext}(0)$ is the process surrounding the external membrane, permitting to expell the garbage membranes.

The encoding of the program counter is the same as in the previous section, whereas the encoding of the contents of registers is slightly simpler (as it is no longer necessary to start a loop in the case the wrong branch is taken):

$$\begin{array}{rcl} [\![PC = i]\!] & = & mate_{p_i}(\!| \; |\!) \\ [\![r_j = c_j]\!] & = & mate^{\perp}_{opr_j}(\!| \bigcirc_{c_i} R_j |\!) \end{array}$$

where

$$R_j = bud_{decr_j}(\!| \; |\!)$$

The main difference w.r.t. the previous section is represented by the encoding of the *DecJump* instruction, whereas the encoding of the *Succ* instruction is unchanged:

$$[\![(i : Succ(r_j))]\!] = \quad mate^{\perp}_{p_i}.mate_{opr_j}.drip(mate_{p_{i+1}}).mate^{\perp}_{opr_j}(\!|R_j|\!)$$

$$\begin{aligned}[\![(i : DecJump(r_j, s))]\!] = \quad & mate^{\perp}_{p_i}.mate_{opr_j}.drip(ZERO_{i,j,s}).\\ & drip(mate_{dor_j}).\\ & bud^{\perp}_{decr_j}(drip(0).mate_{zero}.bud_{ext}).\\ & mate^{\perp}_{dor_j}.drip(0).drip(0).drip(mate_{p_{i+1}}).\\ & mate^{\perp}_{opr_j}(\!|\ |\!)\end{aligned}$$

where

$$\begin{aligned}ZERO_{i,j,s} \quad = \quad & drip(0).drip(0).drip(0).mate^{\perp}_{dor_j}.drip(mate^{\perp}_{opr_j}).\\ & drip(drip(0).mate_{p_s}).bud_{ext} \ |\\ & mate^{\perp}_{zero}\end{aligned}$$

As in the previous section, instruction $(i : I_i)$ is activated by fusing it with the program counter membrane $mate_{p_i}$.

Suppose that the i-th instruction is a decrement of register r_j, or jump to instruction s if the contents of r_j is zero.

The instruction membrane is fused with the register membrane by performing $mate_{opr_j}$, and a zero branch membrane with process $ZERO_{i,j,s}$ is dripped.

Also a mutual exclusion membrane $mate_{dor_j}(\!|\ |\!)$ is dripped, and the zero branch membrane perform the first innocuous $drip(0)$.

Now the register+instruction membrane is ready to perform a budding of a copy of R_j. Two cases can happen:

- If the contents of r_j is not zero, e.g., if $r_j = k + 1$, then the register+instruction membrane contains at least one copy of R_j. Hence, the budding operation is performed, and the expelled copy of R_j is surrounded by a the membrane with process $drip(0).mate_{zero}.mate_{garb}$. The zero branch membrane performs the second $drip(0)$.

  Now the register+instruction membrane contains k copies of R_j.

  The register+instruction membrane removes the membrane for mutual exclusion by performing $mate^{\perp}_{dor_j}$, the zero branch membrane performs the third $drip(0)$ and the membrane surrounding the expelled R_j performs the $drip(0)$.

  At the next step, the register+instruction membrane performs the first $drip(0)$, and the zero branch membrane fuses with the membrane

surrounding the expelled R_j by performing $mate^{\perp}_{zero}$. Note that the zero branch membrane can no longer perform $mate^{\perp}_{dorj}$, because the mutual exclusion membrane has been already been removed.

At the next step, the register+instruction membrane performs the second $drip(0)$, and the membrane, obtained by fusing the zero branch membrane with the membrane surrounding the expelled R_j, is expelled from the external membrane, and surrounded by a membrane with empty process.

At the next step, the only active membrane is the register+instruction membrane, that produces the program counter membrane $mate_{p_{i+1}}(\!|\ |\!)$; now the register+instruction membrane has become the register membrane $[\![r_j = k]\!]$, and is ready to accept the execution of new operations on the register.

- If $r_j = 0$, then no membrane is contained in the register+instruction membrane. Hence, the register+instruction membrane is blocked on the budding instruction. As no membrane can be fused with it, the register+instruction membrane has become innocuous garbage. The only active membrane is the zero branch membrane, which performs the two $drip(0)$, then it consumes the mutual exclusion membrane by performing $mate^{\perp}_{dorj}$. A new register membrane $[\![r_j = 0]\!]$ is produced by performing $drip(mate^{\perp}_{opr_j})$. A quasi program counter membrane $drip(0).mate_{p_s}(\!|\ |\!)$ is produced.

  At the next step, the quasi program counter membrane performs the $drip(0)$ and becomes the program counter $mate_{p_s}(\!|\ |\!)$, and the zero branch membrane performs the budding. Hence, the zero branch membrane has been expelled outside the surrounding external membrane, and surrounded by a membrane with empty program, thus becoming innocuous garbage.

We can now conclude with the Theorem which states that our modelling of RAMs faithfully represents the behaviour of the RAM.

Theorem 10 *Let R be a RAM with program $(1 : I_1), \ldots, (m : I_m)$ and initial state $(1, c_1, \ldots, c_n)$. Then we have that the RAM R terminates if and only if all the computations of the system $[\![(i, c_1, \ldots, c_n)]\!]$ terminate.*

4 Conclusion

We investigated the expressiveness of two different semantics (interleaving and maximal parallelism) for the MBD brane calculus w.r.t. the ability to encode computable functions.

Even if the underlying formalisms are different, the present work is intimately connected with the result in [5], namely, the Turing equivalence of P systems with mate and drip operations. A deep comparison of the two formalisms deserves a further investigation; however, at a first sight, it seems that the interaction primitives in the P systems defined in [5] are more powerful that the primitives of the MBD calculus. Moreover, in [5] only the halting computations are considered as successful, but it is not clear if a deterministic encoding of RAMs can be provided in P systems with mate and drip.

As observed in [5], it is not clear if moving to an interleaving semantics leads to a decrease of the computational power.

In [5], only a finite number of membranes is needed to obtain Turing equivalence, whereas in the present paper an unbounded number of membranes is required. The (im)possibility to encode RAMs in MBD with a fixed number of membranes deserves further investigation. Probably the technique adopted in [2] to reduce the process calculus Mobile Ambients on Petri nets [11] could provide some inspiration for an impossibility result.

Finally, [5] obtains Turing equivalence with mate and drip primitives; we plan to investigate what is the impact of the removal of budding on the computational expressiveness of the MBD brane calculus.

Acknowledgements

A special thanks to my dad, for encouraging me to do research and to write this paper in a difficult period of my life.

References

[1] N. Busi, R. Gorrieri: On the computational power of Brane Calculi. *Proc. Third Workshop on Computational Methods in Systems Biology* (CMSB'05), Edinburgh, Scotland (2005), 106–117.

[2] N. Busi, G. Zavattaro: Deciding Reachability in Mobile Ambients. Proc. ESOP'05, *Lecture Notes in Computer Science* **3444**, Springer, Berlin (2005), 248–262.

[3] L. Cardelli: Brane Calculi - Interactions of biological membranes. Proc. Computational Methods in System Biology (CMSB 2004), Paris, France, May 2004. Revised Selected Papers. *Lecture Notes in Computer Science* **3082**, Springer, Berlin (2005), 257–280.

[4] L. Cardelli, A.D. Gordon: Mobile Ambients. *Theoretical Computer Science* **240** (1) (2000), 177–213.

[5] L. Cardelli, G. Păun. A universality result for a (Mem)Brane Calculus based on mate/drip operations. *International Journal of Foundations of Computer Science*, to appear.

[6] R. Freund: Asynchronous P Systems and P Systems Working in the Sequential Mode. Proc. 5th International Workshop on Membrane Computing (WMC2004), Milan, Italy, 2004. Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **3365**, Springer, Berlin (2005), 36–62.

[7] M.L. Minsky: *Computation: finite and infinite machines*. Prentice-Hall (1967).

[8] G. Păun: Computing with membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108–143.

[9] G. Păun: *Membrane Computing: An Introduction*. Springer, Berlin (2002).

[10] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, E. Shapiro: BioAmbients: An Abstraction for Biological Compartments. *Theoretical Computer Science* **325** (1) (2004),141–167.

[11] W. Reisig: *Petri nets: An Introduction.* EATCS Monographs in Computer Science, Springer, Berlin, 1985.

[12] J.C. Shepherdson, J.E. Sturgis. Computability of recursive functions. *Journal of the ACM* **10** (1963), 217–255.

A Membrane Computing System Mapped on an Asynchronous, Distributed Computational Environment

Giovanni CASIRAGHI, Claudio FERRETTI, A. GALLINI, Giancarlo MAURI

Università degli Studi di Milano Bicocca,
Dipartimento di Informatica, Sistemistica e Comunicazione
via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
E-mail: ferretti@disco.unimib.it

Abstract

We show how to simulate a membrane system on a (simulated) distributed and bio-inspired computational architecture (BME). The advantages of this approach are the ease of representing each membrane with a processing element and the perspective of exploiting the expected nanotechnological implementation of such an architecture.

By combining these two non-conventional computing architectures, we touch interesting subproblems, such as the trade-off between being synchronous (P-systems) and asynchronous (BME), or between structural adjacency and position-independent communications.

1 Introduction

Processing elements (PE) arrays have gained attention as suitable architectures for computational machines based on molecular scale device. Thanks to their regular structure this kind of systems make design simpler, and could allow manufacturing techniques based on molecular self-organization. Lack of a clock avoids signal propagation troubles, but imposes an asynchronous interaction system based on messages exchange. Having such constraints on computational architecture and communication subsystem, the issue about which computational model to employ on them arises.

We want to investigate how to map the membrane synchronous computations on such an environment; in particular, we will describe a technique

to implement P-systems on an asynchronous PE grid by employing a simulation environment for asynchronous parallel computational system, named Bio-Molecular Engine(BME).

BME in short

BME ([2]) is a graphical simulation environment for distributed computational systems architecture. Every simulated system has, as elementary unit, a processing element (PE, or "cell"), characterized by a certain computational capability defined by the virtual machine the user associates to it. Every cell is provided with one or more net-interface to manage messages incoming/outgoing to the computing environment, and all cells are linked together in a toroidal mesh.

Cells dynamically acquire a code to execute from the environment, and they themselves can ask to the environment the availability of a cell to be dedicated to the execution of a certain piece of code. Cells internally execute operations specified by the user who, in the BME simulator, can choose the formalism used to write the code (e.g. a simple specialized programming language, or plain Java). However, a small group of primitives has to be used for environment dependent operations (e.g. assignment of code to cells, communication, and so on). In particular, for our purposes three primitives have been employed:

- Run: used by a cell to dynamically associate a code to another cell in the environment. The new specialized cell executes the code without inheriting constraints from the cell which has instantiated it;
- Send: a cell sends a BME-data-message containing a small amount of information;
- Wait: a cell waits for an incoming BME-data-message.

Motivations

Both BME computing architecture and membrane systems have biological inspiration. In particular, BME considers features like: staminal (redundant) cells, specialization, tissue-like topology, massive parallelism. But there are also significant differences, for instance: P-systems have nested structures, while BME has PEs in a lattice structure, the former usually have a clocked synchronous evolution of computation, while the latter has PEs cooperating without a clock.

Given this kind of comparison, we thought that the two models could be interestingly combined, with BME simulating the P-systems; we look forward to advantages like the exploitation of current highly flexible and distributed BME simulating environment, but also future (nanoscale)hardware implementation of BME architecture, and thus also of P-systems.

2 The Simulated Membrane System Model

We define membrane systems (P-systems) omitting some definitions, which can be found, for instance, in [4].

A *P-system* is defined by a tuple:

$$\Pi = (O, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where:

1. O is an alphabet, whose elements are called *objects*;
2. μ is a nested membrane structure;
3. w_i, $1 \leq i \leq m$, are multisets on O, describing objects initially contained in regions $1, 2, \ldots, m$ of μ;
4. R_i, $1 \leq i \leq m$, are finite sets of evolution rules on strings of objects from O; R_i is the set of rules active in region i of μ; an evolution rule has form $u \rightarrow v$, where u is a string on O and v is a string on $O_{tar} = (O^* \times TAR) \cup (O^* \times TAR \times \{\delta\})$, with $TAR = \{here, out, in\} \cup \{in_j \mid 1 \leq j \leq m\}$;
5. if a rule ends with the symbol δ, then after its application the membrane to which the rule belongs *dissolves*, and the objects contained in that membrane move to its parent membrane;
6. $i_0 \in \{1, 2, \ldots, m\}$ is the label of one of the membranes (*output membrane*);

Evolution rules are applied in each membrane in a *maximally parallel* way, and each membrane applies its rules and then communicates the results in *synchronous parallelism* with other membranes. In case that more than one alternative rule matchings appear, then a *non-deterministic choice* is made. Finally, each evolution rule states, with symbols from TAR, where each of its results have to be sent. If destination is in, without subscript, then the

result is sent to one of inner membranes, non-deterministically chosen, if available.

A computation in the membrane system ends when no membrane can apply any rule.

3 Simulation of P-Systems on the BME Environment

Both P-systems and BME simulated architectures are distributed systems, but while BME cells interact asynchronously P-systems are synchronous i.e. there is a sort of master clock that forces the transitions of data among membranes to occur together at any tick, defining a function from natural numbers (time axis) and the global states of a P-system.

The basic idea of our simulation is to associate a membrane to a single cell, internally keeping a data structure tracing relationships with other membranes (inner membranes and parent membrane). The topology of the membrane system is thus not represented by relative positions of simulating PEs, chosen on the lattice by core algorithms of BME computational model. We use BME-primitives for message exchange to simulate the synchronous behaviour of P-systems. A P-system has a tree structure and when it is mapped on a BME virtual architecture, it is dynamically built starting from a single cell (the root of the tree), named *skin* membrane, that executes the BME primitive "Run" for each of its inner membranes; the same policy is employed by inner membranes and so on until the leaves of the tree are activated (i.e. membranes with no inner membranes).

The P-systems simulator we developed consists of a Java package, which exploits two kinds of messages: *data messages* to pass data between two membranes and *event messages* (both implemented by BME-data-messages) to communicate events about the evolution of system's structure (i.e.: when a inner membrane dissolves it communicates the results of this event to its parents and to its inner membranes).

Given the asynchronous nature of the BME sub-system, every cell, i.e. every simulated membrane, evolves autonomously. The evolution of a membrane is a sequence of computational steps, where each step can be described by a loop of four subsequent stages:

1. internal evolution rules application (to objects acquired in the previous iteration);

2. the sending of data to other PEs-membranes, by exploiting BME communication sub-system, and the waiting for incoming data messages;

3. the sending of event messages to signal structural (i.e. inner/parent membranes) changes, and the waiting for incoming event messages;

4. update of local messages buffer, containing messages arrived out of sequence.

In fact, since the evolution of internal state of the membranes is untied with the rest of the environment, it is not possible, at receiver side, to state to which simulated P-system's step an incoming message is related;

An approach to this problem is to introduce packets numbering policies in order to associate them to a computational step of the emitting membrane. When a membrane M, at computational step n, receives a message (data or event) emitted by another membrane S at step $n+k$ (k is the communication "window"), M stores it into a buffer. When M will be at step $n+k$, it will be able to consume the received message. The problem is to verify if an upper bound exists for k, so to easily manage incoming traffic.

Another important issue is to maintain the structural consistency of the system: in a asynchronous system, the membranes can expire (dissolve) independently by the state of other membranes.

To solve both "window upper-bound" and "structural consistency" problems we have defined a *message processing protocol* (MPP) at the level of the single membrane that covers the second and the third point of the previously described algorithm for simulated membrane evolution. MPP is essentially organized in such a way that, for instance, a PE-membrane, which has a parent membrane and at least a inner membrane, will strictly alternate the exchange of data messages and event messages.

In this way, it can be proved that, for instance, even if we can't be sure about delays and reordering of messages traveling along the lattice toward the PE associated to my parent membrane, nonetheless we will receive messages associated to simulated evolutions far at most one step from what we are now simulating in our PE-membrane.

This allowed us to simulate a synchronous P-system by means of a set of asynchronous PEs in BME, just by numbering messages and allowing PEs to have a buffer memory for the small amount of messages which can arrive one step apart from what we are simulating.

4 Examples

The P-system we want to simulate on BME is described by a set of simple Java modules, one for each membrane, and each of them will be transported to a single PE which will execute it. Inside each module the starting multiset and the set of rules of the associated membrane are defined by simple strings.

A multiset $x_1^{n_1} x_2^{n_2} \ldots x_k^{n_k}$, where $x_1, x_2, \ldots, x_k$ are objects and $n_1, n_2, \ldots$ $\ldots, n_k \in N$, is represented by:

```
[(x1:n1),(x2:n2),...,(xk:nk)]
```

For instance, the multiset a^2b^3c is represented by:

```
[(a:2),(b:3),(c:1)]
```

Evolution rules have a LHS, which is a multiset, and a RHS which lists resulting multisets and their destination, e.g.: $b_{here}(a^2c)_{out}d_{in}$. This example becomes:

```
[(b:1)].here[(a:2),(c:1)].out[(d:1)].in
```

If the destination is a specific membrane, it can be specified. For instance, $d_{in_{two}}$ becomes:

```
[(d:1)].in\_two
```

LHS and RHS are separated by "`=>`". Operator δ, possibly appearing as suffix of evolution rules is represented by "`!`". Therefore, a complete evolution rule is:

```
[(a:2),(b:3)] => [(b:1)].here[(a:2),(c:1)].out[(d:1)].in!
```

Each membrane has a name, which also is the name of the corresponding Java module. The skin membrane has always the name `Main()`.

Each module will:

- build a `RuleList` (using also `parse()`);
- build the starting multiset of the membrane (with `parse()` of a corresponding Java class);
- create and instantiate on BME the (possible) inner membranes ("Run" primitive of BME);

- finally, start itself.

An instance membrane follows: it has name "one", it is inside *skin* membrane, and it has a inner membrane named "two". It has the evolution rule $d \rightarrow e^2_{out} f_{in} \delta$ and a starting multiset d^2:

```
one() {

    bmeModuleSig("null");

    rl = new RuleList();
    rl.add(Rule.parse("[(d:1)] => [(e:2)].out[(f:1)].in!"));
    c = MembraneContent.parse("[(d:2)]");

    m = new Membrane(new MembraneName("one"),
                     Membrane.SKIN_MEMBRANE_NAME, c, rl);
    m.addInnerMembraneName(new MembraneName("two"));
    bmeRun("two");

    bmeMembrane = new BmeMembrane(own, m);
    while (!bmeMembrane.stop()) bmeMembrane.evolve();

}
```

5 Discussion

Our approach is not focused on providing a powerful simulator of P-systems, but it aims instead at discovering synergies and/or contrasts between these different non-conventional computing models. Results and further references about software simulation of P-systems can be found, for instance, in [3].

By using our already well developed BME simulator, rich in graphical visualization tools and in features like tools for doing heavy simulations on distributed high-performance clusters, we easily obtained a good user interface for displaying simulated P-systems. Moreover, since BME is being defined together with researchers from the nanotechnology field, with the aim of making that computational model implementable on (future) nano-scale devices, we like to think that in this way also P-systems could be brought to specialized hardware.

An important detail about the system we presented here, is that our BME-based simulator substitutes non-deterministic choices with random

choices, thus making it more suited to simulation of *confluent* P-systems (that is, systems where, for a given input, all the computations produce the same output).

Two interesting issues are worth of further development. One is that of how well time-independent P-systems ([1]) could be simulated on the asynchronous BME architecture. Another one is that of looking for higher performance by using bigger message buffers in PEs, while exploiting pipelined flow control algorithms to build correct message streams at destination, instead of the current message protocol, essentially corresponding to a pipeline of size 1.

References

[1] M. Cavaliere, D. Sburlan: Time-independent P systems. In: G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. International Workshop WMC 2004, Milan, Italy, 2004, Revised Selected and Invited Papers, *Lecture Notes in Computer Science* **3365**, Springer-Verlag, Berlin (2005), 239–258.

[2] A. Gallini, C. Ferretti, G. Mauri: Bio Molecular Engine: A bio-inspired environment for models of growing and evolvable computation. *Genetic and Evolutionary Computation Conference (GECCO)'05.*

[3] M. A. Gutiérrez-Naranjo, M. J. Pérez-Jiménez, A. Riscos-Nùnez: A Simulator for Confluent P-systems. *Second Brainstorming Week on Membrane Computing* (2004), 169–184.

[4] Gh. Păun: *Membrane Computing: An Introduction.* Springer-Verlag, Berlin (2002).

P Systems with Memory*

Paolo CAZZANIGA, Alberto LEPORATI, Giancarlo MAURI, Claudio ZANDRON

Università degli Studi di Milano Bicocca,
Dipartimento di Informatica, Sistemistica e Comunicazione
via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
E-mail: {cazzaniga,leporati,mauri,zandron}@disco.unimib.it

Abstract

We propose P systems in which solution of previously executed computations can be stored in a sub-system composed by a number of added membranes which act as memory elements. When a new input is inserted into the system, the computation on that input is started in parallel with the search for the corresponding solution in all memory membranes. If the solution is found in memory, then a copy of it is expelled from that memory membrane; the search in all other memory membranes is stopped, and the same is done with the computing sub-system. If no solution for that input is found, then the computation produce the solution which is stored in a memory cell.

1 Introduction

The P systems were introduced by Gh. Păun in [3] as a class of distributed parallel computing devices of a biochemical type, inspired by the functioning of the cell.

The basic model consists of a membrane structure composed by several cell-membranes, hierarchically embedded in a main membrane called the skin membrane. The membranes delimit regions and can contain objects. The objects evolve according to given evolution rules associated with the regions. A rule can modify the objects and send them outside the membrane or to an inner membrane. Moreover, the membranes can be dissolved. When

*This work has been supported by the Italian Ministry of University (MURST), under project FIRB-01 "Biomolecular Algorithms to solve NP-Complete problems".

a membrane is dissolved, all the objects in this membrane remain free in the membrane placed immediately outside, while the evolution rules of the dissolved membrane are lost. The skin membrane is never dissolved.

The evolution rules are applied in a maximally parallel manner: at each step, all the objects which can evolve should evolve. A computation device is obtained: we start from an initial configuration and we let the system evolve. A computation halts when no further rule can be applied. The objects in a specified output membrane are the result of the computation.

Further information concerning P systems can be found in [7] and at the Internet web address: `http://psystems.disco.unimib.it`

In this paper we propose P systems in which solutions of already executed computations can be stored in some static memory cells, to speed-up computations. In fact, one possible way to speed-up computations would be the creation of new membranes by means of division (see, e.g., [6]). This approach requires a number of membranes which can grow exponentially. Here, the defined systems have two main subcomponents: the first sub-component is a standard computing system; the second sub-component is constituted by a fixed number of membranes which act as memory elements. Each time a solution is produced by the computing sub-system, we store that solution in a memory membrane. When a new input is inserted in the system to start a new computation, we start in parallel the search for the solution in all memory membranes and, at the same time, the computation on that input in the computing sub-system.

If the solution is found in one memory cell before the computation ends, then a copy of the solution is expelled from that memory membrane; the search in all other memory membranes is stopped, and the same is done with the computing sub-system. We will show that the search for a solution stored in memory can be done in linear time with respect to the input length. As a consequence, the system could be effectively used in all cases when the computation time order is greater than linear but the output for the same input is requested many times with a high probability within a short time, such as image and vocal processing applications.

In case the solution is not found in any memory membrane, then the computing sub-system produces the new solution, which is stored in a free memory membrane. In order to keep at least one memory membrane available all the time, oldest solutions are deleted from memory when necessary.

We will implement such systems by means of usual rewriting P systems which make use of the following features:

- **Membranes of variable thickness:** membranes can be made thicker

or thinner (also dissolved, as said before), in order to change its permeability with respect to the passage of object through them;

- **Membrane polarization:** electrical charges are associated to membranes: they can be marked with "positive" (+), "negative" (−) or "neutral" (0). The rules are applied to strings according to electrical charges of the membrane where the rule is applied;

- **Replicated rewriting rules:** the rules allow to create k copies of the string starting from a single copy. A symbol in the original copy is deleted from the string to create k different strings; in each of these copies, the symbol deleted is replaced by a specific sub-string and each obtained string is then sent to a specific target membrane (as usual, it can remain in the same membrane, can be sent to the region immediately outside or can be sent to an immediately inner membrane).

For further details about P systems and about the features of the previously described features, we refer to [1, 2, 4, 8, 9, 11, 12]

The rest of the paper is organized as it follows. In section 2, we give formal definitions for rewriting P systems with active membranes. In section 3, we give the description of P systems with memory. In section 4, we describe the functioning of such system. Finally, section 5 contains conclusions and perspective for future work.

2 Rewriting P Systems with Active Membranes

We will not recall here the basic definitions of P systems. We refer for details to [7]. For elements of Formal Language Theory, we refer to [10].

In the following, we will make use of *Replicated Rewriting P Systems with polarized membranes of variable thickness*. In such systems, objects can be described by finite strings over a given finite alphabet. The evolution of an object will correspond to a transformation of a string, by means of context free rewriting rules. The evolution of objects depends on electrical charges associated with the membranes. The thickness of membranes can be modified, to dissolve them or to obtain thicker membranes that are impermeable to the passage of objects. Finally, a string can be replicated to obtain more copies starting from a single one.

Formally, such a system of degree $n, n \geq 1$, is defined as it follows:

$$\Pi = (V, T, H, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n, i_0), \text{ where:}$$

- V is an alphabet (the total alphabet of the system);
- $T \subseteq V$ is the terminal alphabet;
- H is a finite set of labels for membranes;
- μ is a membrane structure, consisting of n membranes, labeled (not necessarily in a one-to-one manner) with elements of H; all membranes in μ are supposed to be initially neutral;
- $M_i, 1 \leq i \leq n$ are finite languages over V.
- i_0 is the label of the output membrane. If i_0 is omitted, then the output is collected in the region outside the skin membrane.
- $R_i, 1 \leq i \leq n$ are finite sets of evolution rules.

  The rules are context free evolution rules of the following form:

  (a) $[_h a \rightarrow v]_h^\alpha$ or $[_h a \rightarrow v\gamma]_h^\alpha$, for $h \in H$, $a \in V$, $v \in V^*, \alpha \in \{+, -, 0\}$, $\gamma \in \{\delta, \tau\}$ (string evolution rules),

  (b) $a[_h]_h^{\alpha_1} \rightarrow [_h v]_h^{\alpha_2}$ or $a[_h]_h^{\alpha_1} \rightarrow [_h v\gamma]_h^{\alpha_2}$, where $a \in V$, $v \in V^*$, $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}, \gamma \in \{\delta, \tau\}$ (the symbol a in the string is rewritten in v and the obtained string is introduced in membrane h),

  (c) $[_h a]_h^{\alpha_1} \rightarrow [_h]_h^{\alpha_2} v$ or $[_h a]_h^{\alpha_1} \rightarrow [_h]_h^{\alpha_2} v\gamma$, where $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a \in V$, $v \in V^*\gamma \in \{\delta, \tau\}$ (the symbol a in the string is rewritten in v and the obtained string is sent out from membrane h to the region immediately outside),
  (d) $[_h a]_h^{\alpha_1} \rightarrow [_h v_1\gamma_1(tar_1)||v_2\gamma_2(tar_2)||...||v_k\gamma_n(tar_k)]_h^{\alpha_2}$, where $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a \in V$, $v_i \in V^*$, γ_i is empty or $\gamma_i \in \{\delta, \tau\}$, $1 \leq i \leq k$, $tar_i \in \{here, out, j \mid j$ is the label of a membrane immediately inside membrane $h\}$, $(1 \leq i \leq k)$ (the string a is replicated in k copies and then a symbol a in each of them is replaced by the corresponding substring v_i; each string is then communicated to a target membrane specified by the target label).

These rules are applied accordingly to the following principles:

1. If a rule contains the special symbol δ and the membrane where this rule is applied has thickness 1, then that membrane is dissolved and it is no longer recreated; the objects in the membrane become objects of the membrane placed immediately outside, while the rules of the dissolved membrane are removed. If the membrane has thickness 2,

this symbol reduces the thickness to 1. The skin membrane is never dissolved.

2. If a rule contains the special symbol τ, the thickness of the membrane where this rule is applied is increased; the thickness of a membrane of thickness 2 is not further increased. If a membrane has thickness 2, then no object can pass through it. All rules involving a passage through a membrane of thickness 2 cannot be applied until the thickness is reduced to 1 by means of another rule which introduce the symbol δ in that membrane (note that this is also the case for replicated rewriting rules: if a replicated string cannot reach its target, then the whole rule cannot be applied).

3. If both the symbols δ and τ are introduced in the same region at the same time (by applying two or more different rules on two or more different objects), then the corresponding membrane preserves its thickness.

4. The communication of objects has priority on the actions of δ and τ; if at the same step an object has to pass through a membrane and a rule changes the thickness of that membrane, then we first transmit the object and after that we change the thickness.

5. All objects evolve in parallel: at each step of computation, an object can be modified by only one rule, non-deterministically chosen among all applicable rules, but any object which can evolve by a rule of any form, must evolve.

6. All objects and membranes not specified in a rule and which do not evolve are passed unchanged to the next step.

The membrane structure at a given time, together with all strings associated with the regions defined by the membrane structure, is the *configuration* of the system at that time. The *initial configuration* is $(\mu, M_1, \ldots, M_n)$. We can pass from a configuration to another one by using the rules in R, according to the principles previously described (we call this a *transition*). A *computation* is a sequence of transitions between configurations. A computation *halts* when there is no rule which can be applied to objects and membranes in the current configuration. The output of the computation consists of all strings in membrane i_0 when the computation halts.

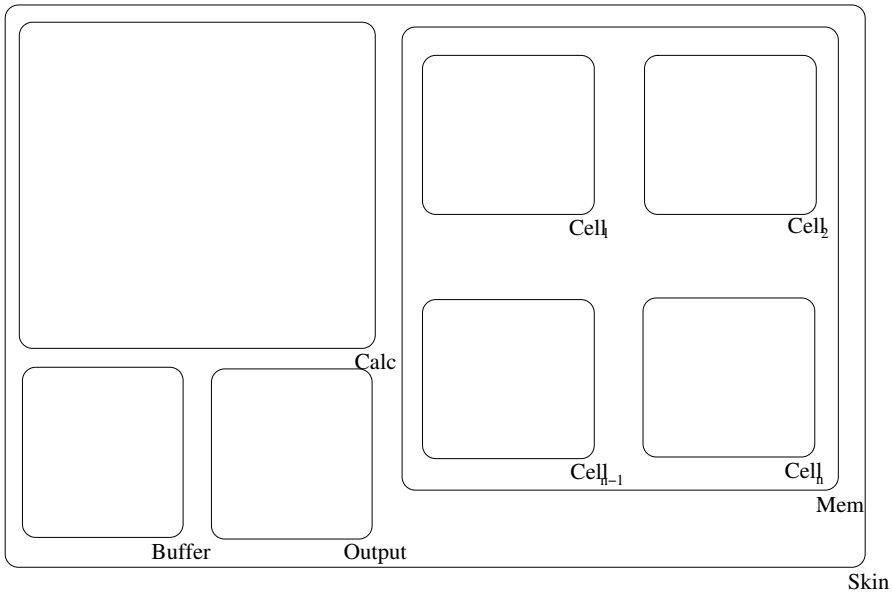


Figure 1: The general structure of the storage membrane system.

3 The Structure of the Storage Device

As previously said, the main idea is to define a system which allows to store information of already executed computation, in order to save computational time. Hence, the final system (we will call it the Storage Device) is obtained by adding to an original computational system (i.e. a standard P System designed to solve a computational problem), a memory sub-system, an output membrane, and a buffer membrane (to store information needed during the computation). All these components are then surrounded by a skin membrane.

In this section, we present the main components of the storage device, and then we formally define it. The detailed functioning of the system will be described in the next section.

The general structure of the storage device is represented in figure 1.

As one can see, there are four main sub-systems:

The **computational sub-system** consists of a standard P system that executes computations on the input strings that are injected from outside. This system does not have an output membrane, but it sends out the output strings to its environment that is the skin membrane of the storage device.

The **memory sub-system** consists of a certain number n of complex cells (a cell is a region with a complex structure). Each cell is used to store already computed solutions (and its corresponding input), and to produce them when the same input is injected again in the system at a later time.

The **buffer membrane** is used to store information that the system is actually computing. In particular, the buffer is used when the input string is not stored in the memory sub-system and, as a consequence, the solution has to be calculated by the computational sub-system. As said, at the end of the computation the storage device has to store the new computed solution and the input string used to compute it within the same memory cell. Inside the buffer there are two sub-membranes used to start up the operations needed to retrieve the string previously stored and to empty the buffer.

The **output membrane** collects the result of the computation, being it retrieved from the memory sub-system or calculated by the computational sub-system.

The computation in the system proceeds as it follows. The input string (injected through the skin membrane) is replicated to obtain three copies: a copy is sent to the computation sub-system, where it will begin a computation in order to calculate a new solution, while a second copy is sent into the buffer membrane. In parallel, the third copy is sent to the memory sub-system, where it is replicated and forwarded to all the memory cells, to check (in parallel in all memory cell) if its solution is already stored into the system.

If the solution is found in the memory sub-system, then it is sent to the output membrane and the system proceeds to empty the buffer and to stop the work in the computational sub-system. Otherwise, the computational sub-system produce the new solution, and sends it both to the output membrane and to the memory sub-system. The solution has to be stored in a memory membrane together with its relative input, that is found in the buffer membrane where the input was stored when the computation started. Thus, the system non-deterministically chooses an empty cell in the memory subsystem and sends into this membrane the two strings. The system is now ready to execute a new computation.

Formally, the system is defined as it follows:

$$DdM = (\mu, V, W_{Calc}, W_{Mem}, W_{Buf}, R_{Calc}, R_{Mem}, R_{Buf}, Output)$$

where:

1. $\mu = [_{Skin}\ [_{Output}\ ]_{Output}\ [_{Calc}\ ]_{Calc}\ [_{Buf}\ ]_{Buf}$
$[_{Mem}\ [_{Cell_1}\ ]_{Cell_1}\ \cdots\ [_{Cell_n}\ ]_{Cell_n}\ ]_{Mem}\ ]_{Skin}$
The internal structure of every cell belonging to Mem is:
$[_{Cell_i}\ [_{Count}\ ]_{Count}\ [_{Count\beta}\ ]_{Count\beta}\ [_{Sol}\ [_{Empty}\ ]_{Empty}\ [_{Double}\ ]_{Double}$
$]_{Sol}\ [_{Input}\ [_{Compare}\ ]_{Compare}\ [_{Syncro}\ [_{Empty}\ ]_{Empty}\ [_{Double}\ ]_{Double}$
$]_{Syncro}\ ]_{Input}\ ]_{Cell_i}$
for $1 \leq i \leq n$.

2. $V = \{a_1, \ldots, a_k, \alpha, \ldots, \omega, A, \$, \$', \$'', \$_P\Pi, X, X'\} \cup$
$\{Y, D, \langle, \langle', \langle'', \rangle, th, pol, per, Stop, F, \lambda, =, \neq, \neq'\}$
is the total alphabet of the system.

3. W_{Calc} is the family of strings belonging to the $Calc$ region.

4. W_{Buf} is the family of strings belonging to the $Buffer$ region, in the starting configuration this is empty.

5. W_{Mem} is the family of strings belonging to the Mem region, in the starting configuration, the only set belonging to W_{Mem} is $w_{Compare} = \{\neq\}$.

6. $R = \{R_{Skin}, R_{Mem}, R_{Cell_i}, R_{Sol}, R_{Double}, R_{Empty}, R_{Input}, R_{Syncro}, R_{Compare}\}$ is the family of rules of the system.

$R_{Skin} = \{r_1 : [_{Skin}\, X\,]_{Skin} \rightarrow [_{Skin}\, (X, in_{Mem})\ ||\ (X, in_{Buf})\ ||$
$(\Pi', in_{Calc})\,]_{Skin}$
$r_2 : [_{Skin}\, \$'\,]_{Skin} \rightarrow [_{Skin}\, (\lambda, in_{Out})\ ||\ (\$', in_{Buf})\ ||\ (Stop, here)\,]_{Skin}$
$r_3 : [_{Skin}\, \$_P\,]_{Skin} \rightarrow [_{Skin}(\$, in_{Buf})\, ||\, (\lambda, in_{Out})\, ||\, (\$, in_{Mem})\ ||$
$(Stop, here)]_{Skin}$
$r_4 : \langle\ [_{Mem}\ ]_{Mem} \rightarrow [_{Mem}\ \langle\ ]_{Mem}\}$
r_1 replicates and forwards the input string to the Mem, Buffer and Calc membranes at the beginning of each computation.
r_2, r_3 are used to forward the solution string to the Output, Buffer and Mem membranes at the end of each computation.
r_4 sends the input string to the Mem membrane, to store it in the system.

$R_{Mem} = \{r_1 :\ [_{Mem}\ \$\ ]_{Mem} \rightarrow [_{Mem}\ \$^-\ ]_{Mem}$
$r_2 : [_{Mem}\, X\,]_{Mem} \rightarrow [_{Mem}\, (X, in_{Cell_1})\ ||\ (X, in_{Cell_2})\ ||\ldots||$
$(X, in_{Cell_n})]_{Mem}$

$r_3:\ [_{Mem}\ \$'\ ]_{Mem} \rightarrow [_{Mem}\ ]_{Mem}\ \$'$
$r_4:\ [_{Mem}\ \langle\ ]_{Mem} \rightarrow [_{Mem}\ \langle^{+}\ ]_{Mem}\}$
r_1, r_4 are used to send non-deterministically the input string and the corresponding solution into an empty cell to be stored in the system.
r_2 is used to forward the input string that comes from the environment to all the memory cell in order to start the comparing procedure.
r_3 when a solution is found in memory, this rule is used to send it to the skin membrane, where it will then be sent to the output membrane.

$R_{Cell_i} = \{r_1:\ [_{Cell_i}\ \langle\ ]^{-}_{Cell_i} \rightarrow [_{Cell_i}\ [_{Input}\langle''\ ]_{Input}\ ]^{0}_{Cell_i}$
$r_2:\ [_{Cell_i}\ pol\ ]_{Cell_i} \rightarrow [_{Cell_i}]^{+}_{Cell_i}\ th$
$r_3:\ X \rightarrow [_{Input}\ X\ ]_{Input}$
$r_4:\ [_{Cell_i}\ \$'\ ]_{Cell_i} \rightarrow [_{Cell_i}\ ]_{Cell_i}\ \$'$
$r_5:\ [_{Cell_i}\ F\ ]_{Cell_i} \rightarrow [_{Cell_i}\ (F, in_{Sol})\ ||\ (F, in_{Input})\ ]_{Cell_i}$
$r_6:\ [_{Cell_i}\ \$\ ]^{+}_{Cell_i} \rightarrow [_{Cell_i}\$'']^{-}_{Cell_i}$
$r_7:\ [_{Cell_i}\ \$'']_{Cell_i} \rightarrow [_{Cell_i}(\$, in_{Sol})\ ||\ (=, in_{count})]_{Cell_i}\}$
r_1, r_6, r_7 are used to store informations into the sub membranes of the cell.
r_2, r_5 are used to empty the cell from solutions which become too old.
r_3 sends the input string to compare into the Input sub-membrane.
r_4 is used to sent out the solution.

$R_{Sol} = \{r_1:\ [_{Sol}\ \$\ ]_{Sol} \rightarrow [_{Sol}\ \$^{-}\ ]_{Sol}$
$r_2 :=\ [_{Double}\ ]^{0}_{Double} \rightarrow [_{Double}\ th\ ]^{+}_{Double}$
$r_3:\ [_{Sol}\ th\ ]_{Sol} \rightarrow [_{Sol}\ ]_{Sol}\ th$
$r_4:\ F[_{Empty}\ ]^{0}_{Empty} \rightarrow [_{Empty}\ th\ ]^{+}_{Empty}$
$r_5:\ [_{Sol}\ \$'\ ]_{Sol} \rightarrow [_{Sol}\ ]_{Sol}\ \$'$
$r_6:\ [_{Sol}\ pol\ ]_{Sol} \rightarrow [_{Sol}\ ]_{Sol}\ pol\}$
r_1, r_2, r_5 are used to duplicate the solution and to send out one copy.
r_4, r_6 are used to empty the cell.

$R_{Double} = \{r_1:\ [_{Double}\ th\ ]_{Double} \rightarrow [_{Double}\ ]_{Double}\ th$
$r_2:\ [_{Double}\ \$\ ]_{Double} \rightarrow [_{Double}\ (\$', here)\ ||\ (\$'', here)\ ]_{Double}$
$r_3:\ [_{Double}\ \$'\ ]_{Double} \rightarrow [_{Double}\ ]_{Double}\ \$$
$r_4:\ [_{Double}\ \$''\ ]^{+}_{Double} \rightarrow [_{Double}\ ]^{0}_{Double}\ \$'$
$r_5:\ [_{Double}\ X'\ ]_{Double} \rightarrow [_{Double}\ X\ ]_{Double}$
$r_6:\ [_{Double}X\ ]^{+}_{Double} \rightarrow [_{Double}\ ]^{0}_{Double}\ X$

$r_7 :\ [_{Double}\ \langle\ ]_{Double} \rightarrow [_{Double}\ (\langle, out)\ ||\ (\langle', out)\ ]_{Double}\}$
r_1, r_2, r_3, r_4 are used to duplicate the solution string.
r_5, r_6, r_7 are used to duplicate the input string stored within a cell.

$R_{Empty} = \{r_1 :\ [_{Empty}\$\ ]^{+}_{Empty} \rightarrow [_{Empty}]^{0}_{Empty}\ pol$
$r_2 :\ [_{Empty}\langle\ ]^{+}_{Empty} \rightarrow [_{Empty}]^{0}_{Empty}\ th\}$
r_1, r_2 are used to delete respectively solution and input strings.

$R_{Input} = \{r_1 : [_{Input}\ X'\ [_{Comp.}]^{+}_{Comp.}\ ]_{Input} \rightarrow [_{Input}\ [_{Comp.}\ X'^{-}\ ]^{+}_{Comp.}\ ]_{Input}$
$r_2 :\ [_{Input}\ X'\ [_{Compare}\ ]^{0}_{Compare}\ ]_{Input} \rightarrow [_{Input}\ [_{Compare}\ th\ ]^{0}_{Compare}\ ]_{Input}$
$r_3 :\ X\alpha\ [_{Compare}\ ]_{Compare} \rightarrow [_{Compare} X\alpha\ ]_{Compare}\ \forall\ \alpha \in\ V$
$r_4 :\ \langle\alpha\ [_{Compare}\ ]_{Compare} \rightarrow [_{Compare}\langle\alpha\ ]_{Compare}\ \forall\ \alpha \in\ V$
$r_5 :\ XY\ [_{Compare}\ ]_{Compare} \rightarrow [_{Compare} XY\ ]_{Compare}$
$r_6 :\ \langle\rangle\ [_{Compare}\ ]_{Compare} \rightarrow [_{Compare}\langle\rangle\ ]_{Compare}$
$r_7 :\ [_{Input} \neq\ ]_{Input} \rightarrow [_{Input}\ ]_{Input}\ th\ \tau$
$r_8 :\ [_{Input}\ X\ ]_{Input} \rightarrow [_{Input}\ ]_{Input}\ th$
$r_9 : \neq'\ \ [_{Compare}\ ]_{Compare} \rightarrow [_{Compare}\ th\ ]_{Compare}$
$r_{10} :\ [_{Input}\ D\ ]_{Input} \rightarrow [_{Input}\ ]_{Input}\ th\ \delta$
$r_{11} :\ [_{Input}\ th\ ]_{Input} \rightarrow [_{Input}\ ]_{Input}\ th$
$r_{12} :\ F\ [_{Syncro}]^{+}_{Syncro} \rightarrow [_{Syncro}\ F\ ]^{0}_{Syncro}$
$r_{13} :\ [_{Input}\ pol\ ]_{Input} \rightarrow [_{Input}\ ]_{Input}\ pol$
$r_{14} :\ \langle''\ [_{Syncro}]^{0}_{Syncro} \rightarrow [_{Syncro}\ \langle\ ]^{+}_{Syncro}\}$
r_1 is used to send the input string into Syncro membrane only if the cell actually stores data.
r_2 is used to delete the input string if the cell is empty.
$r_3 - r_{11}$ (plus some other rules not specified here used for technical reasons) are used to compare the input string that comes from the environment with the string stored into the cell.
r_{12}, r_{13} are used to empty the cell.
r_{14} is used to store the input string into the cell.

$R_{Syncro} = \{r_1 :\ [_{Syncro}\ \langle\ ]_{Syncro} \rightarrow [_{Syncro}\ \langle^{-}\ ]_{Syncro}$
$r_2 :\ X'\ [_{Double}]_{Double} \rightarrow [_{Double} X'\ ]^{+}_{Double}$
$r_3 :\ [_{Syncro}\ th\ ]_{Syncro} \rightarrow [_{Syncro}\ ]_{Syncro}\ th$
$r_4 :\ [_{Syncro}\ X\ ]_{Syncro} \rightarrow [_{Syncro}\ ]_{Syncro}\ X$
$r_5 :\ [_{Syncro}\ \langle'\ ]_{Syncro} \rightarrow [_{Syncro}\ ]_{Syncro}\ \langle$
$r_6 :\ F\ [_{Svuota}]^{0}_{Svuota} \rightarrow [_{Svuota}\ th\ ]^{+}_{Svuota}\}$

r_1, r_2 are used to activate the duplicating procedure of the input string.
r_4, r_5 are used to begin the comparison procedure.
r_6 is used to empty the cell.

$R_{Compare} = \{r_1 : [_{Compare}\ X\ Y\ ]_{Compare} \rightarrow [_{Compare}\ X\ z\ ]_{Compare}$
$r_2 : [_{Compare}\ X\ \alpha_n]_{Compare} \rightarrow [_{Compare}\ X\ \alpha_{n-1}]_{Compare}$
$r_3 : [_{Compare}\ \langle\ \rangle\ ]_{Compare} \rightarrow [_{Compare}\ \langle\ z\ ]_{Compare}$
$r_4 : [_{Compare}\ \langle\ \alpha_n]_{Compare} \rightarrow [_{Compare}\ \langle\ \alpha_{n-1}]_{Compare}$
$\forall\ \alpha\ \in\ V\ and\ n \geq 2$
$r_5 : [_{Compare}\ X\ a\ ]_{Compare} \rightarrow [_{Compare}\ ]_{Compare} X \lambda\ \tau$
$r_6 : [_{Compare}\ \langle\ a\ ]_{Compare} \rightarrow [_{Compare}\ ]_{Compare} \langle \lambda\ \delta$
$r_7 : [_{Compare}\ \langle\ ]_{Compare} \rightarrow [_{Compare}\ per\ ]_{Compare}$
$r_8 : [_{Compare}\ per\ ]_{Compare} \rightarrow [_{Compare}\ ]_{Compare} th\ \delta\ \}$
The comparison of the string is performed in a left to right manner: the leftmost symbols of the two string are compared. If they are different, then a negative answer is returned. Otherwise, both symbols are deleted from their respective strings and the comparison proceed with the (new) leftmost symbols. If all symbols are deleted from both strings, then a positive answer is returned.
$r_1 - r_4$ are used to decrease the value of the first character of the input strings.
r_5, r_6 are used to delete the first character of the input strings.
r_7, r_8 are used to end the comparison algorithm.

7. *Output* is the output membrane.

4 Functioning of the Storage Device

The computation of the system starts when an input string is injected from the environment to the skin region. As previously said, the input string is replicated in three copies: the first copy is sent to the computation subsystem, to start a computation to calculate a new solution; the second copy is sent into the buffer membrane, to be used eventually later to store the string and its relative solution in a memory cell. The third copy is sent to the memory sub-system, where it is replicated in n copies to be forwarded to all memory cells, to check (in parallel in all memory cell) if the corresponding solution is already stored into the system.

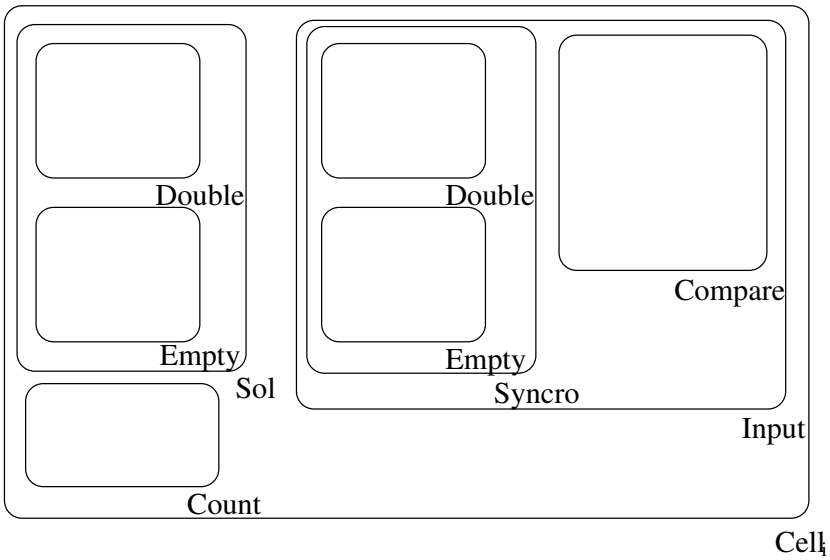


Figure 2: The structure of a memory cell

When the solution string is already stored in memory, it is retrieved from the memory cell and then sent to the output membrane. The system proceeds to empty the buffer membrane and the computational sub-system work is stopped, as it is no more necessary. On the contrary, if the solution is not stored in the system, then the computational sub-system produce the new solution, that is then sent both to the output membrane and to the memory sub-system. The solution has to be stored in a memory membrane together with its relative input string; this last string is thus retrieved from the buffer membrane, where it was stored when the computation started. The system chooses, in a non-deterministic way, an empty cell in the memory subsystem, and sends into this membrane both the input and the corresponding solution strings.

In the following, we will describe the internal structure of a memory cell and (by means of pseudo-code) the tasks executed inside every one of this. The structure of a cell included inside the memory of the system is represented in figure 2.

All of the cells have got different labels but have the same internal structure and the same starting configuration. This is the reason why is enough to describe only one of these. All of the cells are made up of three parts: two of these are used to store the input string and the solution string, the third part

manage the counter of the cell. In the starting configuration of the system all of the cells have a positive electrical charge, but the sub membranes of the cells have null electrical charge. The polarization is used to distinguish the empty cells from the others (the cells that contain some informations). This distinction is useful when the system has to store a new solution, with the different polarization of the cell, the system is able to select an empty cell. It is clear that an empty cell has a positive charge while a taken cell has a null charge.

When the system stores an information inside a cell, it changes the electrical charge of this (from positive to null) and modifies the polarization of the sub membrane (from null to positive) where the input string is put, this value will be used by the system to detect the cells that contains informations when is needed to compare the input string that comes from the environment with the string stored in the cell.

Inside the membrane where the input string is stored, there is a membrane used to compare the strings. The comparison is carried out by synchronizing the entrance and the exit from this membrane of the strings the algorithm is comparing. The strings come into the membrane at the same time, then, inside it the rules that reduce the character value until deleting it are applied. It is clear that, if two characters are equal they will be deleted at the same time, but if they are different, the deleting rules will be applied in different moments. During the deleting operations, the strings are also sent out of the comparison membrane and, while going out, the applied rules perform a δ and τ operation respectively. In this way, if the strings leave the membrane at the same moment, the operations are executed at the same computation step and their effect is avoided. On the other hand, applying one rule before the other one, it is impossible to carry on the algorithm. This is due to the thickness change, because the membrane is dissolved or its permeability is increased and the strings are not able to pass through the membrane.

A part from the structure that realizes the algorithm, there are two membranes that are able to duplicate and to erase data, this membranes are used only when the system explicitly needs to execute this operations. The second region of the cell is used to store the solution strings, also inside this membrane there are two sub membranes: one is used to duplicate solution strings when the comparison algorithm successfully ends and the second to delete it when the system needs to empty the cell.

The last part of the cell is made of two membrane used to manage the counter of the cell. The counter is used to establish when an information is obsolete. We suppose that too old data has got a low probability of being

demanded a second time. Moreover, holding obsolete data increases the risk that the memory is filled up completely. The counter starts to count the life time of a solution, when this is stored inside a cell, at every computational step, the counter value is decreased.

We have to set the starting value of the counter over a established threshold, so that the solution inside a cell is not being deleted too much faster; the aim is to store data as long as possible until definitively deleting it.

Now, we will show the three procedure in order to insert, delete and retrieve the informations inside a cell, moreover, we will describe the comparison algorithm of the strings.

Procedure ***insert*** *(input,solution)*
begin
if *memory is full (no memory cell with positive polarization are available)* ***then***
delete*(F);*
Choose non deterministically a cell among those with positive charge;
Send into this cell the solution string, changing the charge of the cell to negative polarization;
Send the input string in the (unique) membrane with negative polarization, changing the charge of the cell to neutral;

Send the solution string in sub-membrane Sol;
Send then input string in sub-membrane Input and then in sub-membrane Syncro;
Change membrane Syncro charge from 0 to +;
end.

The *insert* procedure starts when the storage device needs to store a new solution. If no free cells are available (no cells with positive polarization are present), then the system proceed to free some cells by means of the *delete* procedure (see description below). When one or more free cells are available, the solution string and its related input string are non-deterministically sent inside one of those empty cells; first, the solution string is sent through the chosen membrane, changing its charge from positive to negative. Then, the input string (that is still inside the general memory region) is sent in the same membrane, that can be identified as it is the only one with negative charge. During this passage, the polarization of the membrane is changed to neutral, to denote this cell as non-empty.

Then, the solution string is sent to the internal sub-membrane *Sol*, while the input string is sent to the internal sub-membrane *Input* and, from here,

in the inner region delimited by membrane *Syncro*. During the last passage, the electrical charge of the membrane *Syncro* is changed from neutral to positive (this will be necessary when the solution will be searched for in a following computation).

After the end of the procedure, the cell begins to count the solution life time.

Procedure ***delete*** *(F)*
begin
Send F in sub-membranes Sol and Input;
Send F in sub-membrane Empty;
Change membrane Empty charge from 0 to +;
Send solution in Empty;
Send input in Empty;
Delete solution;
Delete input;
end.

The *delete* procedure is activated when the memory is full or when a solution stored into a cell is too old to be held. In the first case, when all the cells are full and the system needs to store a new solution of the actual computation, then it has to choose a cell and activate from the outside the deleting procedure. The cell that the system has to empty is non-deterministically chosen, sending inside of it a message marked with the special symbol *F*. In the second case, when the counter of a cell reaches its final value, the F message is created within the cell.

The *delete* procedure begins when the message marked with F is sent or is created inside a memory cell. The first operation forwards the message inside the membranes *Sol* and *Input* (this is achieved using a replicated rewriting rule). Now within membranes *Sol* and *Input*, concurrently, the string message is sent to the membrane labelled with *Empty*, that is a sub membrane belonging to both membranes (*Sol* and *Input*). When the string passes through the membrane, it changes its polarization from null to positive. Now, inside the membranes *Sol* and *Input* it is possible to apply rules to send both strings into the two membranes *Empty*. Inside the two membranes *Empty,* it is now possible to apply the appropriated rules which delete the input and solution strings.

When the procedure is done, the storage device sends out to the environment all the strings that are no longer useful to the system.

Procedure ***retrieve*** *(inputE)*

```
begin
    if cell contains data then
      Send inputE in Syncro
    Duplicate input;
    Send out input and inputE;
    if Comparison(input,inputE) = TRUE then
    begin
      Duplicate solution;
      Send out solution;
    end
end.
```

The *retrieve* procedure is used to search inside the memory cell the solution of the actual computation.

In the beginning of every computation, the input string that comes from the environment (in the pseudo code of the procedure, this string is called *inputE*) is sent into every cell of the system memory. When this string reaches every cell, the *retrieve* procedure starts. This procedure is executed in parallel in every cell.

The first operation is to check if the cell contains data. If this is true, then the input string coming from the environment is sent inside the sub membrane labelled with *Syncro*; otherwise the procedure ends. If the memory cell contains information, then the procedure goes on duplicating the stored input string. Now, in the same computational step, the input string actually stored into the cell and the other one that comes from the environment, are sent out from the membrane *Syncro*; here the comparison algorithm is started. If the comparison procedure successfully ends, then the solution string is duplicated and sent out, otherwise the procedure finishes. The comparison algorithm is the following:

```
Procedure comparison(inputM,inputE)
begin
   while (length(InputM) > 0 AND length(InputE) > 0)
   begin
     Send inputM,inputE in Compare;
     while first char value(InputM) ≠ a1 AND
           first char value(InputE) ≠ a1
       Decrease first char value;
     if first char value(InputM) = a1 AND first char value(InputE) = a1
       Delete first char;
```

```
      Send input,inputE out;
    else return FALSE;
  end
  if (length(InputM) = 0 AND length(InputE) = 0) then
    return TRUE
  else return FALSE

end.
```

The comparison algorithm is the main task executed by the storage device: it is used to compare the input string that comes from the environment (*InputE*) with the string contained within a cell (*InputM*). As the retrieve procedure, the comparison algorithm is executed in parallel in all memory cells.

The algorithm compares every single character of the two strings and, in case all these characters are equal, returns a positive value, otherwise it fails. The procedure to check whether the two strings are equal is realized with the synchronized entrance and exit of the strings through a special membrane. The procedure begins by sending the strings inside the compare membrane at the same computational step, only if their length is greater then zero. Now, within the compare membrane, some rules are activated that operate on the first character of the strings. These rules decrease the character value (we assume the alphabet is sorted in increasing order from a_1 to a_n) until it reaches the symbol a_1 (that is the first character belonging to V).

It is now possible to use the rule that deletes the symbol a_1 and sends the strings to the outside region; then the procedure will start to analyze the second character of the strings. The procedure works on the two strings at the same time, as it needs a method to understand if the actual leftmost characters of the two strings are equal. This is done exploiting the rules that send out the strings from the compare membrane: when the system executes the rules in order to delete the first character of the strings, δ and τ operations are executed respectively by the rule that sends out the string *InputM* and the string *InputE*.

It is clear that if the two rules are concurrently applied, then the effect of the δ and τ operations is null and nothing happens to the membrane. On the contrary, if the two characters that the system is comparing are different, then the two rules used to send out the strings are applied at different computational steps (due to the different number of rules executed to decrease the value of the first character of the two strings). Applying one of these rules before the other one means to use a δ or τ operation before the

other one. It is clear that the resultant effect is to dissolve or to increase the thickness of the compare membrane. In both cases, it is impossible to carry on the comparison algorithm and the system determines that the strings are not equal.

For instance, if the system is comparing the strings $s_1 = a_3a_2a_2a_1$ and $s_2 = a_3a_2a_2a_1$, first sends concurrently s_1 and s_2 into the compare membrane, then within it are applied rules like $a_3 \rightarrow (a_2, here)$ on both strings. Now $s_1 = a_2a_2a_2a_1$ and $s_2 = a_2a_2a_2a_1$, the procedure goes on applying rules as: $a_2 \rightarrow (a_1, here)$ and then $s_1 = a_1a_2a_2a_1$ and $s_2 = a_1a_2a_2a_1$. The analysis of the first character ends with the concurrent execution of the rule $a_1 \rightarrow (\lambda, out)\delta$ on the first string and $a_1 \rightarrow (\lambda, out)\tau$ on the other string. The effect of the δ and τ operation is then avoided and the system understands that the two characters are equal and can go on comparing the remaining characters of the strings. Now $s_1 = a_2a_2a_1$ and $s_2 = a_2a_2a_1$, the algorithm continues analyzing the actual first character. When all characters have been deleted, the algorithm returns a true value.

Let us denote by $T_m(n)$ the computational time of the P system with memory and by $T_P(n)$ the computational time of the original computational system (that is equal to the computational sub-system) and by $T_c(n)$ the time required to compare an input in a memory cell. From the previous description, it is easy to see that the system with memory works with the following computational time:

- If the solution is not stored in memory, then the time required by the systems is the time required by the original computational system plus a (constant) time to store the new solution. Thus, $T_m(n) = O(T_p(n))$.
- If solution is already stored in memory, then the time required by the system with memory is the faster time between the time to calculate the new solution and the time to retrieve that solution from memory; thus, $T_m(n) = min\{T_P(n), T_c(n)\}$. It is easy to see that $T_c(n) = O(n)$, as the time required to compare the input and the string stored in memory is linear. Hence, we can conclude that, when a solution is stored in memory, $T_m(n) = O(n)$.

Thus, the proposed system does require a constant time to store new solutions, when the input is not (anymore) known, while it allows to cut down computational time to a linear one, when the input is stored in memory. Thus, such systems could be effectively used when many computations are requested where the same input is submitted many times with a high probability.

5 Conclusions

We presented P systems in which solutions of already executed computations can be stored in memory cells, which are added to a standard computing system. Each time a solution is produced by the computing sub-system, the solution is stored in a memory membrane. When a new input is inserted in the system to start a new computation, we start in parallel the search for the solution in all memory membranes and, at the same time, the computation on that input in the computing sub-system.

If the solution is found in one memory cell before the computation ends, then a copy of the solution is expelled from that memory membrane; the search in all other memory membranes is stopped, and the same is done with the computing sub-system. In case the solution is not found in any memory membrane, then the computing sub-system produces the new solution, which is stored in a free memory membrane. In order to keep at least one memory membrane available all the time, oldest solutions are deleted from memory when necessary. Such system can be used to speed-up computations, when the output for the same input is requested many times with a high probability within a short time, such as image and vocal processing applications.

We point out that this is a preliminary approach, which could be developed following different lines; as an example, we could consider the possibility to store not only solutions of already executed computations, but also to store programs.

An issue related to the system concerns the dimension of the memory, i.e. the number of memory cells to use. In the solution we propose here, the memory cells are static. The number of cells has a direct influence on the efficiency of the computations. A small number of cells reduce the probability to find a solution in memory, and require further computational step to freeing memory cell and store new solution. One possible enhancement of the system could be to consider dynamic memory cells, which can be added and removed during the computation. This could be accomplished by means of membrane division feature (see [5]) and membrane dissolving feature (already considered in this system).

Acknowledgements

We wish to thank the anonymous referees for their helpful suggestions, that allowed us to improve a previous version of the present paper.

References

[1] V. Manca, C. Martin-Vide, Gh. Păun: On the Power of P Systems with Replicated Rewriting. *J. Automata, Languages, and Combinatorics* **6** (3) (2001), 359–374.

[2] C. Martin-Vide, Gh. Păun: String Objects in P Systems. *Proc. of Algebraic Systems, Formal Languages and Computations Workshop*, Kyoto, 2000, RIMS Kokyuroku, Kyoto Univ. (2000), 161–169.

[3] Gh. Păun: Computing with membranes. *J. of Computer and System Sciences* **61** (1) (2000), 108–143 and *TUCS Research Report No 208*, November 1998 (http://www.tucs.fi).

[4] Gh. Păun: Computing with membranes – A variant: P systems with polarized membranes. *Intern. J. of Foundations of Computer Science* **11** (1) (2000), 167–182, and *Auckland University, CDMTCS Report No 098* (1999) (www.cs.auckland. ac.nz/CDMTCS).

[5] Gh. Păun: Computing with Membranes: Attacking NP-Complete Problems. In: I. Antoniou, C.S. Calude, M.J. Dinneen (Eds.): *Unconventional Models of Computation*. Springer-Verlag, London (2000), 94–115.

[6] Gh. Păun: P Systems with Active Membranes: Attacking NP-Complete Problems. *J. Automata, Languages and Combinatorics* **6** (1) (2001), 75–90, and CDMTCS TR 102, Univ. of Auckland (1999) (www.cs. auckland. ac.nz/CDMTCS).

[7] Gh. Păun: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin (2002).

[8] Gh. Păun, G. Rozenberg, A. Salomaa: Membrane computing with external output. *Fundamenta Informaticae* **41** (3) (2000), 259–266, and *Turku Center for Computer Science-TUCS Report No 218* (1998) (www.tucs.fi).

[9] I. Petre: A normal form for P systems. *Bulletin of EATCS* **67** (1999), 165–172.

[10] G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*. Springer-Verlag, Heidelberg (1997).

[11] C. Zandron, C. Ferretti, G. Mauri: Two Mormal Forms for Rewriting P Systems. In: M. Margenstern, Y. Rogozhin (Eds.): Proc. Third. Intern. Conf. on Universal Machines and Computations, Chişinau, Moldova, *Lecture Notes in Computer Science*, **2055**, Springer-Verlag, Berlin (2001), 153–164.

[12] C. Zandron, C. Ferretti, G. Mauri: Using Membrane Features in P Systems. *Rom. Journ. of Information Science and Technology* **4** (1-2) (2001), 241–257.

On Picture Arrays Generated by P Systems

P. Helen CHANDRA[1], K.G. SUBRAMANIAN[2]

[1]Department of Mathematics
Jayaraj Annapackiam College for Women
Periyakulam, Theni 625 601 India

[2]Department of Mathematics
Madras Christian College
Chennai 600 059 India

Abstract

Array-rewriting P Systems with array objects and array-rewriting rules have been considered in Ceterchi et al [1] extending the notion of P Systems with String objects. Motivated by the study in [5], in this note, we consider L-type rules in the regions of these P systems. The resulting P system, called L-array P System, is examined for its generative power.

1 Introduction

Bringing together the two areas of membrane computing and Picture grammars, Ceterchi et al [1] introduced and studied the power of array-rewriting P systems extending the notion of P Systems with string objects to P systems with array objects with the arrays in the regions being processed by array-rewriting rules. On the other hand array grammars with controlled rewriting have been considered in [5]. The model in [5] generates rectangular arrays by rewriting in parallel the symbols in the edges of an array by L type string rules. Motivated by this study, in this note, we consider L-type rules in the regions of the array-rewriting P system of [1]. We call the resulting system as L-array P System. A beginning is made here in studying the picture array generative power of these P systems.

2 Preliminaries

Let Σ be a finite alphabet. We consider pictures over Σ as rectangular arrays of elements of Σ. The set of all pictures over Σ is denoted by Σ^{**}. The size of a picture p is a pair (m, n) where m is the number of rows of p and n, the number of columns. An empty picture is the only picture of size $(0, 0)$. A picture language L is a subset of Σ^{**}.

We refer to [5] for the notion of a table L array grammar generating pictures of rectangular arrays. We briefly mention only needed details of these grammars.

The classes of array generating devices that are considered in [5] are
i) Table $0L(1L)$ array grammars $(T0L(1L)AG)$
ii) Extended Table $0L(1L)$ array grammars $(ET0L(1L)AG)$
iii) Extended Controlled Table $0L(1L)$ array grammars $(ECT0L(1L)AG)$
Each of these array grammars consists of tables of rules which are $0L$ or $1L$. A $0L$ rule is of the form $a \to \alpha$, $a \in \Sigma$, $\alpha \in \Sigma^*$, as in a string $0L$ system. When the table is a right table, the rules $a_i \to \alpha_i \, (i = 1, \cdots, n) \, (n \geq 1)$ of the table are such that the lengths of all such α_i are equal and the rules can be used to rewrite in parallel all the symbols in the rightmost column of a picture array. A $1L$ rule is of the form

$$\begin{array}{ccc} & b & \\ c & a & \\ & d & \end{array} \quad \to \quad \begin{array}{cc} & b \\ c & \alpha \\ & d \end{array}$$

where b, c, d are the symbols giving the context in which a can be rewritten as α, with all such α having the same length in the rules of a right table. A right table of $1L$ rules is used similar to a right table of $0L$ rules. Likewise, a table can be a left table, up table or down table and used in rewriting all the symbols in the leftmost column, uppermost row or lowermost row respectively, of a picture array. Unlike string L systems in which the rewriting is totally parallel, here only all the symbols in the or leftmost column or in the uppermost or lowermost row are rewritten depending on the table of rules used.

As in string L systems, if we require the arrays generated to be over a terminal alphabet, then we have the extended feature. If the application of the tables is controlled by a control set, which can be regular, CF, CS, then we have the controlled feature in the array grammar.

The picture language generated by an array grammar of the type mentioned above consists of rectangular arrays over the (terminal) alphabet Σ, derived from a start symbol in the extended case, and an axiom array

in the non-extended case, (with the sequence of application of the tables of rules according to the words of a control set, in the case of controlled feature).

3 Array Generation and P Systems

In this section we consider array-rewriting P systems as defined in [1] but with a difference that the objects in the regions are rectangular arrays and the "rules" are tables of $0L$ or $1L$ rules as described in the previous section, with each table having an attached target $here, out, in$ with the usual meaning. We call the resulting P system an $L-$array P system. We consider halting computations as in array-rewriting P systems [1], resulting in the rectangular picture arrays collected in the output (an elementary) membrane.

Definition 1. An L-array P system of degree $m \geq 1$ is a construct

$$\Pi = (V, \Sigma, \#, \mu, A_1, \cdots, A_m, R_1, \cdots, R_m, i_0)$$

where V is the total alphabet, $\Sigma \subseteq V$ is the terminal alphabet, $\#$ is the blank symbol, μ is a membrane structure with m membranes labelled in a one-to-one manner with $1, 2, \cdots, m$; $A_i (i = 1, \cdots, m)$ is a finite set of picture arrays over V associated with the m regions of μ; $R_i = \{xt_j / 1 \leq j \leq n, n \geq 1\}$; xt_j is a right, left, up or down table according as $x = r, l, u$ or d; the tables have attached targets $here, out, in$ ($here$, is in general, omitted); the rules of a table are either $0L$ or $1L$ type; i_0 is the label of an elementary membrane of μ, called the output membrane. A table of rules in a region is chosen non-deterministically. The application of the rules of a right table to the rightmost column of an array is done by choosing the rules non-deterministically and rewriting all the symbols of the rightmost column in parallel. If the target indication of the table is out, then array is sent to the immediately outer membrane and if it is in, to the immediately inner membrane. Likewise the application of a left, up, down table is done. The arrays generated are captured in an elementary membrane with halting computation.

We illustrate the definitions with some examples.

Example 1. Consider the non-extended L-array P system

$$\Pi_1 = (\{x,.\},\{x,.\},\#,[_1\,[_2\,[_3\,]_3\,]_2\,]_1,\{xx\},\phi,\phi,\{ut_1\},\{rt_2,rt_3\},\phi,3)$$

$$ut_1 = \left\{ \begin{array}{l} \# \; x \; x \;\; \rightarrow \;\; \# \; \overset{.}{x} \; x \;\; , \;\; x \; x \; \# \;\; \rightarrow \;\; \overset{x}{x} \; x \; \# \;\; , \\ \\ \# \; . \; x \;\; \rightarrow \;\; \# \; \overset{.}{.} \; x \;\; , \;\; . \; x \; x \;\; \rightarrow \;\; \overset{.}{.} \; x \; x \;\; , \\ \\ \# \; . \; . \;\; \rightarrow \;\; \# \; \overset{.}{.} \; . \;\; , \;\; . \; . \; x \;\; \rightarrow \;\; . \; \overset{.}{.} \; x \;\; , \;\; . \; . \; . \;\; \rightarrow \;\; \overset{.}{.} \; . \; . \end{array} \right\} (in)$$

$$rt_2 = \{x \rightarrow xx\}(out), \quad rt_3 = \{x \rightarrow x\}(in)$$

Starting from the unique array xx present initially in region 1, the rules of the up table ut_1 are applied to the uppermost row in parallel to yield $\begin{matrix} . & x \\ x & x \end{matrix}$ and the array is sent to region 2; note that the table ut_1 is in region 1 ; the tables rt_2 and rt_3 are in region 2; there is no table of rules in region 3; if the rule of the right table rt_3 is applied in parallel to the rightmost column the array is unchanged but enters region 3 where it remains for ever and the computation stops; if the rules of the right table rt_2 are applied to the rightmost column, one column of $x's$ is grown to the right and the array is sent back to region 1; the computation continues. A picture array generated is shown in Figure 1. Interpreting . as 'blank', the picture array describes a digitized right triangle of symbols x.

$$\begin{matrix} . & . & . & . & x \\ . & . & . & x & x \\ . & . & x & x & x \\ . & x & x & x & x \\ x & x & x & x & x \end{matrix} \qquad\qquad \begin{matrix} & & & & x \\ & & & x & x \\ & & x & x & x \\ & x & x & x & x \\ x & x & x & x & x \end{matrix}$$

Figure 1 (a) Picture array of Example 1. (b) Right triangle of symbols x.

Example 2. Consider the non extended L-array P system Π_2 which has components as in Π_1 except that the initial array in region 1 is $xxxx$ and the tables of rules are

$$
\begin{array}{rcl}
ut_1 & = & \left\{ \begin{array}{ccccccccc}
 & & \begin{array}{c}.\\.\\.\end{array} & & & & \begin{array}{c}.\\.\\.\end{array} & & \\
 & , & & & & , & & & \\
\# \; x \; x & \rightarrow & \# \; x \; x & & x \; x \; x & \rightarrow & x \; x \; x & &
\end{array} \right. \\
 & & \begin{array}{ccccccccc}
 & & x & & & & . & & \\
 & & x & & & & . & & \\
 & , & x & & & , & . & , &
\end{array} \\
 & & \begin{array}{ccccccccc}
x \; x \; \# & \rightarrow & x \; x \; \# & & \# \; . \; . & \rightarrow & \# \; . \; . & &
\end{array} \\
 & & \left. \begin{array}{ccccccccc}
 & & . & & & & . & & . \\
 & & . & & & & . & & . \\
 & , & . & & & , & . & , & . \\
. \; . \; . & \rightarrow & . \; . \; . & & . \; . \; x & \rightarrow & . \; . \; x & . \; x \; x & \rightarrow \; . \; x \; x
\end{array} \right\} (in) \\
rt_2 & = & \left\{ \begin{array}{ccccccccc}
\# & & \# & & x & & x & x & \\
x & \rightarrow & x \; x \; x \; x & & x & \rightarrow & x \; . \; . \; . & x & \rightarrow \; x \; . \; . \; . \; , \\
x & & x & & x & & x & \# & \#
\end{array} \right. \\
 & & \begin{array}{ccccccccc}
x & & x & & x & & x & & \\
x & \rightarrow & x \; . \; . \; . \; , & . & \rightarrow & . \; . \; . \; . \; , & & &
\end{array} \\
 & & \left. \begin{array}{ccccccccc}
. & & . & & . & & . & & \\
. & & . & & . & & . & & \\
. & \rightarrow & . \; . \; . \; . \; , & . & \rightarrow & . \; . \; . \; . & & & \\
. & & . & & \# & & \# & &
\end{array} \right\} (out) \\
rt_3 & = & \{x \rightarrow x, \quad . \rightarrow .\}(in) &&&&&&
\end{array}
$$

A picture array generated by this P system is shown in Figure 2.

$$
\begin{array}{ccccccc}
. & . & . & . & . & . & x \\
. & . & . & . & . & . & x \\
. & . & . & . & . & . & x \\
. & . & . & x & x & x & x \\
. & . & . & x & . & . & . \\
. & . & . & x & . & . & . \\
x & x & x & x & . & . & .
\end{array}
$$

Figure 2 : Staircases of symbols x of fixed proportion.

Example 3. Consider the non-extended L-array P system Π_3 which has again components as in Π_1 except that the tables of rules are

$$
\begin{aligned}
ut_1 &= \left\{ \begin{array}{ccc} & & x \\ x & \rightarrow & x \end{array} \right\}(in) \\
rt_2 &= \{x \rightarrow xx\}(out) \\
rt_3 &= \{x \rightarrow x\}(in)
\end{aligned}
$$

This system generates square arrays of symbols x of all sizes (Figure 3).

$$
\begin{array}{cccc}
x & x & x & x \\
x & x & x & x \\
x & x & x & x \\
x & x & x & x
\end{array}
$$

Figure 3: A square array of symbols x.

We now compare the generative power of the L-array P systems with the non-extended Table Array grammars [5]and (Two-dimensional) Regular Matrix Grammars (RMG) [6,3].

Theorem 3.1 (i) Any picture array language generated by a non-extended Table Array grammar can be generated by a non-extended L-array P system of degree 1.

(ii) Any language generated by a RMG [6,3] is generated by an extended L-array P system of degree 1.

The results are immediate from the definitions of the respective grammars in view of the fact that i) in a non-extended Table Array Grammar every application of any table (starting from the axiom array) yields an array in the language generated and ii) in a RMG [6], in the first phase of derivation strings of intermediate symbols are generated by right-linear rules and hence suitable right tables of L-type rules can be included in the region; in the second phase these strings of intermediates are rewritten in parallel "vertically" to yield the terminal array and hence suitable down tables can also be included in the same region.

Theorem 3.2 (i) The classes of picture array languages generated by L-array P systems and Regular array grammars [4,7] intersect when restricted to rectangular picture array generation (see Example 3).

(ii) There is a rectangular picture array language generated by a L-array P system with $0L$ rules that cannot be generated by any RAG [4,7] (see Examples 1 and 2).

Theorem 3.3 The class of L-array P systems with tables of 0L rules is properly contained in the class of L-array P systems with tables of 1L rules. (The proper inclusion follows from examples 1 and 2.)

4 Conclusion

The rectangular picture array generating P systems, called L-array P Systems, considered here is a continuation of the study undertaken in [1] but uses a different type of rewriting in arrays and restricts only to rectangular pictures. Interesting examples of picture array languages generated by the L-array P Systems are shown in this note but the array languages of these examples can be generated by table array grammars with regular control. In fact connections between using regular control and P systems is brought out in [2]. It remains to explore connections of CF and CS controlled Table array grammars with P systems besides problems like hierarchy results based on the number of membranes and so on.

References

[1] R. Ceterchi, M. Mutyam, G. Păun, K.G. Subramanian: Array-rewriting P systems. *Natural Computing* **2** (2003), 229–249.

[2] K.S. Dersanambika, K. Krithivasan: Contextual array systems. *IJCM* **81** (2004), 955–969.

[3] R. Freund: Control Mechanisms on ♯-context-free array grammars. In: Gh. Păun (Ed.): *Mathematical Aspects of Formal and Natural Languages*. World scientific (1994), 97–137.

[4] D. Giammarresi, A. Restivo: Two-dimensional languages In: G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages,* Vol. 3 Springer Verlag (1997), 215 – 267.

[5] A. Rosenfeld: *Picture Languages (Formal Models for Picture recognition).* Academic Press (1979).

[6] G. Siromoney, R. Siromoney: Extended Controlled Table L-Arrays. *Information and Control* **35** (1977), 119–138.

[7] G. Siromoney, R. Siromoney, K. Krithivasan: Abstract families of matrices and picture languages. *Computer Graphics and Image Processing* **1** (1972), 234 – 307.

[8] P.S. Wang: Some new results on isotonic array grammars. *Information Processing Letters* **10** (1980), 129–131.

Algebraic and Coalgebraic Aspects of Membrane Computing

Gabriel CIOBANU[1], **Viorel Mihai GONTINEAC**[2]

[1]Institute of Computer
Science, Romanian Academy, Iaşi
and Research Institute "e-Austria"
Timişoara, Romania E-mail: gabriel@iit.tuiasi.ro

[2]Faculty of Mathematics
"A.I. Cuza" University of Iaşi,
E-mail: gonti@uaic.ro

Abstract

We introduce and study a new automaton able to consume and produce multisets. We are interested in their algebraic and coalgebraic properties. After some useful properties of multisets, we present the notions of bisimulation, observability and behaviour for Mealy multiset automata. We give a characterization of the bisimulation between two Mealy multiset automata, and a result relating their general behaviour to their sequential behaviour. We describe an endofunctor of the category of *Set* such that a Mealy multiset automaton is a coalgebra of this functor. This functor preserves coproducts, coequalizers, and weak pullbacks. Moreover, the new defined bisimulation is an instance of a more general coalgebraic definition.

1 Introduction

Membrane systems described in [11] represent bio-inspired abstract models. We try to connect membrane computing with the classical theory of Mealy automata. The approach is mainly algebraic, identifying the main operations able to describe membrane systems, and some algebraic rules governing their functioning.

Membrane systems are also called P systems, and they represent a new abstract model of parallel and distributed computing inspired by cell [11].

A cell is divided in various compartments, each compartment with a specific duty, and all of them working simultaneously to accomplish the task of the whole system. The membranes of a P system determine regions where objects and evolution rules can be placed. The objects evolve according to the rules associated with each region, and the regions cooperate in order to maintain the proper behaviour of the whole system. It is desirable to find more connections with various fields of computer science, including the classical automata theory.

In this paper we present some algebraic properties of multisets, we present Mealy multiset automata [3], and then we define direct and cascade products of Mealy multiset automata corresponding to their parallel and serial connections. We give a characterization of the bisimulation between two Mealy multiset automata, and a result relating their general behaviour to their sequential behaviour. Mealy multiset automata satisfy the criteria of general state based systems, namely their behaviour depends on internal states which can be invisible for the user, the system interacts with its environment and it is not necessarily terminating, and it has a set of operations through which this interaction takes place. For these (reactive) systems, the notions of behaviour, observability, bisimulation become interesting and important. We describe two distinct concepts of behaviour: the sequential behaviour dealing with a specific order of consuming multisets, and a more general behaviour dealing with the outcome of a Mealy multiset automata. The concept of observer could be useful, particularly when we deal with complex structures. The behaviour of a system can be defined as the set of all possible sequences of configurations during a computation. Rather than being concerned with the computations resulting in new states, coalgebraic approaches models for dynamical systems focus on the observable behaviour of system states; the notion of bisimulation is used to formalize observational indistinguishability.

This later remark (and also other advantages that we emphasize in this paper) gives reason for a secondary approach, via category theory. We organize the category of Mealy multiset automata, and we prove that we can view every Mealy multiset automata as a coalgebra of a suitable endofunctor of the category of sets.

2 On Algebra of Multisets

The evolution rules performed by membranes are multiset operators; the multiset operators are associative and commutative, and have also an iden-

tity. In this section we look at multisets, providing some of their algebraic properties.

A *multiset* over an alphabet $A = \{a_1, a_2, ..., a_n\}$ is a mapping $\alpha : A \to \mathbb{N}$. It can be represented by $\{(a_1, \alpha(a_1)), (a_2, \alpha(a_2)), ..., (a_n, \alpha(a_n))\}$. As it is mentioned in [11], a multiset can be also represented as a string $a_1^{\alpha(a_1)} a_2^{\alpha(a_2)} ... {a_n}^{\alpha(a_n)}$ together with all its permutations. A certain extra computation power of P systems comes from the fact that applying a multiset rule $u \to v$ means that we actually apply at most $\mid u \mid! \mid v \mid!$ classical rules on strings. The use of multiplicative notation for both multisets and strings may produce confusion when we interplay multiset and strings rules. It is therefore useful to have distinct notations for multiset operations and string operations.

There are various approaches to deal with multiplicities of the elements of a set. Multisets could be viewed as a particular case of the so-called *formal power polynomials* [10] (i.e. a formal power polynomial over a finite alphabet). However almost all the studies in formal power series do not take care of multisets, and we think that a lot of specific properties are lost. Inspired from formal power polynomials, we denote by $\mathbb{N}\langle A\rangle = \{\alpha : A \to \mathbb{N} \mid \alpha \; isamapping\}$ the set of all multisets on A . The structure of $\mathbb{N}\langle A\rangle$ is mainly an additive one, since we add multiplicities of appearance (in fact, it is induced by the addition in $\mathbb{N}$). This argument is sustained also by the chemical reactions that are the base of the biological modelling. They provide a notation for defining the way a biological system evolves.

If $\alpha, \beta \in \mathbb{N}\langle A\rangle$, then their *sum* is the multiset $(\alpha + \beta) : A \to \mathbb{N}$ defined by $(\alpha + \beta)(a_i) = \alpha(a_i) + \beta(a_i)$, $i = \overline{1, n}$. Moreover, if we consider the letters from A as multisets, i.e. a_i is given by μ_{a_i}, where $\mu_{a_i} : A \to \mathbb{N}$, $\mu_{a_i}(a_i) = 1$ and $\mu_{a_i}(a_j) = 0$ for all $j \neq i$, then we can express every multiset $\alpha \in \mathbb{N}\langle A\rangle$ as a linear combination of a_i, i.e. $\alpha = \sum_{i=1}^{n} \alpha(a_i) \cdot a_i$ (see also [3]). We can define an external operation $m\alpha = \sum_{i=1}^{n} (m\alpha(a_i)) \cdot a_i$, for all $m \in \mathbb{N}$ and $\alpha \in \mathbb{N}\langle A\rangle$.

Proposition 1 *$\mathbb{N}\langle A\rangle$ has a structure of $\mathbb{N}$-semimodule (semimodule over the semiring of positive integers).*

If we want to deal with strings, and apply both kinds of rules, we can work with multisets over A^*, or formal power polynomials, $\mathbb{N}\langle A^*\rangle = \{\alpha : A^* \to \mathbb{N} \mid Supp(\alpha) < \infty\}$, where $Supp(\alpha) = \{w \in A^* \mid \alpha(w) \neq 0\}$ (as usual for multisets, by $supp(\alpha)$ we denote the set $\{w \in A^* \mid \alpha(w) > 0\}$).

The addition of two multisets over strings is defined like in the multisets' case, and ,w.r.t. "+", we have a structure of commutative monoid. In this manner every multiset over A^* may be viewed as a finite linear combination with natural coefficients, $\alpha = \sum_{w \in A^*}^{*} \alpha(w) \cdot w$

It is also defined a product (the Cauchy product) induced from concatenation of strings, $\alpha \bullet \beta = \sum_{w \in A^*}^{*} \sum_{uv=w} \alpha(u)\beta(v) \cdot w$ (we denote in this manner the Cauchy product in order to avoid any confusions). Since concatenation is not commutative, the product is also a non-commutative one. $(\mathbb{N}\langle A^* \rangle, \bullet)$ is a monoid. Note that the star from $\sum_{w \in A^*}^{*}$ means that this sum is finite.

Proposition 2 $(\mathbb{N}\langle A^* \rangle, +, \bullet)$ *is a semiring.*

For other properties of formal power series and related subjects we refer to [10].
We use also the *difference* between two multisets over A or A^*, defined by

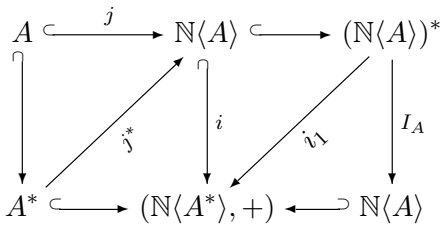
$$(\alpha - \beta)(w) = \alpha(w) - \beta(w),$$

for all α, β such that $\alpha \supseteq \beta$ (i.e. $\alpha(w) \geq \beta(w)$ for all w).

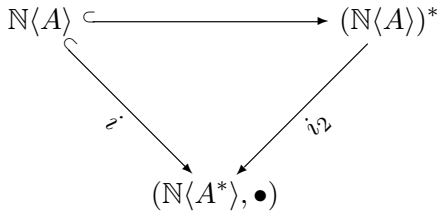
It is possible to work also with strings of multisets, i.e. with elements from the free monoid $(\mathbb{N}\langle A \rangle)^*$. It is worth to mention that this non-commutative monoid has a different structure than $\mathbb{N}\langle A^* \rangle$. Therefore it is useful to clarify what are the relationships between $\mathbb{N}\langle A \rangle$, $(\mathbb{N}\langle A \rangle)^*$ and $\mathbb{N}\langle A^* \rangle$.

We consider the canonical inclusion $i : \mathbb{N}\langle A \rangle \to (\mathbb{N}\langle A \rangle)^*$ and the identity map $id : \mathbb{N}(A) \to \mathbb{N}(A)$. By the universal property of the free monoid $(\mathbb{N}\langle A \rangle)^*$, we know that there exists a unique homomorphism of monoids $\mathbf{I}_A : (\mathbb{N}\langle A \rangle)^* \to \mathbb{N}\langle A \rangle$ such that $\mathbf{I}_A \circ i = id$. We also know that $\mathbf{I}_A$ is defined by $\mathbf{I}_A(a_1...a_n) = a_1 + ... + a_n$, where a_i are all from $\mathbf{I}_A$. Since id is onto, it follows that $\mathbf{I}_A$ is onto, and so, applying the isomorphism theorem for monoids, we obtain that $(\mathbb{N}\langle A \rangle)^*/ker\mathbf{I}_A \backsimeq \mathbb{N}\langle A \rangle$.

Moreover, we have the following diagrams showing all the connections:



and



In these diagrams, "$\hookrightarrow$" represent the canonical inclusions, and $j^* : A^* \to \mathbb{N}\langle A\rangle$ represents the unique homomorphism induced by $j : A \to \mathbb{N}\langle A\rangle$.

We mention here the well-known property of universality of the free monoid over a set, in order to explain better the other homomorphisms of our diagram:

Theorem 1 *If Σ is an arbitrary set, Σ^* is the free monoid on Σ and $i : \Sigma \to \Sigma^*$ is the canonical inclusion, then any mapping $f : \Sigma \to M$, where $(M, \star)$ is a monoid, can be uniquely extended to a monoid homomorphism $f^* : \Sigma^* \to (M, \star)$.*

Moreover, f^* is defined by $f^*(\alpha_1\alpha_2...\alpha_n) = f(\alpha_1) \star f(\alpha_2) \star ... \star f(\alpha_n)$. We look back to our diagrams, and we consider $\Sigma = \mathbb{N}\langle A\rangle$ and $f = i : \mathbb{N}\langle A\rangle \hookrightarrow \mathbb{N}\langle A^*\rangle$. We have two cases:

$(M, \star) = (\mathbb{N}\langle A^*\rangle, +)$. According to the previous theorem, there exists a unique homomorphism $i_1 : (\mathbb{N}\langle A\rangle)^* \to \mathbb{N}\langle A^*\rangle$ extending i. Moreover $i_1(\alpha_1\alpha_2...\alpha_n) = \alpha_1 + \alpha_2 + ... + \alpha_n$, where α_i are from $\mathbb{N}\langle A^*\rangle$. Since i is one-to-one, and its image is $\mathbb{N}\langle A\rangle$, we can conclude that $ker\ i_1 = ker\ \mathbf{I}_A$ and $Im\ i_1 = \mathbb{N}\langle A\rangle$. This means that we do not have any hierarchical relationship between $(\mathbb{N}\langle A^*\rangle, +)$ and $(\mathbb{N}\langle A\rangle)^*$

$(M, \star) = (\mathbb{N}\langle A^* \rangle, \bullet)$, where "$\bullet$" is the Cauchy product of formal power polynomials. According to the previous theorem, there exists a unique homomorphism $i_2 : (\mathbb{N}\langle A \rangle)^* \to \mathbb{N}\langle A^* \rangle$ extending i. Moreover $i_2(\alpha_1\alpha_2...\alpha_n) = \alpha_1 \bullet \alpha_2 \bullet ... \bullet \alpha_n$, where α_i are from $\mathbb{N}\langle A^* \rangle$.

We pay a little more attention to the second case because there are a lot of possibilities to confuse the reader. *ker* $i_2 \neq \emptyset$ since $(a + b)(2a + 2b) \neq (2a + 2b)(a + b)$ in $(\mathbb{N}\langle A \rangle)^*$, but $(a + b) \bullet (2a + 2b) = (2a + 2b) \bullet (a + b)$ in $\mathbb{N}\langle A^* \rangle$. Once again, we can not claim any hierarchical relationship between $\mathbb{N}\langle A^* \rangle$ and $(\mathbb{N}\langle A \rangle)^*$. By a hierarchical relationship we understand here any connection of epimorphic or monomorphic type allowing us to express that one structure can be viewed as a substructure of the other.

Looking back to all these considerations, we can say that, when dealing with *sequential behaviour* of P-systems, $(\mathbb{N}\langle A \rangle)^*$ is more suitable than $\mathbb{N}\langle A^* \rangle$. The main reason is given by the fact that the Cauchy product of $\mathbb{N}\langle A^* \rangle$ is not able to keep the multiplicities of the objects; for instance, $2a \bullet 5b \bullet 2(a + b) = 30a \bullet b \bullet (a + b)$ in $\mathbb{N}\langle A^* \rangle$ and, whenever we have $30a \bullet b \bullet (a + b)$, we can not recover the initial sequence.

3 Mealy Multiset Automata

3.1 Algebraic Description

We introduce here the notion of *Mealy multiset automata* (MmA). Roughly speaking, a MmA consists of a *storage location* (a *box* for short) in which we place a multiset over an input alphabet, and a device to translate that multiset into a multiset over an output alphabet. MmA works in the following way: we have a detection head able to detect whether a given sub-multiset appears in the multiset available in the box. If the sub-multiset is detected, then it is removed from the box, and MmA inserts a multiset over an output alphabet. MmA stops when no further move is possible. We say that the sub-multiset read by the head was translated to a multiset over the output alphabet.

Definition 1 *Formally, a* Mealy multiset automaton *is a construct*

$$\mathcal{A} = (Q, V, O, f, g, q_0)$$

where

Q is a finite set, the set of states*;*

$q_0 \in Q$ *is a special state, both initial and final;*
V *is a finite set of objects, the* input alphabet*;*
O *is a finite set of objects, the* output alphabet*, such that* $O \cap V = \emptyset$*;*
$f : Q \times \mathbb{N}(V) \to \mathcal{P}(Q)$ *is the* state-transition *(*partial*)* mapping*;*
$g : Q \times \mathbb{N}(V) \to \mathcal{P}(\mathbb{N}(O))$ *is the* output *(*partial*)* mapping.

If $\mid f(q,a) \mid \leq 1$ we say that $\mathcal{A}$ is *Q-deterministic*, and if $\mid g(q,a) \mid \leq 1$ we say that $\mathcal{A}$ is *O-deterministic.* MmA is endowed with a box where it receives a multiset. It begins to process this multiset over V, passing through different *configurations*. It starts with a multiset from $\mathbb{N}(V)$, and ends with a multiset from $\mathbb{N}(V \cup O)$.

Definition 2 *A* configuration *of* $\mathcal{A}$ *is a triple* $(q, \alpha, \bar{\beta})$ *where* $q \in Q, \alpha \in \mathbb{N}(V)$, $\bar{\beta} \in \mathbb{N}(O)$. *We say that a configuration* $(q, \alpha, \bar{\beta})$ passes *to* $(s, \alpha - a, \bar{\beta} + \bar{b})$ *(or, that we have a* transition *between those configurations) if there is* $a \subseteq \alpha$ *such that* $s \in f(q,a), \bar{b} \in g(q,a)$. *We denote this by* $(q, \alpha, \bar{\beta}) \vdash (s, \alpha - a, \bar{\beta} + \bar{b})$. *We also denote by* $\vdash^*$ *the reflexive and transitive closure of* $\vdash$.

Remark 1 *We could alternatively define a configuration to be a pair* (q, α) *where* $\alpha \in \mathbb{N}(V \cup O)$*, and the transition relation is* $(q, \alpha) \vdash (s, \alpha - a + \bar{b})$*, with the same conditions as above.*

Definition 3 *A multiset* $\alpha \in \mathbb{N}(V)$ *is said to be a* totally consumed multiset *(*tc-multiset*) for* $\mathcal{A}$ *if, starting from the configuration* $(q_0, \alpha, \varepsilon)$ *MmA can pass through various configurations till it arrives in a configuration* $(q_0, \varepsilon, \bar{\beta})$ *(i.e.,* $(q_0, \alpha, \varepsilon) \vdash^* (q_0, \varepsilon, \bar{\beta})$*).*

A multiset $\alpha \in \mathbb{N}(V)$ *is said to be a* consumed multiset *(*c-multiset*) for* $\mathcal{A}$ *if, starting from a configuration* (q, α, ε)*, MmA can pass through various configurations till it arrives in a configuration* $(s, \varepsilon, \bar{\beta})$ *(i.e.,* $(q, \alpha, \varepsilon) \vdash^* (s, \varepsilon, \bar{\beta})$*).*

In both cases, we say also that α *was* entirely translated to $\bar{\beta}$*. In all the other situations we say that* $\alpha \in \mathbb{N}(V)$ *is* partially consumed *(*pc-multiset*), or it is* partially translated.

We denote by $TC(\mathcal{A})$ the set of all tc-multisets of $\mathcal{A}$, by $C(\mathcal{A})$ the set of all c-multisets of $\mathcal{A}$, and by $PC(\mathcal{A})$ the set of all pc-multisets of $\mathcal{A}$. It is clear that $TC(A) \subseteq C(A)$.

Theorem 2 $TC(\mathcal{A})$ *is a* $\mathbb{N}$*-sub-semimodule of* $\mathbb{N}(V)$*. Moreover, if we define* $\mathcal{A}(\alpha) = \bar{\beta}$ *for all* $\alpha \in TC(\mathcal{A})$ *with* $(q_0, \alpha, \varepsilon) \vdash^* (q_0, \varepsilon, \bar{\beta})$*, we may view* $\mathcal{A}$ *as an* $\mathbb{N}$*-homomorphism from* $TC(\mathcal{A})$ *to* $\mathbb{N}(O)$*.*

Remark 2 *In general, $C(\mathcal{A})$ it is not a $\mathbb{N}$-sub-semimodule of $\mathbb{N}(V)$. Let us consider $\alpha, \alpha' \in C(\mathcal{A})$. We have $(q, \alpha, \varepsilon) \vdash^* (q', \varepsilon, \bar{\beta})$ and $(s, \alpha', \varepsilon) \vdash^* (s', \varepsilon, \bar{\beta}')$; therefore $(q, \alpha + \alpha', \varepsilon) \vdash^* (q', \alpha', \bar{\beta})$, and it is possible that the automaton can not go further (for instance, we may have $f(q', a') = \emptyset$ for all $a' \subseteq \alpha'$).*

It is possible for two multisets $\alpha, \alpha' \in \mathbb{N}(V)$ to have their sum in $TC(\mathcal{A})$, even they are not in $TC(\mathcal{A})$ (see the example bellow). Let us give an example:

Example 1 *Consider $\mathcal{A} = (\{s_0, s_1, s_2\}, \{a, b, c\}, \{d, e, f\}, f, g, s_0)$ with*

- *$f(s_0, 2a) = \{s_1, s_2\}$, $f(s_1, b) = s_0$, $f(s_2, c) = s_0$ for the transition function,*
- *$g(s_0, 2a) = e$ if $f(s_0, 2a) = s_1$, $g(s_0, 2a) = d$ if $f(s_0, 2a) = s_2$, $g(s_1, b) = e$, $g(s_2, c) = f$ for the output mapping.*

It is easy to see that $TC(\mathcal{A}) = \{m(2a + b) + n(2a + c) \mid m, n \in \mathbb{N}\}$. The set of tc-translations is $\mathcal{A}(TC(\mathcal{A})) = \{2me + n(d + f) \mid m, n \in \mathbb{N}\}$. We have also that $a + b, a$ are not in $TC(\mathcal{A})$, not even in $C(\mathcal{A})$, but their sum belongs to $TC(\mathcal{A})$. Similarly, $6a + b, 2a + 3c \notin TC(\mathcal{A})$, $6a + b, 2a + 3c \in PC(\mathcal{A})$, and their sum is in $TC(\mathcal{A})$.

Remark 3 *We do not provide a representation for MmA in the form of a graph as in [5], simply because graphs are strongly related with sequencing and do not permit to express facts like "if we can consume two multisets a and b, and their sum is available in the box, it does not matter the order of consuming them". This is an important difference between MmA and weighted automata [10] (or K-Σ automata [7]).*

From now on, we restrict ourselves to the *deterministic* case, i.e. our MmA's are both Q-deterministic and O-deterministic. Moreover, we do not include an initial state in our definition, simply because there is no reason to focus attention to one particular state. In the classical theory of automata, initial states play a certain role, for instance in the definition of the sequential composition of two automata, where all the terminating states of the first automaton are connected to the initial state of the second automaton. Without specifying the initial set of a MmA, all the considerations are valid, except the tc-multiset notion; we consider now only c-multisets and pc-multisets associated with an arbitrary state q.

Definition 4 *Given two MmA's* $\mathcal{A} = (Q, V, O, f, g)$ *and* $\mathcal{A}' = (Q', V, O, f', g')$*, a function* $h : Q \to Q'$ *is called a* morphism *from* $\mathcal{A}$ *to* $\mathcal{A}'$ *if the following conditions are satisfied:*

- $h(f(q, a)) = f'(h(q), a)$,
- $g(q, a) = g'(h(q), a)$,

for all $q \in Q$*, and for all* $a \in \mathbb{N}(V)$.

If $h : Q \to Q'$ is a morphism between $\mathcal{A}$ and $\mathcal{A}'$, we denote this by $h : \mathcal{A} \to \mathcal{A}'$.

Let $h : \mathcal{A} \to \mathcal{A}'$ be a morphism, and $(q, \alpha, \bar{\beta})$ a configuration of $\mathcal{A}$. Let us suppose that we have the transition $(q, \alpha, \bar{\beta}) \vdash (s, \alpha - a, \bar{\beta} + \bar{b})$. This means that $s = f(q, a), \bar{b} = g(q, a)$. We get that $h(s) = h(f(q, a)) = f'(h(q), a)$, and $\bar{b} = g(q, a) = g'(h(q), a)$, and so we have $(h(q), \alpha, \bar{\beta}) \vdash (h(s), \alpha - a, \bar{\beta} + \bar{b})$. Therefore we have the following result:

Theorem 3 *Let* $h : \mathcal{A} \to \mathcal{A}'$ *be a morphism of MmA's. If the multiset* $\alpha \in \mathbb{N}(V)$ *is a c/pc-multiset for* $\mathcal{A}$*, then* α *has the same nature for* $\mathcal{A}'$.

This result underlines that if h is a morphism between two MmA's, then it is not possible to have α both as a pc-multiset for $\mathcal{A}$, and as a c-multiset for $\mathcal{A}'$, i.e. we get a kind of invariance property under morphisms for $C(\mathcal{A})$ and $PC(\mathcal{A})$.

3.2 Series and Parallel Connections

The cascade product

This is a way to make a series connection in the case of Mealy Automata, and provide also some results in decompositions of such machines in irreducible ones. Even we are not prepared yet to give such theorems (this involves a lot of algebra for multisets), we define the cascade product of two MmA's.

Let $\mathcal{A} = (Q, V, O, f, g)$ and $\mathcal{A}' = (Q', V', O', f', g')$ two MmA's. In order to link them by a series connection, we need a multiset mapping to link the output of one of them to the input of the other. This can be done using a $\mathbb{N}$-homomorphism from $\mathbb{N}(O')$ to $\mathbb{N}(V)$; this homomorphism can be obtained as usually using a mapping from O' to V. We denote this homomorphism by $\Lambda : \mathbb{N}(O') \to \mathbb{N}(V)$, and we obtain a mapping $\Omega : Q' \times \mathbb{N}(V') \to \mathbb{N}(V)$, defined by $\Omega(q', a') = \Lambda(g'(q', a'))$.

- This mapping gives us *the cascade product induced by* Ω:

$$\mathcal{A}\Omega\mathcal{A}' = (Q \times Q', V', O, f^{\Omega}, g^{\Omega})$$

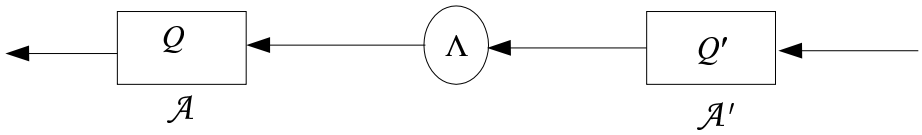
where $f^{\Omega} : (Q \times Q') \times \mathbb{N}(V') \to Q \times Q'$, $g^{\Omega} : (Q \times Q') \times \mathbb{N}(V') \to O$ are given by $f^{\Omega}((q, q'), a') = (f(q, \Omega(q', a')), f'(q', a'))$, and $g^{\Omega}((q, q'), a') = g(q, \Omega(q', a'))$ for all $a' \in \mathbb{N}(V')$, $(q, q') \in Q \times Q'$.

- The transition relation becomes $((q, q'), \alpha', \bar{\beta}) \vdash ((s, s'), \alpha' - a', \bar{\beta} + \bar{b})$ if there is $a' \subseteq \alpha'$ such that $(s, s') \in f^{\Omega}((q, q'), a'), \bar{b} \in g^{\Omega}((q, q'), a')$, where $a', \alpha' \in \mathbb{N}(V'), (q, q') \in Q \times Q', \bar{\beta} \in \mathbb{N}(O)$.

We can alternatively define the transition relation by

$$((q, q'), \alpha' + \bar{\beta}) \vdash ((s, s'), \alpha' - a' + \bar{\beta} + \bar{b})$$

if there is $a' \subseteq \alpha'$ such that $s \in f(q, \Lambda(g'(q', a')))$, $s' \in f'(q', a')$, and $\bar{b} \in g(q, \Lambda(g'(q', a')))$, where $a', \alpha' \in \mathbb{N}(V'), (q, q') \in Q \times Q', \bar{\beta} \in \mathbb{N}(O)$.
The graphical representation of the cascade product is given by the following figure:



As we already mention in [3], in order to simulate an elementary membrane, we also need a kind of direct product of MmA's. We consider only a restricted variant, because the input alphabets (and also the output alphabets) are the same for all MmA's involved.

Restricted direct product

Let $\mathcal{A}_i = (Q_i, V, O, f_i, g_i)$ be a finite family of Mealy multiset automata, and B_i their corresponding boxes, $i = \overline{1, n}$. We can connect them in *parallel* in order to obtain the *restricted direct product* of $\mathcal{A}_i$ defined by $\mathcal{A} = \bigwedge_{i=1}^{n} \mathcal{A}_i = (\times_{i=1}^{n} Q_i, V, O, f, g)$, where:

- $f((q_1, q_2, ..., q_n), a) = (f_1(q_1, a), f_2(q_2, a), ..., f_n(q_n, a))$;
- $g((q_1, q_2, ..., q_n), a) = (g_1(q_1, a), g_2(q_2, a), ..., g_n(q_n, a))$;

- The box B of $\mathcal{A}$ is the disjoint union $\bigsqcup_{i=1}^{n} B_i$;
- A *configuration* of $\mathcal{A}$ is a triple $(q, \alpha, \bar{\beta})$, where $q = (q_1, q_2, ..., q_n), \alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$, and $\bar{\beta} = (\bar{\beta}_1, \bar{\beta}_2, ..., \bar{\beta}_n)$;
- The (*asynchronous*) *transition relation* of $\mathcal{A}$: $(q, \alpha, \bar{\beta}) \vdash (s, \alpha - a, \bar{\beta} + \bar{b})$ if and only if there is at least an $i \in \overline{1, n}$ such that $s_i \in f_i(q_i, a_i)$, and $\bar{b}_i \in g_i(q_i, a_i)$, where $a = (a_1, a_2, ..., a_n)$, and $\bar{b} = (\bar{b}_1, \bar{b}_2, ..., \bar{b}_n)$.

3.3 Bisimulation and Observability

The **bisimulation** relation between states of a transition system was originally introduced by Park and Milner, in order to formalize the behavioural equivalence of concurrent processes. In our case,

Definition 5 *A* bisimulation *between two MmA's* $\mathcal{A} = (Q, V, O, f, g, q_0)$ *and* $\mathcal{A}' = (Q', V, O, f', g', q_0')$ *is a relation* $R \subseteq Q \times Q'$ *such that for all* $a \in \mathbb{N}(V)$, *if* qRq' *then* $g(q, a) = g'(q', a)$ *and* $f(q, a)Rf'(q', a)$. *A bisimulation between* Q *and itself is called a* bisimulation on Q.

It can be verified without difficulty that union and (relational) composition of bisimulations are bisimulations again. We write $q \sim q'$ whenever there exists a bisimulation R with qRq'. This relation is the union of all bisimulations and, therewith, the greatest bisimulation. The greatest bisimulation on the same automaton, again denoted by $\sim$, is called the *bisimilarity* relation, and it is an equivalence relation.

Two states related by a bisimulation relation are *observationally indistinguishable* in the sense that

1. they provide the same output, and
2. performing the same experiment on both states, we get states that are indistinguishable again.

We can relate the notion of MmA morphism to that of bisimulation (see also [4] for more details).

Theorem 4 *A function* $h : Q \to Q'$ *is a morphism from* $\mathcal{A}$ *to* $\mathcal{A}'$ *if and only if its graph relation* $G(h) = \{(q, h(q)) \mid q \in Q\}$ *is a bisimulation.*

Regarding the **observability**, we can remark that one of the main features of our Mealy multiset automata is that we have an intrinsic *observer*

given by the *output mapping*. As we can see in our previous example, there are transitions that cannot be observed from outside, i.e. transitions for which the output mapping is the empty multiset ε.

A state q is *observable* from an other state s if there exist a multiset α such that $(s, \alpha, \varepsilon) \vdash (q_1, \alpha - a_1, b_1) \vdash (q_2, \alpha - a_1 - a_2, b_1 + b_2) \vdash ... \vdash (q_n, \alpha - a_1 - ... - a_n, b_1 + ... + b_n)$, $q_n = q$ and $b_n \neq \varepsilon$.

One of the main differences between MmA and the classical automata is given by the possibility of the detection head of MmA to choose, in a given state, various sub-multisets from the input multiset. This means that for the same input multiset, we can have various possibilities to go further from a given state. This remark emphasize the important role played by the output mapping as an observer.

3.4 Behaviour

Behaviour is often appropriately viewed as consisting of both dynamics and observations, namely state-transition and output mappings. The main advantage of a MmA is given by its output function playing an important role in observability (we do not construct other machinery to describe the behaviour of our MmA).

Definition 6 *Let $\mathcal{A} = (Q, V, O, f, g)$ be a Mealy multiset automaton. The* general behaviour *of a state $q \in Q$ is a function* $\mathbf{beh}(q)$ *that assigns to every input multiset $\alpha \in \mathbb{N}(V)$ the output multiset obtained after consuming α.*

We have to remark that, when we talk about behaviour of a state, we have to consider a specific order of consuming multisets in terms of strings of multisets. A certain feature for MmA is that the behaviour is always finite since we can not go further after consuming the given multiset. On the other hand, since the outputs go back into the box, and it can become larger, it is possible that we can not track the sequence of intermediate states. If we are interested only on the outcome of our machine, we should not take care of the intermediate states, but if the input multiset is partially consumed (i.e. is a pc-multiset), it should be of interest to know the state where the MmA arrives, in order to (possibly) provide the box with a supplementary multiset, or in order to make the initial one a consumed multiset. These considerations lead us to the following

Definition 7 *Let $\mathcal{A} = (Q, V, O, f, g)$ be a Mealy multiset automaton. The* sequential behaviour *of a state $q \in Q$ is a function* $\mathbf{seqbeh}(q)$ *that assigns*

to every multiset $\alpha \in \mathbb{N}(V)$ *all the sequences of the output multisets obtained during consuming* α.

Example 2 *Suppose that we have the following sequence of transitions* $(q, \alpha, \varepsilon) \vdash (q_1, \alpha - a_1, b_1) \vdash (q_2, \alpha - a_1 - a_2, b_1 + b_2) \vdash ... \vdash (q_n, \alpha - a_1 - ... - a_n, b_1 + ... + b_n)$ *and suppose that this MmA stops. Then* $\mathbf{beh}(q)(\alpha) = b_1 + ... + b_n$, *and* $\mathbf{seqbeh}(q)(\alpha)$ *contains* $b_1...b_n$. *Moreover,* $b_1 + ... + b_n$ *belongs to* $\mathbb{N}(O)$, *while* $b_1...b_n$ *belongs to* $(\mathbb{N}(O))^*$, *the free monoid on* $\mathbb{N}(O)$.

We consider the canonical inclusion $i : \mathbb{N}(O) \to (\mathbb{N}(O))^*$, and the identity map $id : \mathbb{N}(O) \to \mathbb{N}(O)$. As we have already mentioned there exists a unique homomorphism of monoids $\mathbf{I}_O : (\mathbb{N}(O))^* \to \mathbb{N}(O)$ such that $\mathbf{I}_O \circ i = id$. This homomorphism $\mathbf{I}_O$ is defined by $\mathbf{I}_O(b_1...b_n) = b_1 + ... + b_n$. Since id is onto, it follows that $\mathbf{I}_O$ is onto, and so, by applying the isomorphism theorem for monoids, we have that $(\mathbb{N}(O))^*/ker\mathbf{I}_O \backsimeq \mathbb{N}(O)$. Moreover, $\mathbf{I}_O \circ (\mathbf{seqbeh})(q) = \mathbf{beh}(q)$.

Example 3 *Consider* $\mathcal{A} = (\{s_0, s_1, s_2, s_3\}, \{a, b\}, \{c, d\}, f, g)$ *with the transition function* f *given by* $f(s_0, 2a) = s_1, f(s_0, a) = s_2, f(s_1, 2b) = s_2, f(s_0, 2b) = s_3, f(s_0, 3b) = s_2, f(s_1, 2a + b) = s_3, f(s_1, a) = s_3, f(s_2, a) = s_3, f(s_2, b) = s_1$, *and the output function* g *given by* $g(s_0, 2a) = 2c, g(s_0, a) = c, g(s_1, 2b) = d, g(s_0, 2b) = 2c + d, g(s_0, 3b) = \varepsilon$, $g(s_1, 2a + b) = c, g(s_1, a) = \varepsilon, g(s_2, a) = \varepsilon, g(s_2, b) = c$, *where* ε *corresponds to transitions that cannot be "viewed" by an external observer.*

Then $\mathbf{seqbeh}(s_0)(3a + 2b)$ *contains* $(2c)d\varepsilon$ *for the following sequence of transitions* $(s_0, 3a + 2b, \varepsilon) \vdash (s_1, a + 2b, 2c) \vdash (s_2, a, 2c + d) \vdash (s_3, \varepsilon, 2c + d)$. *The same input multiset can also be consumed in the following ways:*

$(s_0, 3a + 2b, \varepsilon) \vdash (s_1, a + 2b, 2c) \vdash (s_0, 2b, 2c) \vdash (s_3, \varepsilon, 2c + d)$, *or*

$(s_0, 3a + 2b, \varepsilon) \vdash (s_2, 2a + 2b, d) \vdash (s_1, 2a + b, c + d) \vdash (s_3, \varepsilon, 2c + d)$.

Hence $\mathbf{seqbeh}(s_0)(3a + 2b) = \{(2c)d, dcc\}$. *Therefore, independent of the consuming sequences, the general behaviour of* s_0 *is* $\mathbf{beh}(s_0)(3a + 2b) = 2c + d$.

It is interesting to remark that our bisimulation preserves **seqbeh**. If q, q' are two bisimilar states, i.e. $q \sim q'$, they have the same sequential behaviour $\mathbf{seqbeh}(q) = \mathbf{seqbeh}(q')$. This implies that they also have the same behaviour $\mathbf{beh}(q) = \mathbf{beh}(q')$. Since the reciprocal it is not true, we can define a weaker equivalence relation, namely

$$q \approx q' \Leftrightarrow \mathbf{beh}(q) = \mathbf{beh}(q')$$

Therefore we can easily obtain:

Proposition 3 *$q \approx q'$ if and only if $(\mathbf{seqbeh}(q), \mathbf{seqbeh}(q')) \in ker\mathbf{I}_O$ for all $\alpha \in \mathbb{N}(V)$.*

Since this equivalence relation over states is given by the general behaviour **beh**, we can say that this relation is independent of the order of consuming resources from the box, and we call it an *output conservative equivalence*. The importance of this equivalence is given mainly by the idea of consuming and producing *resources* by overpassing the sequential framework represented by **seqbeh**. This problem appears to be of interest when we consider the concurrent processes competing for resources.

4 Category Theory and Mealy Multiset Automata

The aim of this section is to explain and explore some of the current ideas from category theory that enable various mathematical descriptions of hierarchical structures and membrane systems in particular. The abstraction level of category theory allows us to work with objects and morphisms without considering their internal structure. This seems to be a very appropriate setting for membrane computing. The categorical approach is based on the definition of a category whose objects model system components, and whose morphisms represent how systems are composed, simulated, refined, etc. Complex systems can be expressed by diagrams in category theory. This approach is appropriate for modelling systems having shared resources (see also eMMA of [3, 4]).

4.1 Categories and Functors

Definition 8 *A* category $\mathcal{C}$ *consists of:*

- a class of objects*;*

- a class of morphisms *(*arrows*);*

- for each morphism f, one object as domain*, and another as* codomain *of f;*

- for each object A, an identity morphism *id_A;*

- for each pair of morphisms $f : A \to B$ and $g : B \to C$ (i.e. cod(f)=dom(g)),
a composite morphism $g \circ f : A \to C$.
This composition have to satisfy the following rules:

- Associativity*: For each set of morphisms $f : A \to B, g : B \to C, h : C \to D$,*
$h \circ (g \circ f) = (h \circ g) \circ f$.

- Identity*: For each morphism* $f : A \to B$, $f \circ id_A = f$, $id_B \circ f = f$

The *functorial character* of a categorial construction is important for at least two reasons:

- working within categories, we make explicit the morphisms which correspond to appropriate notions of simulation or refinement between systems;

- functors act on objects and behave consistently on their simulations, preserving them (moreover, when functors are adjoint, they preserve limits or colimits, yielding good compositional properties, since complex systems can be expressed as (co)limits of their simpler components).

Definition 9 *Given two categories* $\mathcal{C}$ *and* $\mathcal{D}$*, a* functor *between them* $F : \mathcal{C} \to \mathcal{D}$ *is a pair* $F = (F_{ob}, F_{mor})$ *where*

$F_{ob} : Ob(\mathcal{C}) \to Ob(\mathcal{D})$ *the* object mapping*;*

$F_{mor} : Mor\mathcal{C}(A, B) \to Mor\mathcal{D}(F_{ob}(A), F_{ob}(B))$ *the* morphism mapping*,*

such that if $f : A \to B$ *then* $F_{mor}(f) : F_{ob}(A) \to F_{ob}(B)$*,*

satisfying the following axioms:

Compositionality: $F_{mor}(gf) = F_{mor}(g)F_{mor}(f)$

Identity: $F_{mor}(id_A) = id_{F_{mor}(A)}$.

Some interesting examples that come from computer science should include the category $\mathcal{MAUT}$ of Mealy automata, and the category $\mathcal{BEH}$ of behaviours. We emphasize the functorial connection between the category of Mealy automata and the category of their behaviours.

Example 4 *Let* $\mathcal{A} = (Q, V, O, f, g, s_0)$ *be a Mealy automaton with a distinguished state* s_0*, where* Q, V, O *are the sets of states, input symbols and output symbols, respectively;* f *is the next state function, and* g *is the output function. We denote, as usual, with the same letters* f *and* g *the extension of* f *from* $Q \times V$ *to* $Q \times V^*$*, and the extension of* g *from* $Q \times O$ *to* $Q \times O^*$*. For every state we can define* $beh_{\mathcal{A}} : V^* \to O^*, beh_{\mathcal{A}}(w) = g(s_0, w)$*. We consider now the following categories:*

$\mathcal{MAUT}$*: the category of Mealy automata. It has Mealy automata as objects, and a morphism* $\alpha : \mathcal{A} \to \mathcal{A}'$ *is a triple* $(\alpha_1, \alpha_2, \alpha_3)$*,* $\alpha_1 : Q \to Q', \alpha_1(s_0) = s_0'$*,* $\alpha_2 : V \to V'$ *and* $\alpha_3 : O \to O'$ *such that* $\alpha_1(f(s, a)) = f'(\alpha_1(s), \alpha_2(a))$ *and* $\alpha_3(g(s, a)) = g'(\alpha_1(s), \alpha_2(a))$*.*

$\mathcal{BEH}$*: the category of behaviours. It has triples* $(V, O, beh : V \to O)$ *as objects, and a morphism* $\beta : (V, O, beh) \to (V', O', beh')$ *is a pair* (β_1, β_2)*,* $\beta_1 : V \to V', \beta_2 : O \to O'$*such that* $\beta_2 \circ beh = beh' \circ \beta_1$*.*

We can define the functor $Beh : \mathcal{MAUT} \to \mathcal{BEH}$ *in the following manner:*

- on objects: $Beh(\mathcal{A}) = (V, O, beh_{\mathcal{A}})$,

- on morphisms: if $\alpha : \mathcal{A} \to \mathcal{A}'$ *is a morphism in* $\mathcal{AUT}$*, then* $Beh(\alpha)=(\alpha_2, \alpha_3)$.

We can also organize Mealy multiset automata as a category.

Proposition 4 *For fixed alphabets* V *and* O*, the collection of Mealy multiset automata together with their morphisms form a category denoted by* $\mathcal{MA}_{VO}$.

More information on category theory are freely available in [2]. As we have already explained, the lack of a comprehensive approach for the algebra of multisets, together with the necessity of some mechanisms to connect and compose several MmA's, lead us to initiate a study of the category $\mathcal{MA}_{VO}$. We have two possible approaches to obtain the existence of some usual constructions (i.e. limits and colimits like (co)products, pushout, pullback, (co)equalizers,) in the category of Mealy multiset automata. Once we can use the classical way, i.e. we can construct step by step everything we need, leading us to consume a lot of "paper" and without visible benefits or, more elegant, using the categorial coalgebraic point of view for transition-like systems. Moreover, this latter approach permits us to work easier with concepts like bisimulation, bisimilarity and behaviour.

4.2 Short Introduction to Coalgebra

We introduce briefly some of the basic notions of coalgebra, homomorphism, and bisimulation relation; see [12] for more details.

Let $\mathcal{C}$ be a category, and $F : \mathcal{C} \to \mathcal{C}$ be a functor. An *F-coalgebra* or *F-system* is a pair (S, α_S) consisting of an object S and a morphism $\alpha_S : S \to F(S)$. The object S is called the *carrier* of the system, also to be called the *set of states*; the morphism α_S is called the *F-transition structure* of the system. When no explicit reference to the functor is needed, we simply speak of system and transition structure. Moreover, when no explicit reference to the transition structure is needed, we often use S instead of (S, α_S).

Definition 10 *Let* (S, α_S) *and* (T, α_T) *be two F-systems, where F is again an arbitrary functor. A morphism* $f : S \to T$ *is a* homomorphism *of F-systems, or F-*homomorphism*, if* $F(f) \circ \alpha_S = \alpha_T \circ f$*, i.e. the following*

diagram is commutative:

$$\begin{array}{ccc} S & \xrightarrow{\alpha_S} & F(S) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle F(f)} \\ T & \xrightarrow{\alpha_T} & F(T) \end{array}$$

Intuitively, homomorphisms are functions that preserve and reflect F-transition structures. We sometimes write $f : (S, \alpha_S) \to (T, \alpha_T)$ to express that f is a F-homomorphism. The identity function of an F-system (S, α_S) is a homomorphism, and the composition of two homomorphisms is again a homomorphism. Thus the collection of all F-systems together with F-system homomorphisms is a category, denoted by $\mathcal{C}_F$.

Definition 11 *Let F be an arbitrary functor $F : \mathcal{C} \to \mathcal{C}$ and let $(S, \alpha_S), (T, \alpha_T)$ be F-coalgebras. An object (R, α_R) from $\mathcal{C}_F$, together with two morphisms $p : R \to S, q : R \to T$ (called* projections*), is called to be a* bisimulation *between (S, α_S) and (T, α_T) if p and q are also homomorphisms of F-coalgebras, i.e. $F(p) \circ \alpha_R = \alpha_S \circ p$ and $F(q) \circ \alpha_R = \alpha_T \circ q$. See also the following diagram:*

$$\begin{array}{ccccc} S & \xleftarrow{p} & R & \xrightarrow{q} & T \\ {\scriptstyle \alpha_S}\downarrow & & \downarrow{\scriptstyle \alpha_R} & & \downarrow{\scriptstyle \alpha_T} \\ F(S) & \xleftarrow[F(p)]{} & F(R) & \xrightarrow[F(q))]{} & F(T) \end{array}$$

A special case is obtained for $\mathcal{C}$ is Set, the category of sets. For a comprehensive approach we refer to [12]. We mention only some facts: A subset $R \subseteq S \times T$ of the Cartesian product of S and T is called an F-*bisimulation* between S and T if there exists an F-transition structure $\alpha_R : R \to F(R)$ such that the projections from R to S and T are F-homomorphisms. We say that (R, α_R) is a bisimulation between (S, α_S) and (T, α_T). If $(S, \alpha_S) = (T, \alpha_T)$, then (R, α_R) is called a *bisimulation* on (S, α_S). A *bisimulation equivalence* is a bisimulation that is also an equivalence relation. Two states s and t are called *bisimilar* if there exists a bisimulation R with $(s, t) \in R$. According to [12], a fundamental relationship between homomorphisms and bisimulations is given by

Theorem 5 *Let (S, α_S) and (T, α_T) be two systems. $f : S \to T$ is a homomorphism if and only if its graph $G(f)$ is a bisimulation between (S, α_S) and (T, α_T).*

5 Mealy Multiset Automata as Coalgebras

Coalgebra can be understood as a theory that deals with behavioural aspects of dynamic systems in a rather wide sense. Behaviour is often appropriately viewed as consisting of both dynamics and observations, which have to do with change of states and partial access to states, respectively. Bisimulation was introduced into the world of coalgebra by Aczel and Mendler [1], who gave a categorical definition of bisimulation that applies to arbitrary coalgebras. Let us consider two alphabets V and O and the functor $F : Set \to Set$ defined by

- $F(Q) = (Q \times \mathbb{N}\langle O\rangle)^{\mathbb{N}\langle V\rangle}$
- If $h : Q \to Q'$ is a mapping (i.e. morphism in Set) then $F(h) : (Q \times \mathbb{N}\langle O\rangle)^{\mathbb{N}\langle V\rangle} \to (Q' \times \mathbb{N}\langle O\rangle)^{\mathbb{N}\langle V\rangle}$ is defined by $F(h)(k) = \left\langle h, id_{\mathbb{N}\langle O\rangle}\right\rangle \circ k$

Definition 12 *A* coalgebra *for F is a set Q together with a morphism $\alpha_Q : Q \to F(Q) = (Q \times \mathbb{N}\langle O\rangle)^{\mathbb{N}\langle V\rangle}$.*

It is obvious that, starting from a coalgebra (Q, α_Q), we can obtain a MmA $\mathcal{A} = (Q, V, O, f, g)$, where $f(q, a)$is the first component of $\alpha_Q(q)(a)$ and $g(q, a)$ is the second component of $\alpha_Q(q)(a)$. Of course, if $\mathcal{A} = (Q, V, O, f, g)$ is a MmA, we can obtain a coalgebra for F, with $\alpha_Q : Q \to F(Q) = (Q \times \mathbb{N}\langle O\rangle)^{\mathbb{N}\langle V\rangle}$, defined by $\alpha_Q(q)(a) = (f(q, a), g(q, a))$.

Let $h : (Q, \alpha_Q) \to (Q', \alpha_{Q'})$ a F-morphism, i.e. $F(h) \circ \alpha_Q = \alpha_{Q'} \circ h$, and $\mathcal{A} = (Q, V, O, f, g)$,$\mathcal{A}' = (Q', V, O, f', g')$ their attached MmA's. This implies that for all $q \in Q$ and for all $a \in \mathbb{N}\langle V\rangle$ we have $(F(h) \circ \alpha_Q)(q)(a) = (\alpha_{Q'} \circ h)(q)(a) \Leftrightarrow F(h)(\alpha_Q(q))(a) = \alpha_{Q'}(h(q))(a) \Leftrightarrow \left\langle h, id_{\mathbb{N}\langle O\rangle}\right\rangle(\alpha_Q(q)(a)) = (f'(h(s), a), g'(h(s), a)) \Leftrightarrow \left\langle h, id_{\mathbb{N}\langle O\rangle}\right\rangle(f(q, a), g(q, a)) = (f'(h(q), a), g'(h(q), a)) \Leftrightarrow (h(f(q, a)), g(q, a)) = (f'(h(s), a), g'(h(s), a))$. We obtain the following

Proposition 5 *$h : (Q, \alpha_Q) \to (Q', \alpha_{Q'})$ is a F-morphism if and only if $h : Q \to Q'$ is a morphism between their associated automata.*

It can be proved that the classical MmA bisimulation is an instance of the general coalgebraic definition.

Theorem 6 *If $R \subseteq Q \times Q'$ is an* F-bisimulation *between coalgebras (Q, α_Q) and $(Q', \alpha_{Q'})$ then R is a bisimulation between their corresponding MmA, $\mathcal{A} = (Q, V, O, f, g,)$ and $\mathcal{A}' = (Q', V, O, f', g',)$.*

If we want to prove statements like: "the union of a collection of bisimulations is again a bisimulation"; "the quotient of a system with respect to a bisimulation equivalence is again a system"; and: "the kernel of a homomorphism is a bisimulation equivalence" we need three basic constructions in the category of F-systems: the formation of coproducts (sums), coequalizers, and (weak) pullbacks. The first two constructions exist conform theorem 4.2 from [12]:

Theorem 7 *Let $F : Set \to Set$ be any functor. In the category Set_F of F-coalgebras, all coproducts and coequalizers exist, and are constructed as in Set.*

As a corollary, we obtain directly that all coproducts and coequalizers exist in the category of Mealy multiset automata.

Therefore we have to construct the pullbacks. Following and using some results from [12], we have the following theorem:

Theorem 8 *Let $F : Set \to Set$ be any functor, Set_F the category of F-coalgebras, and $U : Set_F \to Set$ the forgetful functor.*

1. *If F preserves pullbacks, then pullbacks exist in Set_F.*
2. *If F preserves weak pullbacks, and let $f : (S, \alpha_S) \to (T, \alpha_T)$ and $g : (Q, \alpha_Q) \to (T, \alpha_T)$ be homomorphisms of F-coalgebras. Then the pullback (P, π_1, π_2) of f and g in Set is a bisimulation on S and T.*
3. *The functor $U : Set_F \to Set$ creates colimits. This means that any type of colimit in Set_F exists, and it is obtained by first constructing the colimit in Set and next supplying it (in a unique way) with an F-transition structure.*

In order to obtain our desired construction, the only thing that we have to do is to prove that our functor F introduced in the beginning of this section preserves weak pullbacks.

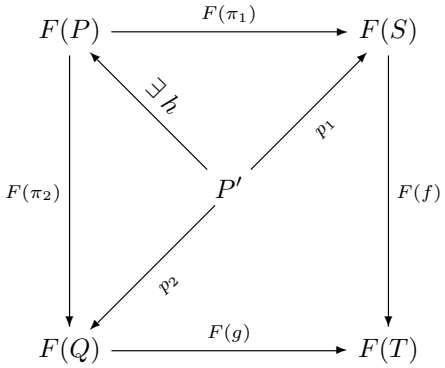
Let $f : S \to T$ and $g : Q \to T$ be morphisms in Set, (P, π_1, π_2) a weak pullback of f and g in Set ($P = \{(s, q) \in S \times Q \mid f(s) = g(q)\}$), and our functor $F : Set \to Set$ which is defined by $F(-) = (- \times \mathbb{N}\langle O \rangle)^{\mathbb{N}\langle V \rangle}$; we have to prove that $(F(P), F(\pi_1), F(\pi_2))$ is a weak pullback of $F(f)$ and $F(g)$.

Let us consider the following diagram

$$\begin{array}{ccc} F(P) & \xrightarrow{F(\pi_1)} & F(S) \\ {\scriptstyle F(\pi_2)}\downarrow & & \downarrow{\scriptstyle F(f)} \\ F(Q) & \xrightarrow{F(g)} & F(T) \end{array}$$

Since $f \circ \pi_1 = g \circ \pi_2$, and F is a functor, we have $F(f) \circ F(\pi_1) = F(g) \circ F(\pi_2)$, and so the above diagram is commutative.

It remains to prove the property of universality, i.e. for all (P', p_1, p_2) such that $F(f) \circ p_1 = F(g) \circ p_2$, there is a morphism $h : P' \to F(P)$ such that $F(\pi_1) \circ h = p_1$, $F(\pi_2) \circ h = p_2$. This means the commutativity of the following diagram:



Consider $r \in P'$. It follows that $p_1(r) \in F(S), p_2(r) \in F(Q)$. Denote by $p_j^i(r)(a)$ the i-th component of $p_j(r)(a)$. From $F(f) \circ p_1 = F(g) \circ p_2$ we obtain $F(f)(p_1(r)) = F(g)(p_2(r))$, and so $\left\langle f, id_{\mathbb{N}\langle O\rangle} \right\rangle \circ p_1(r) = \left\langle g, id_{\mathbb{N}\langle O\rangle} \right\rangle \circ p_2(r)$. This means that for all $a \in \mathbb{N}\langle V\rangle$, we have $(\left\langle f, id_{\mathbb{N}\langle O\rangle} \right\rangle \circ p_1(r))(a) = (\left\langle g, id_{\mathbb{N}\langle O\rangle} \right\rangle \circ p_2(r))(a)$, and so $(f(p_1^1(r)(a)), p_1^2(r)(a)) = (g(p_2^1(r)(a)), p_2^2(r)(a))$. This later equality lead us to $f(p_1^1(r)(a)) = g(p_2^1(r)(a))$ and $p_1^2(r)(a) = p_2^2(r)(a)$. Since P contains all the pairs that have the same image under f and g, we have $(p_1^1(r)(a), p_2^1(r)(a)) \in P$. This enables us to say that $(p_1(r), p_2(r)) \in F(P)$.

The so-called "mediating morphism" that we need for the universality property is $h : P' \to F(P)$ defined by $h(r) = (p_1(r), p_2(r))$. We check now that it satisfies the commutativity of the diagram. $(F(\pi_1) \circ h)(r) = F(\pi_1)(h(r)) = \left\langle \pi_1, id_{\mathbb{N}\langle O\rangle} \right\rangle \circ h(r) = \left\langle \pi_1, id_{\mathbb{N}\langle O\rangle} \right\rangle \circ (p_1(r), p_2(r)) = p_1(r)$ for all r from P'.

Similarly, it can be proved that $F(\pi_2) \circ h = p_2$.

Theorem 9 *The functor* $(- \times \mathbb{N}\langle O \rangle)^{\mathbb{N}\langle V \rangle} : Set \to Set$ *preserves weak pullbacks.*

Combining this last theorem with the results of Section 5 in [12], we get the following result:

Theorem 10 *Let* $(S, \alpha_S), (T, \alpha_T), (Q, \alpha_Q)$ *be three coalgebras associated to the functor* $(- \times \mathbb{N}\langle O \rangle)^{\mathbb{N}\langle V \rangle} : Set \to Set$*. The following assertions are true:*

1. *The diagonal* Δ_S *of a system* S *is a bisimulation.*
2. *Let* (R, α_R) *be a bisimulation between systems* S *and* T*. The inverse* R^{-1} *of* R *is a bisimulation between* T *and* S*.*
3. *The composition* $R \circ R'$ *of two bisimulations* $R \subseteq S \times T$ *and* $R' \subseteq T \times Q$ *is a bisimulation between* S *and* Q*.*
4. *The union* $\bigcup_k R_k$ *of a family* $\{R_k\}_k$ *of bisimulations between systems* S *and* T *is again a bisimulation. In particular, the greatest bisimulation between* S *and* T *exists, and it is the union of all bisimulations.*
5. *The kernel* $K(f)$ *of a homomorphism* $f : S \to T$ *is a bisimulation equivalence.*
6. *Let* $f : S \to T$ *be a homomorphism. If* $R \subseteq S \times S$ *is a bisimulation on* S*, then* $f(R)$ *is a bisimulation on* T*.If* $R' \subseteq T \times T$ *is a bisimulation on* T*, then* $f^{-1}(R')$ *is a bisimulation on* S*.*

6 Conclusion and Related Work

The proposal of this paper is to present a class of automata able to work with resources represented by multisets. Roughly speaking, a Mealy multiset automata is a machine able to consume and produce multisets. Mealy multiset automata could be related to the multiset automata presented in [5]as a particular accepting MmA having a "two letters" output alphabet. While [5] deals with multiset grammars and Chomsky hierarchy, we are mainly interested in algebraic, categorial and coalgebraic properties, emphasizing on their bisimulations, observation, and behaviour. The results presented in this paper guarantee useful properties of our natural computing automata, including that the (relational) product of bisimulations is a bisimulation, the largest bisimulation is an equivalence relation, and kernels of homomorphisms are always bisimulations. The link between MmA and P systems

is initiated in [3], where the description of an elementary Mealy membrane automata is based on Mealy multiset automata (see also [4]).

We give here only the first results regarding an algebraic and categorical approach of Mealy multiset automata, and we begin to develop the instruments for a further approach of molecular computing based on this notions. The algebraic constructions are useful for defining and operating with notions like bisimulation, observability and behaviour.

We have used elementary categorical language to model Mealy multiset automata and their behaviour. The idea of this approach is that a categorical formal language which is rich enough to describe and analyze various aspects of complex systems should be applicable to membrane systems. The first paper in category theory was by Eilenberg and Mac Lane in 1945 [8]. It aimed to describe (i) interaction and comparison within a given context (topological spaces, groups, other algebraic structures, etc.) and (ii) interactions between different contexts, for instance within the area of pure mathematics known as algebraic topology, problems in the theory of spaces are attacked by assigning various types of algebraic requirements to spaces, thus translating the topological problem to a more tractable algebraic one. Some links of category theory with the modelling of biological systems are briefly explored in [9]. Some categorical aspects of the theory of hierarchical systems are presented in [6].

References

[1] P. Aczel, N. Mendler: A final coalgebra theorem. *Lecture Notes in Computer Science* **389**, Springer (1989), 357–365.

[2] M. Barr, C. Wells: Category Theory Lecture Notes for ESSLLI. Available on-line at http://www.let.uu.nl/esslli/Courses/barr/barrwells.ps.

[3] G. Ciobanu, M. Gontineac: Mealy membrane automata and P systems complexity. In: M.A.Gutiérrez-Naranjo, Gh.Păun, M.J. Pérez-Jiménez (Eds.): *Cellular Computing. Complexity Aspects*. Fénix Editora, Sevilla (2005), 149–164.

[4] G. Ciobanu, M. Gontineac: Mealy Multiset automata. *International Journal of Foundations of Computer Science*, to appear 2006.

[5] E. Csuhaj-Varju, C. Martin-Vide, V. Mitrana: Multiset Automata. In: Multiset Processing. *Lecture Notes in Computer Science* **2235**, Springer (2001), 69–83.

[6] A.C. Ehresmann, J.-P. Vanbremersch: Hierarchical Evolutive Systems: A mathematical model for complex systems. *Bull. of Math. Biol.* **49** (1) (1987), 13–50.

[7] S. Eilenberg. *Automata. Languages and Machines*, Vol. A. Academic Press (1976).

[8] S. Eilenberg, S. MacLane: The general theory of natural equivalences. *Trans. Amer. Math. Soc.* **58** (1945), 231–294.

[9] M.J. Fisher, G. Malcolm, R.C. Paton: Spatio-logical processes in intracellular signalling. *Biosystems* **55** (2000), 83–92.

[10] W. Kuich, A. Salomaa: *Semirings, Automata, Languages*. Springer (1986).

[11] Gh. Păun: *Membrane Computing: An Introduction*. Springer (2002).

[12] J. Rutten: Universal coalgebra: a theory of systems. *Theoretical Computer Science* **249** (2000), 3–80.

On Symport/Antiport Systems and Semilinear Sets

Oscar H. IBARRA[1]*, Sara WOODWORTH[1], Hsu-Chun YEN[2], Zhe DANG[3]

[1]Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
E-mail: {ibarra,swood}@cs.ucsb.edu

[2]Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan 106, R.O.C.
E-mail: yen@cc.ee.ntu.edu.tw

[3]School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164, USA
E-mail: zdang@eecs.wsu.edu

Abstract

We introduce some restricted models of symport/antiport P systems that are used as acceptors (respectively, generators) of sets of tuples of nonnegative integers and show that they characterize precisely the semilinear sets. Specifically, we prove that a set $R \subseteq N^k$ is accepted (respectively, generated) by a restricted system if and only if R is a semilinear set. We also show that "slight" extensions of the models will allow them to accept (respectively, generate) nonsemilinear sets. In fact, for these extensions, the emptiness problem is undecidable.

1 Introduction

A general problem of clear interest in the area of membrane computing or P systems is to find classes of nonuniversal P systems that correspond

*Corresponding author.

to (i.e., characterize) known families of languages or subsets of N^k (where N is the set of nonnegative integers, and k is a positive integer), and to investigate their closure and decidability properties. For example, P system characterizations of ET0L, bounded languages accepted by multihead finite automata, and context-sensitive languages are known (see, e.g., [6, 8, 5, 1]). Here, we give characterizations of semilinear sets in terms of restricted models of symport/antiport systems.

A popular model of a P system is the symport/antiport system first introduced in [9]. It is a system whose rules closely resemble the way membranes transport objects between themselves in a purely communicating manner. Symport/antiport systems (SA systems) have rules of the form (u, out), (v, in), and $(u, out; v, in)$ where u, v are multisets that are represented as strings (the order in which the symbols are written is not important, since we are only interested in the multiplicities of each symbol). A rule of the form (u, out) in membrane i sends the elements of u from membrane i out to the membrane (directly) containing i. A rule of the form (v, in) in membrane i transports the elements of v into membrane i from the membrane enclosing i. Hence this rule can only be used when the elements of v exist in the outer membrane. A rule of the form $(u, out; v, in)$ simultaneously sends u out of the membrane i while transporting v into membrane i. Hence this rule cannot be applied unless membrane i contains the elements in u and the membrane surrounding i contains the elements in v. The rules are applied in a nondeterministic maximally parallel manner. In general, the number of times a particular rule is applied at anyone step can be unbounded. We require that the application of the rules is maximal: all objects, from all membranes, which *can be* the subject of local evolution rules *have to* evolve simultaneously. Note that there may be several maximal multisets of rules applicable in a step, but we nondeterministically select only one such multiset to apply.

Formally an SA system is defined as

$$M = (V, H, \mu, w_1, \ldots, w_{|H|}, E, R_1, \ldots, R_{|H|}, i_o)$$

where V is the set of objects (symbols) the system uses. H is the set of membrane labels. The membrane structure of the system is defined in μ. The initial multiset of objects within membrane i is represented by w_i, and the rules are given in the set R_i . E is the set of objects which can be found within the environment, and i_o is the designated output membrane. (When the system is used as a recognizer or acceptor, there is no need to specify i_o.) A large number of papers have been written concerning symport/antiport

systems. For example, it has been shown that "minimal" such systems (with respect to the number of membranes, the number of objects, the maximum "size" of the rules) are universal in the sense that they can simulate the computation of Turing machines or, equivalently, counter machines. See the P system website at http://psystems.disco.unimib.it for papers in symport/antiport systems and in the general area of membrane computing, and in particular the monograph [10]. In this paper, we introduce restricted models of symport/antiport systems that are used as acceptors or generators of sets of tuples of nonnegative integers and show that they characterize exactly the semilinear sets.

First, we look at systems that are acceptors. One model is called *simple SA*. The system consists of $k+1$ membranes, arranged in a 2-level structure: membranes $m_1, m_2, .., m_k$ (the *input membranes*) are at the same level and enclosed in membrane m_{k+1} (the *skin membrane*). The set of objects is $V = F \cup \{o\}$, where F is a *finite* set of objects not containing the distinguished symbol o. The restriction is that in the rules of the forms (v, in) and $(u, out; v, in)$, v does not contain o's. Thus, the number of o's in each membrane can only be decreased. The environment initially contains a fixed (finite) multiset over F. The system accepts a k-tuple $(n_1, ..., n_k)$ of nonnegative integers if, when the k input membranes are given $o^{n_1}, ..., o^{n_k}$ and no o's in membrane m_{k+1} (with some fixed strings $w_1, ..., w_{k+1} \in F^*$ in membranes $m_1, ..., m_{k+1}$, respectively), the system halts (i.e., no rule in any of the membranes is applicable). We show that a set $R \subseteq N^k$ is accepted by a simple SA if and only if it is a semilinear set. (This result generalizes to the case when there is an infinite supply of o's in the environment, and the v's can contain o's in the rules in the skin membrane m_{k+1}.) As a consequence, the class of sets of tuples accepted by these SAs are closed under union, intersection, and complementation. Moreover, the emptiness, disjointness, containment, and equivalence problems for simple SAs are decidable. When the model is generalized to a multi-level structure, the set of tuples accepted need no longer be semilinear. In particular, suppose we have a k-membrane SA, where membrane m_i is enclosed in membrane m_{i+1} for $1 \leq i \leq k-1$. Membrane m_1 is the only input membrane and membrane m_k is the skin membrane. Again, in the rules (v, in) and $(u, out; v, in)$, v does not contain o's. We call this model a *k-membrane cascade SA*. Note that the system accepts a subset of N. We show that 3-membrane cascade SAs can accept nonsemilinear subsets of N. We also prove that their emptiness problem is undecidable by showing that they can simulate the computations of two-counter machines.

The k-membrane cascade SA can be generalized. A *k-membrane ex-*

tended cascade SA has a set of objects $V = F \cup \Sigma_r$, where now the input alphabet is $\Sigma_r = \{a_1, ..., a_r\}$ $(r \geq 1)$. Again the rules are restricted in that in the rules of the forms (v, in) and $(u, out; v, in)$, v does not contain any symbol in Σ_r. The environment initially contains only a fixed multiset over F. Also, there are fixed strings $w_1, ..., w_k \in F^*$ such that the system starts with $w_1 a_1^{n_1} ... a_r^{n_r}$ in membrane m_1 (the input membrane) and w_i in membrane m_i for $2 \leq i \leq k$. If the system halts, then we say that the r-tuple $(n_1, ..., n_r)$ is accepted. We show that a set $R \subseteq N^r$ is accepted by a 1-membrane extended cascade SA if and only if it is semilinear. However, 2-membrane extended cascade SAs can accept nonsemilinear sets, and their emptiness problem is undecidable, even for $r = 2$ (i.e., there are two symbols in the input alphabet). Note that for the case $r = 1$ (i.e., Σ contains only a single symbol), the set of unary numbers is semilinear (since this is a special case of the result above for 2-level simple SA).

We then consider symport/antiport models that are used as generators. One such model is a 2-level symport/antiport system with membranes $m_1, ..., m_k, m_{k+1}$, where membranes $m_1, ..., m_k$ are at the same level, and they are enclosed in the skin membrane m_{k+1}. There is an infinite supply of o's in the environment (but the initial multiplicities of symbols in F in the environment are fixed). We require that for membranes $m_1, ..., m_k$, in the rules of the forms (u, out) and (u, out, v, in), u does not contain o's. Note that there is no restriction on the rules in the skin membrane. We say that $(n_1, ..., n_k)$ is generated if, when started with no o's in the system and fixed $w_i \in F^*$ in membrane m_i $(1 \leq i \leq k+1)$, the system halts with $o^{n_1}, ..., o^{n_k}$ in membranes $m_1, ..., m_k$. We call this system a *simple SA generator*. We show that a set $R \subseteq N^k$ is generated by a simple SA generator if and only if R is a semilinear set. Again, generalizing the model to have at least 3 levels would allow it to generate a nonsemilinear set. In fact, for any recursively enumerable (RE) set R, the set $\{2^n \mid n \in R\}$ can be accepted by a 3-level system, while R can be accepted by a 4-level system.

We also look at a 1-membrane symport/antiport system with a set of objects $V = F \cup \Sigma_r$, where $\Sigma_r = \{a_1, ..., a_r\}$, and whose rules are restricted so that in the rules of the forms (u, out) and $(u, out; v, in)$, u does not contain any symbol in Σ_r. Thus symbols in Σ_r can only be transported from the environment into the membrane (note that, by the restriction, once these symbols enter the membrane, they remain in the membrane). The system starts with a fixed string $w \in F^*$. The environment initially contains a fixed multiset over F and an infinite supply of each a_i $(1 \leq i \leq r)$. We show that the sets of r-tuples generated by these systems are exactly the semilinear sets over N^r. However, when there are 2 membranes, where again, the

second (i.e., innermost) membrane cannot transport symbols in Σ_r into the first (skin) membrane, the set of tuples generated by such a system need not be semilinear. In fact, for any RE set R, the set $\{(2^n, 0) \mid n \in R\}$ can be generated by a 2-membrane system with input alphabet Σ_2, while the set $\{(n, 0, 0) \mid n \in R\}$ can be generated by a 2-membrane system with input alphabet Σ_3.

2 Restricted SA Acceptors and Semilinear Sets

We first introduce a restricted model of a symport/antiport system [9] which is used as an acceptor of tuples of nonnegative integers. A *simple SA* $\mathcal{P}$ is defined as follows:

1. The alphabet of objects is $V = F \cup \{o\}$, where F is a finite set and o is a distinguished object.

2. There are $k + 1$ membranes ($k \geq 1$) arranged in a 2-level structure: membranes $m_1, m_2, .., m_k$ (the *input membranes*) are at the same level and enclosed in membrane m_{k+1} (the *skin membrane*).

3. At the start of the computation the k input membranes are given $o^{n_1}, ..., o^{n_k}$, respectively, for some nonnegative integers $n_1, ..., n_k$ (the skin membrane initially does not contain any o).

4. Also, at the start of the computation, there are fixed strings, i.e., multisets $w_1, ..., w_{k+1} \in F^*$ in membranes $m_1, ..., m_{k+1}$, respectively. Thus, the w_i's do not contain any o.

5. The environment initially only contains a fixed (finite) multiset over F. Of course, symbols that are exported to the environment from the skin membrane during the computation can be retrieved from the environment.

6. Each membrane has a set R_i of rules (some may be empty) The rules are of the form:

   (a) (u, out)

   (b) (v, in)

   (c) $(u, out; v, in)$

where $u, v \in V^+$. Rule of type (a) transports multiset u from the membrane containing the rule into the surrounding membrane (if the membrane contains u). Rule of type (b) imports multiset v from the surrounding membrane into the membrane containing the rule (if the surrounding membrane contains v). Rule of type (c) simultaneously transports u to the surrounding membrane and imports v from the surrounding membrane (if the membrane contains u and the surrounding membrane contains v).

The restriction is:

In the rules of types (b) and (c), v **does not** contain o's. This just means that the number of o's in any membrane can only be decreased and cannot be increased.

7. As usual in a P system, the rules are applied in a nondeterministic maximally parallel manner.

Notice that the fixed multisets over F given initially in the membranes as well as in the environment are part of the specification of the simple SA $\mathcal{P}$ (which we do not always explicitly state). We say that a tuple $(n_1, ..., n_k)$ is accepted by $\mathcal{P}$ if, when the k input membranes are given $o^{n_1}, ..., o^{n_k}$ respectively, the system halts (i.e., none of the rules is applicable). The set of all such tuples is denoted by $R(\mathcal{P})$.

Simple SAs are intimately related to counter machines. Let $\mathcal{M}$ be a nondeterministic multicounter machine all of whose counters are reversal-bounded. A counter is reversal-bounded if the number of alternations between nondecreasing mode and nonincreasing mode during any computation is at most a fixed number. The first k counters are input counters. We say that $\mathcal{M}$ accepts $(n_1, ..., n_k)$ if, when started in its start state with counter i set to n_i $(1 \leq i \leq k)$ and the other counters to zero, $\mathcal{M}$ halts in an accepting state with all counters zero. The set of all such tuples accepted by $\mathcal{M}$ is denoted by $R(\mathcal{M})$. We call $\mathcal{M}$ a *reversal-bounded (multi) counter machine*.

A special case is a counter machine with only k counters (the input counters) each of whose counters can only be decremented. Moreover, at every step, the machine decrements exactly one counter. We call this machine a *decreasing counter machine*.

We can augment a reversal-bounded multicounter machine with an unrestricted counter, i.e., a free counter. This counter can make an unbounded number of reversals. We call such a machine *reversal-bounded counter machine with a free-counter*.

Convention: In our definition of counter machines above, acceptance is by "accepting state". Clearly, given a counter machine $\mathcal{M}$, we can easily construct an equivalent machine $\mathcal{M}'$ which accepts if and only if it eventually halts in some state (accepting or not). $\mathcal{M}'$ simulates $\mathcal{M}$ faithfully. If $\mathcal{M}$ enters an accepting state, then $\mathcal{M}'$ halts. If $\mathcal{M}$ halts in a rejecting state, say s, then $\mathcal{M}'$ goes into an infinite loop by executing the following (where c is some counter of the machine):

s : If counter c is nonzero then decrement c and go to state s else go to state s

The reason we need the second mode of acceptance is that in our constructions characterizing the different SA systems by counter machines, the equivalences are of the form: The SA halts (i.e., accepts) if and only if the machine halts. All the machines discussed in the paper can easily be converted to ones whose mode of acceptance is by halting. □

Next we recall the definition of a semilinear set [3]. Let N be the set of nonnegative integers and k be a positive integer. A subset R of N^k is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_t$ in N^k such that

$$R = \{v \mid v = v_0 + m_1 v_1 + \cdots + m_t v_t,\ m_i \in N\}.$$

The vectors v_0 (referred to as the *constant vector*) and $v_1, v_2, \ldots, v_t$ (referred to as the *periods*) are called the *generators* of the linear set R. The set $R \subseteq N^k$ is *semilinear* if it is a finite union of linear sets. The empty set is a trivial (semi)linear set, where the set of generators is empty. Every finite subset of N^k is semilinear – it is a finite union of linear sets whose generators are constant vectors. It is also clear that the semilinear sets are closed under (finite) union. It is also known that they are closed under complementation and intersection.

Theorem 1 *Let $R \subseteq N^k$. Then the following statements are equivalent:*

1. *R is a semilinear set.*
2. *R is accepted by a reversal-bounded counter machine with a free counter.*
3. *R is accepted by a reversal-bounded counter machine.*
4. *R is accepted by a decreasing counter machine.*

Proof. It is obvious that (4) implies (3) and (3) implies (2). From the definition of a semilinear set, it is easy to construct, given a semilinear set R, a decreasing counter machine $\mathcal{M}$ accepting R. Since $\mathcal{M}$ is nondeterministic, it sufficient to describe the construction of $\mathcal{M}$ when R is a linear set. So suppose, $R = \{v \mid v = v_0 + m_1 v_1 + \cdots + m_t v_t,\ m_i \in N\} \subseteq N^k$, with $v_i = (v_{i1}, ..., v_{ik})$ for $0 \leq i \leq t$. $\mathcal{M}$, when given $(n_1, ..., n_k)$ in its counters, applies the constant vector v_0 to decrement the counters simultaneously by $v_{01}, ..., v_{0k}$, respectively. Then $\mathcal{M}$ nondeterministically guesses the number of times m_i to apply v_i to the counters (again, decreasing the counters simultaneously by the amounts $m_i v_{i1}, ..., m_i v_{ik}$, respectively for $1 \leq i \leq t$. If all the counters become zero at the same time, $\mathcal{M}$ accepts. Thus, (1) implies (4). That (2) implies (1) is a trivial consequence of a result in [4], which showed that if a bounded language $L \subseteq a_1^* ... a_k^*$ (where $a_1, ..., a_k$ are distinct symbols and $n_1, ..., n_k$ are nonnegative integers) is accepted by a nondeterministic finite automaton augmented with reversal-bounded counters and one unrestricted counter, then the set $\{(n_1, ..., n_k) \mid a_1^{n_1} ... a_k^{n_k} \in L\}$ is semilinear. □

Lemma 2 *Let $\mathcal{P}$ be a simple SA. Then $R(\mathcal{P})$ can be accepted by a reversal-bounded counter machine with a free counter $\mathcal{M}$.*

Proof. We construct a counter machine $\mathcal{M}$ with $k+1$ counters to simulate $\mathcal{P}$. The intuitive idea behind the simulation is the following. The first k counters are reversal-bounded (the input counters) and the last is the free counter. Initially, the input counters are set to $n_1, ..., n_k$, respectively. The free counter will keep track of the current number of o's in the skin membrane (at the start, there is none). The initial configuration $(w_1, ..., w_k, w_{k+1})$ and the rules $(R_1, ..., R_{k+1})$ are stored in the finite-state control of $\mathcal{M}$. The finite-state control keeps track of the numbers of non-o symbols and their distributions within the membranes and the environment (this can be done since their total multiplicities remain the same (as ones initially given as fixed constants in the definition of $\mathcal{P}$) at any time, independent of the n_i's). $\mathcal{M}$ simulates each nondeterministic maximally parallel step of $\mathcal{P}$ by several moves. Clearly, because of the restrictions on the rules, the counters keeping track of the multiplicities of o's in the input membranes are only decremented. Special care has to be taken when simulating a rule of type either (u, out) or $(u, out; v, in)$ when u contains multiple copies of o's. In order to tell whether such a rule is applicable or not, for each membrane we associate a finite buffer of size d (where d is the maximum number of

o's that can be thrown out by a single rule in the membrane) to the finite control of M to keep track of the first d o's in the membrane while using the counter of M associated with the membrane to hold the number of the remaining o's. By doing so, checking whether the above rule is applicable can be done by examining the contents of the finite buffer associated with the membrane in which the rule resides.

Now in a maximally parallel step, some (possibly all) of the input membranes can transport o's to the skin membrane and the skin membrane itself can also transport some o's to the environment. However, the total number of o's transferred from the input membranes to the skin membrane and the total number of o's transferred from the skin membrane to the environment may have no relationship, so the free counter may be decremented and incremented an unbounded number of times during the computation. This is the reason why we need a free counter. It follows from the description that $\mathcal{M}$ can simulate the computation of $\mathcal{P}$. □

We now prove the converse of Lemma 2.

Lemma 3 *Let $\mathcal{M}$ be a reversal-bounded counter machine with a free counter. Then $R(\mathcal{M})$ can be accepted by a simple SA $\mathcal{P}$.*

Proof. By the proof of Theorem 1, we may assume that $\mathcal{M}$ is a decreasing counter machine with k counters accepting $R(\mathcal{M}) \subseteq N^k$. Thus $\mathcal{M}$, when started in its initial state with $n_1, ..., n_k$ in the counters halts in an accepting state if $(n_1, ..., n_k)$ is in $R(\mathcal{M})$. Moreover, at each step of the computation, before it halts, $\mathcal{M}$ decrements exactly one counter (there are no increments).

We will construct a simple SA $\mathcal{P}$ simulating $\mathcal{M}$. As defined, $\mathcal{P}$ will have a 2-level structure with k input membranes $m_1, ..., m_k$ (at the same level) enclosed by the skin membrane m_{k+1}. The k input membranes will keep track of the values of the counters. The construction of $\mathcal{P}$ follows the construction in [11] where a two-level SA system is shown to simulate a multicounter machine. In the construction, each of the inner membranes represents a counter and the multiplicity of the distinguished symbol o within each membrane represents the value of that counter. The rules associated with each subtract instruction in the construction adhere to the restrictions required by a simple SA system. Since $\mathcal{M}$ has no increment instructions, the associated $\mathcal{P}$, by the construction in [11], is a simple SA. We omit the details. □

From Theorem 1 and Lemmas 2 and 3, we have:

Theorem 4 *Let $R \subseteq N^k$. Then the following statements are equivalent:*

1. *R is a semilinear set.*

2. *R is accepted by a reversal-bounded counter machine with a free counter.*

3. *R is accepted by a reversal-bounded counter machine.*

4. *R is accepted by a decreasing counter machine.*

5. *R is accepted by a simple SA.*

Note that in a simple SA, the number of o's in the membranes cannot be increased, since in the rules of the form (v, in) and $(u, out; v, in)$, we do not allow v to contain o's. We can generalize the model. The environment can have an infinite supply of o's, and in the rules of the forms (v, in) and $(u, out; v, in)$ in the skin membrane, v is in F^+o^*. Thus v can contain o's but must contain at least one symbol in F. (We do not allow v to only contain o's since, otherwise, the system will not halt because there is an infinite supply of o's in the environment.) Thus the number of o's in the skin membrane can increase during the computation by importing o's from the environment. Call this model *simple SA^+*. Clearly the construction in Lemma 2 still works when $\mathcal{P}$ is a simple SA^+. The only modification is that in the simulation of a maximally parallel step of $\mathcal{P}$ by $\mathcal{M}$, we also need to consider the o's that may be brought into the skin membrane from the environment by the (v, in) and $(u, out; v, in)$ rules. Thus, we have:

Corollary 5 *Let $R \subseteq N^k$. Then the following statements are equivalent: items (1), (2), (3), (4), (5) of Theorem 4, and (6): R is accepted by a simple SA^+.*

The following corollary follows from known results concerning semilinear sets.

Corollary 6 *Let k be any positive integer. Then:*

1. *The class of subsets of N^k accepted by simple SAs is closed under union, intersection, and complementation.*

2. *The membership, disjointness, containment, and equivalence problems for simple SAs accepting subsets of N^k are decidable.*

3 Cascade Counter Machines and Cascade SAs

In this section, we will show that Theorem 4 does not generalize to the case when the simple SA has a 3-level structure. In particular, consider a simple SA with only three membranes m_1, m_2, m_3, where membrane m_1 is enclosed in m_2, and m_2 is enclosed in m_3 (the skin membrane). Initially, membrane m_1 contains the input o^n. The same restriction (i.e., in the rules of the forms (v, in) and $(u, out; v, in)$, v does contain o's) applies. We will show that such a system can accept a nonsemilinear set. In fact, the emptiness problem for such systems is undecidable. To facilitate the proofs, we first introduce the notion of cascade counter machines.

3.1 Cascade Counter Machines

A k-counter cascade machine $\mathcal{M}$ is a finite-state machine with k counters, $c_1, ..., c_k$. The instructions of $\mathcal{M}$ are of the following forms:

$s \to (s', c_i := c_i - 1; c_{i+1} := c_{i+1} + 1)$ (decrement c_i then increment c_{i+1})
$s \to (s'$ if c_i is zero else $s'')$ (test if $c_i = 0$)
$s \to (s', c_k := c_k - 1)$ (counter c_k can be independently decremented)

Notice that in the above, it is implicit that $\mathcal{M}$ cannot increment c_1 (there is no such instruction). We say that a nonnegative integer n is accepted if $\mathcal{M}$, when started in its initial state with counter values $(n, 0, ..., 0)$ eventually enters an accepting state.

We first show that the emptiness problem for deterministic 3-counter cascade machines is undecidable by showing how a 3-counter cascade machine with initial counter values $(n, 0, 0)$ can simulate the computation of a deterministic (unrestricted) 2-counter machine with intial counter values $(0, 0)$. The former accepts accepts some n if and only if the latter halts. The result then follows from the undecidability of the halting problem for 2-counter machines [7].

So suppose that $\mathcal{M}$ is a deterministic (unrestricted) 2-counter machine. We show that $\mathcal{M}$ can be simulated by a deterministic 3-counter cascade machine $\mathcal{M}'$ with counters c_1, c_2, c_3. The two counters x_1 and x_2 of $\mathcal{M}$ are simulated by c_2 and c_3 of $\mathcal{M}'$, respectively. Clearly, testing if counter x_i is zero for $i = 1, 2$ can be directly simulated in $\mathcal{M}'$. Incrementing/decrementing counters x_1 and x_2 of $\mathcal{M}$ can also be simulated in $\mathcal{M}'$:

1. When $\mathcal{M}$ increments x_1, $\mathcal{M}'$ performs the following: Decrement c_1, increment c_2.

2. When $\mathcal{M}$ increments x_2, $\mathcal{M}'$ performs the following: Decrement c_1, increment c_2, decrement c_2, increment c_3.

3. When $\mathcal{M}$ decrements x_1, $\mathcal{M}'$ performs the following: Decrement c_2, increment c_3, decrement c_3.

4. When $\mathcal{M}$ decrements x_2, $\mathcal{M}'$ also decrements c_3.

During the simulation, if c_1 is zero when an instruction being simulated calls for decrementing c_1, $\mathcal{M}'$ rejects. Note that all state transitions in $\mathcal{M}$ are simulated faithfully by $\mathcal{M}'$. It follows that we can construct $\mathcal{M}'$ so that it accepts the input n (initially given in c_1) if and only if n is "big" enough to allow the simulation of $\mathcal{M}$ to completion. If $\mathcal{M}$ does not halt or n is not big enough to carry out the simulation (at some point), $\mathcal{M}'$ goes into an infinite loop or rejects. It follows that the emptiness problem for deterministic 3-counter cascade machines is undecidable.

Example. We now give an example of a deterministic 3-counter cascade machine $\mathcal{M}$ accepting a nonsemilinear set. Starting with $c_1 = n, c_2 = 0, c_3 = 0$,

1. If c_1 is zero, $\mathcal{M}$ rejects.

2. $\mathcal{M}$ configures the counters to contain: $c_1 = n - 1, c_2 = 0, c_3 = 1$.

3. If c_1 is zero, $\mathcal{M}$ accepts.

4. Set $k = 1$.

5. Starting with values: $c_1 = n-(1+3+...+(2k-1)), c_2 = 0, c_3 = (2k-1)$,

   (*) $\mathcal{M}$ iteratively decrements c_3 by 1 while decrementing c_1 by 1 and incrementing c_2 by 1 until $c_3 = 0$. Then $\mathcal{M}$ decrements c_1 by 2 and increments c_2 by 2 (this is done in two steps). After that, $\mathcal{M}$ iteratively decrements c_2 by 1 while incrementing c_3 by 1 until $c_2 = 0$.
   – If c_1 becomes zero before the completion of (*), $\mathcal{M}$ rejects.
   – If $c_1 = 0$ after the completion of (*), $\mathcal{M}$ accepts, else $\mathcal{M}$ sets $k := k + 1$ and goes back to (*).

Clearly, the values of the counters when k becomes $k + 1$ are: $c_1 = n - (1 + 3 + ... + (2k - 1) + (2k + 1)) = n - (k + 1)^2, c_2 = 0, c_3 = (2k + 1)$. It follows that M can be constructed to accept the set $\{n^2 \mid n \geq 1\}$, which is not semilinear. □

From the above discussion and example, we have:

Theorem 7 *Deterministic 3-counter cascade machines can accept nonsemilinear sets. Moreover, their emptiness problem is undecidable.*

Remark 1. The construction of the deterministic 3-counter cascade machine in the example above can be modified to accept the set $R_1 = \{2n^2 \mid n \geq 1\}$. Now define for eack $k \geq 1$, the set $R_k = \{2n^{2^k} \mid n \geq 1\}$. One can show by essentially iterating the construction in the example that R_k can be accepted by a deterministic $(2+k)$-counter cascade machine. We believe (but have no proof at this time), that the R_k's form an infinite hierarchy: R_{k+1} can be accepted by a deterministic $(2+k)$-counter cascade machine but cannot be accepted by any deterministic or nondeterministic $(2+(k-1))$-counter cascade machine. Note that 1- and 2- counter cascade machines are equivalent — both accept exactly the semilinear sets. □

It is interesting to observe that for a k-counter cascade machine $\mathcal{M}$, if counter c_1 cannot be tested for zero, then either $R(\mathcal{M}) = \emptyset$ (if $\mathcal{M}$ never enters an accepting state regardless of the input initially given in c_1) or there exists an $m \in N$ such that $R(\mathcal{M}) = \{n \mid n \geq m, n \in N\}$ (m is the smallest input for which $\mathcal{M}$ accepts). Hence, for cascade counter machines lacking the capability of testing counter c_1 for zero, they accept only semilinear sets. The emptiness problem, nevertheless, remains undecidable for such a restricted class of cascade counter machines, implying that the semilinear sets associated with such machines are not effective.

We conclude this section by noting that Theorem 7 is not true for (deterministic or nondeterministic) 2-counter cascade machines. In fact, consider a nondeterministic machine $\mathcal{M}$ which has $k+1$ counters, where the first k counters are initially set to input values $n_1, ..., n_k$, respectively, and the last counter set to zero. The computation is restricted in that the first k counters can only be decremented, but the last counter can be decremented/incremented independently. It follows from Theorem 1 that these machines accept exactly the semiliner sets.

3.2 Cascade SAs

A cascade SA has k membranes $m_1, ..., m_k$ (for some k) that are nested: For $1 \leq i \leq k-1$, membrane m_i is enclosed in membrane m_{i+1}. The input membrane, m_1, initially contains o^n for some n. Again, in the rules of the forms (v, in) and $(u, out; v, in)$, v does not contain o's. There are fixed

multisets $w_1, ..., w_k$ not containing o's in membranes $m_1, ..., m_k$ initially. The environment initially contains a fixed multiset of symbols.

The connection between cascade counter machines and cascade SAs is given by the following theorem.

Theorem 8 *Let $k \geq 1$ be a positive integer. A set $Q \subseteq N$ is accepted by a k-membrane cascade SA if and only if it can be accepted by a k-counter cascade machine.*

Proof. Let $\mathcal{P}$ be a k-membrane cascade SA. We construct an equivalent k-counter cascade machine $\mathcal{M}$. We associate a counter c_i for every membrane m_i to keep track of the number of o's in membrane m_i during the computation. The construction of $\mathcal{M}$ simulating $\mathcal{P}$ is straightforward, following the strategy in the proof of Lemma 2.

We now prove the converse. Let $\mathcal{M}$ be a k-counter cascade machine. For notational convenience we will assume the program instructions for $\mathcal{M}$ are labeled $l_0, l_1, \ldots, l_n$ and begin with instruction l_0. We also assume they are written in the form $l_i : (SUB(r), l_s, l_t)$ meaning that when instruction l_i is executed, counter r is decremented. If counter r was initially positive (meaning it was able to be decremented), the machine will next execute the instruction l_s, otherwise it will execute the instruction l_t. Also, since $\mathcal{M}$ is a cascade counter machine, each decrement from counter r where $r < k$ must be followed by an instruction which increments the counter $r + 1$. Hence, we can incorporate each increment instruction into its preceding decrement instruction. (In the case where we decrement counter r and $r = k$, no increment instruction follows since the decremented value is thrown out of the system.) In this way we can consider the program for $\mathcal{M}$ to consist entirely of decrement instructions. We now construct an equivalent k-membrane cascade SA $\mathcal{P}$ which simulates each decrement instruction of $\mathcal{M}$. The membrane structure of $\mathcal{P}$ is a set of nested membranes which each correspond to a counter in $\mathcal{M}$. The skin membrane also acts as program control membrane. Formally, the simulation occurs by creating the following cascade SA membrane system from a given cascade counter machine:

$$\mathcal{P} = (V, H, \mu, w_{m_1}, \ldots, w_{m_k}, E, R_{m_1}, \ldots, R_{m_k})$$

where

$$\begin{aligned} V = \{&l_{i1}, l_{i2}, l_{i3}, l_{i4}, d_{ij} | l_i \text{ is an instruction label of the form} \\ &l_i : (SUB(r), l_s, l_t) \text{ where } r \neq k \text{ and } 0 \leq j \leq 2(k-r)+1\} \cup \\ &\{l_{i1}, l_{i2}, l_{i3} | l_i \text{ is an instruction label of the form} \end{aligned}$$

$l_i : (SUB(r), l_j, l_s)$ where $r = k\} \cup$
$\{d_0, d_1\} \cup$
$\{c, c', c_1, \ldots, c_m\} \cup$
$\{o\}$
$H = \{m_1, m_2, \ldots, m_k\}$
$\mu = [_{m_k}[_{m_{k-1}} \cdots [_{m_1}]_{m_1} \cdots]_{m_{k-1}}]_{m_k}$
$w_{m_1} = c_1 o^n$
$w_{m_i} = c_i$ for all $1 < i < k$
$w_{m_k} = l_{01} c_k$ (since l_0 is the first instruction to execute)
$E =$ one copy of each element in V except o and l_{01}

The rule sets $(R_{m_1}, \ldots, R_{m_k})$ are created based on the cascade machine's program. Initially we create the rule $(d_0, out; d_1, in)$ within R_{m_k}. For each rule of the form $l_i : (SUB(r), l_s, l_t)$ where $r \neq k$ we add the following rules:

1. R_{m_k} contains:
   (a) $(l_{i1}, out; l_{i2} c d_0 d_{i0}, in)$
   (b) $(d_{ij}, out; d_{i(j+1)}, in)$ where $0 \leq j \leq 2(k - r)$
   (c) $(d_1 d_{i[2(k-r)+1]}, out; l_{i4} c', in)$
   (d) $(l_{i2} d_1, out; l_{i3}, in)$
   (e) $(l_{i3} c d_{i[2(k-r)+1]}, out; l_{s1}, in)$
   (f) $(l_{i2} l_{i4}, out; l_{t1}, in)$
   (g) (cc', out)
2. R_{m_n} where $k \geq n > r$ contains:
   (a) $(l_{i2} c, in)$
   (b) $(l_{i2} c_r, out)$
   (c) $(l_{i3} c_r, in)$
   (d) $(l_{i3} c, out)$
   (e) $(l_{i4} c', in)$
   (f) $(l_{i2} l_{i4}, out)$
   (g) (cc', out)
3. R_{m_r} contains:
   (a) $(c_r, out; l_{i2} c, in)$

(b) $(l_{i2}o, out)$
(c) $(l_{i3}c_r, in)$
(d) $(l_{i3}c, out)$
(e) $(l_{i2}, out; c_r c', in)$
(f) (cc', out)

For a rule of the form $l_i : (SUB(r), l_s, l_t)$ where $r = k$ we create the following rules:

4. $R_{m_r} = R_{m_k}$ contains:

(a) $(l_{i1}, out; l_{i2}d_0, in)$
(b) $(l_{i2}o, out; l_{i3}, in)$
(c) $(l_{i3}d_1, out; l_{s1}, in)$
(d) $(l_{i2}d_1, out; l_{t1}, in)$

Informally, the above simulation operates as follows. The process of simulating a single subtract instruction $l_i : (SUB(r), l_s, l_t)$ if $r \neq k$ begins by the presence of the object l_{i1} within the outermost membrane (m_k). This object is used to bring in the necessary execution objects l_{i2}, c, d_0, and d_{i0} using rule 1a. The objects l_{i2} and c are used cooperatively and are drawn deeper through the membrane hierarchy until they have reached the membrane m_{r+1}. Here they are drawn into membrane m_r while the object c_r is thrown out.

If membrane m_r contains an o object (meaning counter r is not empty) the objects l_{i2} and o are thrown out into membrane m_{r+1}. This simulates both the current subtract instruction along with the add instruction we know must follow. Now, the objects l_{i2} and c_r are used cooperatively and are thrown out of each membrane until they located in the skin membrane.

While this has been occurring, the delay objects in the skin membrane have been being incremented. The d objects are delay objects and are used to delay certain execution steps. During each step of computation, their subscripts are incremented by one. The object d_0 only changes to d_1 to delay an action for a single step while the object d_{i0} increments to $d_{i[2(k-r)+1]}$. This number $(2(k-r)+1)$ corresponds to the number of steps plus one that l_{i2} will take to travel to membrane r and back if membrane r contains a o. This allows us to determine whether the object l_{i2} is stuck in membrane r.

If the membrane m_r contains an o (meaning counter r is not zero), objects l_{i2} and c_r will return to the skin membrane in $2(k-r)$ steps and

rule 1d is applicable before $d_{i[2(k-r)+1]}$ has been brought into the membrane. So, l_{i2} and d_2 are thrown out into the environment and object l_{i3} is brought into the system. Now the objects c and c_r must be swapped to their original positions. This occurs by having objects l_{i3} and c_r work cooperatively to move deeper through the membranes to membrane r and then objects l_{i3} and c work cooperatively to be thrown out of each membrane until they return to the skin membrane. At this point, everything is completed and all objects are in the correct location. So objects l_{i3}, c, and $d_{i[2(k-r)+1]}$ are thrown out into the environment while object l_{s1} is brought in. At this point, instruction l_i is complete and instruction l_s will execute next.

If the objects l_{i2} and c_r have not returned to the skin membrane after $2(k-r)+1$ steps, then the membrane r must not have contained an o. At this point, the objects d_1 and $d_{i[2(k-r)+1]}$ are thrown out of the skin membrane and objects l_{i4} and c' are brought in. Now, objects l_{i4} and c' work cooperatively to move deeper through the membranes to membrane m_{r+1}. Object c' is drawn into membrane m_r while object l_{i2} is thrown out. At this point, membrane m_r contains the objects c and c' while membrane m_{r+1} contains the objects l_{i2} and l_{i4}. These pairs of objects work cooperatively to be thrown out of each membrane. The pair $l_{i2}l_{i4}$ will get to the skin membrane a step ahead of the pair cc'. The objects l_{i2} and l_{i4} are thrown out into the environment while bringing in the object l_{t1}. During the next step the pair cc' will be thrown out into the environment. At this point, instruction l_i is complete and instruction l_t will execute next.

If the instruction to be simulated is of the form $l_i : (SUB(r), l_s, l_t)$ where $r = k$, the simulation is much simpler. In this case, since the instruction is immediately placed within the counter membrane, only a single delay object is needed along with the instruction object l_{i2}. If membrane k contains an o, it is thrown out during the next step along with the object l_{i2} and the object l_{i3} is brought in allowing the final step to clean up and bring in the instruction object l_{s1}. If l_{i2} is still in membrane m after one step, the delay object can cooperate with object l_{i2} to bring in the next instruction object l_{t1}.

Consequently, these cascade SA rules simulate the operation of M. □

From Theorems 7 and 8, we have,

Corollary 9 *3-membrane cascade SAs can accept nonsemilinear sets. Moreover, their emptiness problem is undecidable.*

A careful examination of the proof of Theorem 8 reveals that the degree of maximal parallelism for the constructed SA is finite (i.e., at every step

of the computation, the size of the multiset of applicable rules is bounded by some fixed integer). Hence, Corollary 9 holds even if the 3-membrane cascade SAs have bounded degree of maximal parallelism.

4 Another SA Acceptor Characterizing the Semilinear Sets

The k-membrane cascade SA of the previous section can be generalized. A *k-membrane extended cascade SA* has a set of objects $V = F \cup \Sigma_r$, where now the input alphabet is $\Sigma_r = \{a_1, ..., a_r\}$ ($r \geq 1$). Again the rules are restricted in that in the rules of the forms (v, in) and $(u, out; v, in)$, v does not contain any symbol in Σ_r. The environment initially contains only F. There are fixed strings $w_i \in F^*$ such that the system starts with $w_1 a_1^{n_1} ... a_r^{n_r}$ in membrane m_1 (the input membrane) and w_i in membrane m_i for $2 \leq i \leq k$. If the system halts, then we say that the r-tuple $(n_1, ..., n_r)$ is accepted.

Next consider a finite-state device $\mathcal{M}$ with a finite-state control and a "bag" that can hold a multiset of symbols. $\mathcal{M}$ starts in its initial state with the bag containing a multiset $a_1^{n_1} ... a_r^{n_r}$. The instructions of $\mathcal{M}$ are of the following form:

$q \rightarrow (q'$ delete a_i from the bag if it is in the bag else $q'')$

Thus, from state q, $\mathcal{M}$ removes a_i from the bag if it is in the bag and goes to state q'; otherwise, $\mathcal{M}$ goes to state q''. The initial multiset in the bag is accepted if $\mathcal{M}$ enters an accepting state. We call this device a *1-bag automaton*. A 1-bag automaton is like a multiset automaton studied in [2]. Although the notion is not the same, the idea is quite similar.

We can generalize the 1-bag automaton to a *k-bag automaton*, where now, a symbol is deleted from bag i if and only if it is exported into bag $i + 1$. A symbol can be deleted from the k-th bag independently.

Lemma 10 *A set $R \subseteq N^r$ is accepted by a 1-bag automaton if and only if it is accepted by a decreasing r-counter machine.*

Proof. Clearly, deleting a_i from the bag corresponds to decrementing counter i ($1 \leq i \leq r$). □

Theorem 11 *Let $k \geq 1$. A set of tuples R is accepted by a k-membrane extended cascade SA if and only if R is accepted by a k-bag automaton.*

Proof. The proof for the "only if" part is a straightforward generalization of the proof of the first part of Theorem 1 (which was for $k+1$). For the second part, let $\mathcal{M}$ be a k-bag automaton. We construct a k-membrane extended cascade SA $\mathcal{P}$ equivalent to $\mathcal{M}$, in the same manner as the construction of Theorem 8 where each membrane corresponds to a bag. The rules can be created by mapping each subtraction rule of the form $l_i : (SUB(r), l_s, l_t)$ to the bag rule of the form $q \rightarrow (q'$ delete a_i from the bag if it is in the bag else $q'')$ as follows. The instruction labels of a counter machine can also be viewed as states so we can say l_i corresponds to q, l_s corresponds to q', and l_t corresponds to q''. The bag associated with q corresponds to the counter r. The additional difference is that the bag also specifies the object (a_i) in Σ which should be thrown out of the bag. Hence, we can create $\mathcal{P}$ to simulate $\mathcal{M}$ using the techniques in Theorem 8 and the above mapping along with the following changes. The set V will now additionally contain the set of objects $\{a_1, \ldots, a_r\}$ rather than the single object $\{o\}$. The set $w_{m_1} = c_1 a_1^{n_1} ... a_r^{n_r}$ rather than $c_1 o^n$. Also, the rules 3b: $(l_{i2} o, out)$ and 4b: $(l_{i2} o, out; l_{i3}, in)$ will be changed to $(q_2 a_i, out)$ and $(q_2 a_i, out; s_3, in)$ respectively. Clearly this k-membrane extended cascade SA now simulates a k-bag automaton. □

Theorem 12 *A set $R \subseteq N^r$ is accepted by a 1-membrane extended cascade SA if and only if it is a semilinear set.*

Proof. Let $\mathcal{P}$ be a 1-membrane extended cascade SA with input alphabet Σ_r. We can easily construct a decreasing r-counter machine $\mathcal{M}$ which, when the counters are initially given $n_1, ..., n_r$, simulates the computation of $\mathcal{P}$ on $wa_1^{n_1} ... a_r^{n_r}$. The simulation is straightforward, as in Lemma 2. It follows from Theorem 1 that $R(\mathcal{P})$ is a semilinear set.

For the converse, by Lemma 10, we need only show that a 1-bag automaton can be simulated by a 1-membrane extended cascade SA. This follows from Theorem 11. □

Let $\Sigma_2 = \{a_1, a_2\}$. Using the ideas in the example of the previous section, we can easily construct a 2-bag automaton accepting the nonsemilinear set $\{(n_1, n_2) \mid n_1, n_2 \geq 0, n_1 + n_2 = m^2$ for some $m \geq 1\}$. It is also easy to construct a 2-bag automaton with input alphabet Σ_2 that simulates the computations of a 2-counter automaton. The values of the counters are represented in the second bag. The number of a_1's (resp., a_2's) in that bag denotes the value of the first (resp., second) counter. The a_1's and the a_2's in the first bag are the suppliers (sources) of the "increments" for the two counters in the second bag.

From the above discussion and Theorem 11, we have,

Theorem 13 *2-bag automata (and, hence, 2-membrane extended cascade SAs) can accept nonsemilinear sets. Moreover, their emptiness problem is undecidable.*

5 Restricted Symport/Antiport Systems as Generators

In this section, we look at symport/antiport systems used as generators of tuples. We only state the results; the proofs will be given in the full paper.

In the definition of a k-membrane cascade SA, the input o^n is initially given in m_1 (the innermost membrane) with no o's in the other membranes. The computation is such that the o's can only be exported from membrane m_i to membrane m_{i+1} (or to the environment in the case of m_k).

Now consider a model $\mathcal{P}$ which is a generator of tuples and the cascading (flow of o's) is from the environment to the innermost membrane. More precisely, let $m_1, ..., m_k$ be the membranes of $\mathcal{P}$, where m_i is enclosed in m_{i+1} for $1 \leq i \leq k-1$ (m_1 is the innermost membrane and m_k is the skin membrane). Initially, there are no o's in the membranes, but there is an infinite supply of o's in the environment. There may also be a finite supply of other symbols in the environment initially. The rules of the forms (u, out) and $(u, out; v, in)$ are restricted in that u cannot contain o's. Thus the o's can only be moved from the environment to membrane m_k and from m_{i+1} to m_i for $1 \leq i \leq k-1$. (Note that once o's reach membrane m_1, they remain there.) The set of numbers generated by $\mathcal{P}$ is $G(\mathcal{P}) = \{\ n \mid \mathcal{P}$ halts with o^n in the skin membrane $m_k\}$. It is important to point out that the skin membrane is the output membrane. We call this new model a *k-membrane reverse-cascade SA*.

Theorem 14 *1. 1-membrane and 2-membrane reverse-cascade SAs are equivalent, and they generate exactly the semilinear sets over N.*

2. 3-membrane reverse-cascade SAs can generate nonsemilinear sets. In fact, for any recursively enumerable (RE) set $R \subseteq N$, the set $\{2^n \mid n \in R\}$ can be generated by a 3-membrane reverse-cascade SA. (Hence, their emptiness problem is undecidable.)

3. Any RE set R can be generated by a 4-membrane reverse-cascade SA.

The proof of Theorem 14 involves the use of a counter machine similar to the k-counter cascade machine in Section 3. Define a *k-counter reverse-cascade machine* $\mathcal{M}$ as a nondeterministic machine with k counters $c_1, ..., c_k$. $\mathcal{M}$ starts in its initial state with all counters zero. As usual, the counters can be incremented/decremented and tested for zero but with the following restrictions:

1. If counter c_{i+1} is decremented it must be followed by an increment of counter c_i for $1 \leq i \leq k-1$, and this is the only way counter c_i can be incremented.

2. Counter c_k can be incremented independently.

3. Counter c_1 cannot be decremented. (Thus, c_1 is nondecreasing, hence, essentially useless. The reason is that once it becomes positive, it will remain positive, and can no longer affect the computation. We include this counter for convenience.)

We say that $\mathcal{M}$ generates a nonnegative integer n if it halts with value n in counter c_k, and the set of all such numbers generated is the set generated by $\mathcal{M}$.

It can be shown that for any $k \geq 1$, a set $R \subseteq N$ is generated by a k-membrane reverse-cascade SA if and only if it can be generated by a k-counter reverse-cascade machine. Then to prove items (1), (2), and (3) of Theorem 14, we need only show that they hold for 1-counter/2-counter, 3-counter, and 4-counter reverse-cascade machines, respectively.

Remark 2. We believe that the 4 membranes in Theorem 14, item (3) is best possible. We think that there are RE sets (even recursive sets) that cannot be generated by 3-counter reverse-cascade machines based on the following discussion.

By definition, in a 3-counter reverse-cascade machine $\mathcal{M}$, with three counters, c_1, c_2, c_3, counter c_1 cannot be decremented. So, in fact, the computation of $\mathcal{M}$ can be simulated by a machine $\mathcal{M}'$ with only two counters: d_1, d_2. Again, the only restriction is that if d_2 is decremented, it must be followed by an increment of d_1, and this is the only way d_1 can be incremented. But now, we allow d_1 to be decremented independently and, as before, d_2 can be incremented independently.

We conjecture that there is an RE set (even a recursive set) that cannot be generated by a 2-counter machine $\mathcal{M}'$ as defined above. (Note that by definition, the generated number is in counter d_2 when the machine halts.)

However, we have no formal proof at this time. □

We can generalize the reverse-cascade SA by using, instead of only one input symbol o, a set of symbols $\Sigma_r = \{a_1, ..., a_r\}$ as input symbols, again with the restriction that these symbols can only be moved from the environment to membrane m_k and from m_{i+1} to m_i for $1 \leq i \leq k-1$. Now the system generates a set of r-tuples of nonnegative integers in the skin membrane when it halts. We can prove:

Theorem 15 *1. 1-membrane reverse-cascade SAs with input alphabet $\Sigma_r = \{a_1, ..., a_r\}$ generate exactly the semilinear sets over N^r.*

2. 2-membrane reverse-cascade SAs with input alphabet $\Sigma_2 = \{a_1, a_2\}$ can generate nonsemilinear sets over N^2. In fact, for any RE set R, the set $\{(2^n, 0) \mid n \in R\}$ can be generated by a 2-membrane reverse-cascade SA with input alphabet Σ_2.

3. For any RE set R, the set $\{(n, 0) \mid n \in R\}$ can be generated by a 3-membrane reverse-cascade SA with input alphabet Σ_2.

4. For any RE set R, the set $\{(n, 0, 0) \mid n \in R\}$ can be generated by a 2-membrane reverse-cascade SA with input alphabet Σ_3.

Remark 3. Again, as in Remark 2, we believe that Theorem 15, item (3), does not hold for 2-membrane reverse-cascade SAs with input alphabet Σ_2. □

In the definition of a reverse-cascade SA, the skin membrane is the output membrane. We now consider the model where the output membrane is the innermost membrane m_1 (and not the skin membrane). Similar to Theorem 14, we can prove the following (but item (3) is weaker):

Theorem 16 *Under the assumption that the output membrane is the innermost membrane m_1 (and not the skin membrane), we have:*

1. 1-membrane and 2-membrane reverse-cascade SAs are equivalent, and they generate exactly the semilinear sets over N.

2. 3-membrane reverse-cascade SAs can generate nonsemilinear sets (e.g., the set $\{n^2 \mid n \geq 1\}$). Moreover, their emptiness problem is undecidable.

3. Any RE set R can be generated by a 5-membrane reverse-cascade SA.

Remark 4. It does not seem that item (3) of the above theorem holds for a 4-membrane reverse-cascade SA, but we have no proof at this rime. □

Finally, consider a 2-level symport/antiport system $\mathcal{P}$ has membranes $m_1, m_1, ..., m_{k+1}$, where $m_1, ..., m_k$ are at the same level, and they are enclosed in the skin membrane m_{k+1}. The environment contains F initially and an infinite supply of o's. We require that for membranes $m_1, ..., m_k$, in the rules of the forms (u, out) and (u, out, v, in), u does not contain o's. Note that there is no restriction on the rules in the skin membrane. For this system, we say that $(n_1, ..., n_k)$ is generated if, when $\mathcal{P}$ is started with no o's in the system and fixed $w_i \in F^*$ in m_i $(1 \leq i \leq k+1)$, $\mathcal{P}$ halts with $o^{n_1}, ..., o^{n_k}$ in membranes $m_1, ..., m_k$. Call the system just described a *simple SA generator*. We can show the following:

Theorem 17 *A set $R \subseteq N^k$ is generated by a simple SA generator if and only if R is a semilinear set.*

The theorem above no longer holds when the simple SA generator is extended to a 3-level structure, as Theorem 16, item (2) shows.

6 Conclusion

In this paper, we introduced restricted models of symport/antiport P systems and proved that they characterize precisely the semilinear sets. We also showed that "slight" generalizations of the models allowed them to accept nonsemilinear sets, and made their emptiness problem undecidable. We also looked at related models that are used as generators of sets of tuples. Some models generate exactly the semilinear sets; others generate the recursively enumerable sets. We mentioned some interesting open questions in Remarks 1-4.

Acknowledgement

The work of Oscar H. Ibarra and Sara Woodworth was supported in part by NSF Grants CCR-0208595 and CCF-0430945. The work of Hsu-Chun Yen was supported in part by NSC Grant 93-2213-E-002-003, Taiwan. The work of Zhe Dang was supported in part by NSF Grant CCF-0430531.

References

[1] E. Csuhaj-Varjú, O. H. Ibarra, G. Vaszil: On the computational complexity of P automata. In *Proc. DNA 10* (2004).

[2] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana: Multiset automata. *Proc. Workshop on Multiset Processing*, *Lectures Notes in Computer Science* **2235** (2000), 69–84.

[3] S. Ginsburg: The Mathematical Theory of Context-Free Languages. McGraw-Hill, New York (1966).

[4] O. H. Ibarra: Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM* **25** (1978), 116–133.

[5] O. H. Ibarra: On membrane hierarchy in P systems. *Theoretical Computer Science* **334** (2005), 115–129.

[6] M. Ito, C. Martín-Vide, Gh. Păun: A characterization of Parikh sets of ETOL languages in terms of P systems. In: M. Ito, Gh. Păun, S. Yu (Eds.): *Words, Semigroups, and Transductions.* Festscrift in Honor of Gabriel Thierrin. World Scientific, Singapore (2001), 239–253.

[7] M. Minsky: Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines. *Ann. of Math* 74 (1961), 437–455.

[8] C. Martín-Vide, Gh. Păun, J. Pazos, A. Rodríguez-Patón: Tissue P systems. *Theoretical Computer Science* **296** (2003), 295–326.

[9] A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing* **20** (3) (2002), 295–306.

[10] Gh. Păun: *Membrane Computing: An Introduction.* Springer-Verlag, Berlin (2002).

[11] Gh. Păun, J. Pazos, M. J. Pérez-Jiménez, A. Rodríguez-Patón: Symport/antiport P systems with three objects are universal. Fundamenta Informaticae **64** (1-4) (2005), 353–367.

P Systems, Petri Nets and Program Machine

Pierluigi FRISCO

Deptartment of Computer Science
School of Eng., C. S., and Math.
University of Exeter, Harrison Building
North Park Road
Exeter, EX4 4QF, U.K.
E-Mail: P.Frisco@exeter.ac.uk

Abstract

Some features capturing the computational completeness of P systems with maximal parallelism, priorities or zero-test using symbol objects are studied through Petri nets. The obtained results are not limited to P systems.

1 Introduction

Membrane systems (also called *P systems*) are a new class of distributed and parallel theoretical computing devices introduced in [17]. In the seminal paper the author considers systems based on a hierarchically arranged, finite *cell-structure* consisting of several cell-membranes embedded in a main membrane called the *skin*. The membranes delimit *regions* where *objects*, elements of a finite set, and evolution rules can be placed.

In [17] the author examines three ways to view P systems: transition, rewriting and splicing P systems. Starting from these, several variants were considered [24, 18]. These variant can be divided in two main categories: P systems using symbol objects and P systems using string objects. Several proofs of computationally complete P systems using symbol objects are a simulation of Program machines.

In this research we tried to discover and study the principles underlying the P systems using symbol objects that happen to be computationally

complete. We focused our attention on the processes of these systems and we used Petri nets as tool for our investigations. Links between P systems and Petri nets have been already investigated [25, 20, 11].

The results obtained by us are not limited to P systems.

2 Basic Definitions

We assume the reader to have familiarity with basic concepts of formal language theory [10], in particular with the topic of P systems [18], Petri nets [21] and program machines [14]. In the following subsections we recall particular aspects relevant to our presentation.

2.1 General

We denote with $\mathbb{N}$ the set of natural numbers while $\mathbb{N}_0 = \{0\} \cup \mathbb{N}$. We use $\mathbb{N}_0\mathsf{RE}$ to denote the family of recursively enumerable sets of natural numbers. For $k \in \mathbb{N}_0$, $\mathbb{N}_k \cdot \mathsf{RE}$ equals the family of recursively enumerable sets with elements greater or equal than k.

Let V be a finite set of objects. With V^* we indicate the free monoid generated by V with the operation of concatenation, λ indicates the empty word. A *multiset* (over V) is a function $M : V \to \mathbb{N}_0 \cup \{+\infty\}$; for $a \in V$, $M(a)$ defines the *multiplicity* of a in the multiset M. We will say that an element a of a multiset M has *infinite multiplicity* if $M(a) = +\infty$. In case the multiplicity of an element of a multiset is 1 we will indicate just the element, otherwise $(a, M(a))$ is indicated. The *support* of a multiset M is the set $supp(M) = \{a \in V \mid M(a) > 0\}$. The *size* of a multiset is defined by the function $|\cdot| : (V \to \mathbb{N}_0 \cup +\{\infty\}) \to \mathbb{N}_0 \cup \{+\infty\}$, where for M multiset over V, $|M| = \sum_{a \in supp(M)} M(a)$. The symbol ϕ indicates the *empty multiset*, that is the multiset whose support is the empty set.

Let $M_1, M_2 : V \to \mathbb{N}_0 \cup \{+\infty\}$ be two multisets. The *union* of M_1 and M_2 is the multiset $M_1 \cup M_2 : V \to \mathbb{N}_0 \cup +\{\infty\}$ defined by $(M_1 \cup M_2)(a) = M_1(a) + M_2(a)$, for all $a \in V$. The *difference* $M_1 \backslash M_2$ is here defined only when M_2 is *included* in M_1 (which means that $M_1(a) \geq M_2(a)$ for all $a \in V$) and it is the multiset $M_1 \backslash M_2 : V \to \mathbb{N}_0 \cup \{+\infty\}$ given by $(M_1 \backslash M_2)(a) = M_1(a) - M_2(a)$ for all $a \in V$. Of course, if $M_1(a) = +\infty$ and $M_2(a)$ is finite, then $M_1(a) \backslash M_2(a) = +\infty$. If $M_2(a) = +\infty$, then $M_1(a) \backslash M_2(a) = 0$.

2.2 P Systems

In this subsection we will not give the definition of a specific P system, we will give a general definition for P systems whose content of the membrane compartments are multisets of objects.

Let us consider the construct $\Pi = (V, \mu, L_1, \ldots, L_m, R, fin)$, where:

V is a set of objects;

$\mu = (N, E)$ is a *directed graph* underlying Π. The set $N \subset \mathbb{N}$ contains *vertices*, for simplicity we define $N = \{1, \ldots, m\}$. Each vertex in N defines a *membrane* of Π. The set $E \subseteq N \times N$ defines directed *edges* between vertices indicated by (i, j).

L_i over $V \to \mathbb{N}_0 \cup \{+\infty\}$ are the multisets associated to membranes $i \in N$, they define the input of the system. If an object belongs to the support of a multiset associated to a membrane, then we will say that the object is present into the membrane.

R is a finite sets of *rules*, that is quadruples of the form $(\alpha_i, \beta_i; \gamma_j, \delta_j)$ where each α, β, γ and δ are (possibly empty) multisets over V and $i, j \in N$. If M_i and M_j are the multisets associated to membranes i and j respectively, then the application of $(\alpha_i, \beta_i; \gamma_j, \delta_j) \in R$ changes M_i into M_i' and M_j into M_j' such that: $M_i' = M_i \backslash \alpha_i \cup \beta_i$ and $M_j' = M_j \backslash \delta_j \cup \gamma_j$.

$fin \in N$ defines the *final membrane* indicating the multiset output of the system when a certain condition is met. This condition can be, for instance, the impossibility to apply any rule (as in P systems with symport/antiport [16, 12, 8, 2, 23]) or the presence of an object in a specific membrane (as in conformon-P systems [6]).

Let us clarify how the rules in R generalize other rules present in variants of P systems. We consider a P system $\mathbf{P}$ having $[_0[_1[_2]_2]_1]_0$ as membrane structure. If $\mathbf{P}$ is with symport/antiport and (aa, out) and $(b, \text{in}; cc, \text{out})$ are associated to membrane 2, then this is equivalent to a system Π having rules $(aa_2, \emptyset; \emptyset, aa_1)$ and $(cc_2, b_2; b_1, cc_1)$. If $\mathbf{P}$ is a system with catalyst [17, 4] and $Cab \to Cd_{here}$ and $Cab \to Cd_{out}$ are associated to membrane 2 (C is a catalyst while a, b and d are not), then this is equivalent to a system Π having rules $(ab_2, d_2; \emptyset_2, \emptyset_2)$ and $(ab_2, \emptyset_2; \emptyset_1, d_1)$. If $\mathbf{P}$ is a conformon-P system having the rule $r : A \xrightarrow{3} B$ associated to membrane 2, then this is equivalent

to a system Π having rules $([A,x][B,y]_2, [A,x-3][B,y+3]_1; \emptyset_1, \emptyset_1), x \in \{3,4,5,\ldots\}, y \in \mathbb{N}_0$ associated to membrane 2.

A *configuration* of Π is an m-tuple $(M_1,\ldots,M_m)$ of multisets over $V \times \mathbb{N}_0 \cup \{+\infty\}$. From two configurations $(M_1,\ldots,M_m), (M'_1,\ldots,M'_m)$ of Π we write $(M_1,\ldots,M_m) \Rightarrow (M'_1,\ldots,M'_m)$ indicating a *transition* from $(M_1,\ldots,M_m)$ to $(M'_1,\ldots,M'_m)$ that is the parallel application of a multiset of rules. If no rule is applied to a multiset M_i, then $M_i = M'_i$. Notice that transitions could be applied under the requirement of *maximal parallelism* that is, a multiset of rules cannot be applied if there is a strictly larger multiset of rules that could be applied.

A *computation* is a finite sequence of transitions between configurations of a system Π starting from $(L_1,\ldots,L_m)$. The result of a computation is given by the multiset of objects present in membrane fin when a condition is met, and such a multiset is indicated by $L(\Pi)$.

2.3 Petri Nets

Definition 1 *A Petri net is a quadruple $M = (P,T,F,C_{in})$, where:*

i) (P,T,F) is a net, that is:

1. *P and T are sets with $P \cap T = \emptyset$;*
2. *$F \subseteq (P \times T) \cup (T \times P)$;*
3. *for every $t \in T$ there exist $p,q \in P$ such that $(p,t),(t,q) \in F$;*
4. *for every $t \in T$ and $p,q, \in P$, if $(p,t),(t,q) \in F$, then $p \neq q$.*

ii) $C_{in} \subseteq P$ is the initial configuration.

Given a Petri net $M = (P,T,F,C_{in})$, (P,T,F) is the *underlying net* of M.

A *directed edge-labelled tree* is a tree provided with a labelling function for its edges. Given a Petri net $M = (P,T,F,C_{in})$ we define the *sequential configuration graph of M*, denoted by $SCG(M)$, as a directed edge-labelled tree having elements in $\mathbb{C}_M$ (the set of all *reachable configurations*) as vertices, C_{in} as root, $E = \{(C,D) \mid C,D \in \mathbb{C}_M, t \in T, C[t\rangle D\}$ as set of directed edges and $label : E \to T$ as labelling function. If $e = (C,D) \in E$, then $label(e) = t$ if $C[t\rangle D$ (that is, if $t \in T$ *fires* from C to D).

Moreover, $U_{C,D} = \{U_i \subseteq T \mid U_i$ is a concurrent step from C to $D\}$ is the *set of concurrent steps from C to D* and $U^{max}_{C,D} = \{U \mid |U| \geq |V| \ \ \forall U,V \in U_{C,D}\}$ is the *maximal set of concurrent steps from C to D.*

The *configuration graph of M*, denoted by $CG(M)$, is similar to the $SCG(M)$ but it has $E = \{(C, D) \mid C, D \in \mathbb{C}_M, U \subseteq T, C[U\rangle D\}$ and $label : E \to \mathcal{P}(T)$ (where, given a set A, $\mathcal{P}(A)$ indicates its *power set*, the set of all subsets of A). If $e = (C, D) \in E$, then $label(e) = U$ if $C[U\rangle D$ (that is, $U \subseteq T$ is a *concurrent step* from C to D).

The *configuration graph with maximal concurrency of M*, denoted by $CGMC(M)$, is similar to the $CG(M)$ but is has $E = \{(C, D) \mid C, D \in \mathbb{C}_M, U \in U^{max}_{C,D}\}$ and $label : E \to \mathcal{P}(T)$ is such that if $e = (C, D) \in E$, then $label(e) = U \in U^{max}_{C,D}$.

The definition of Petri nets can be extended to the one of *place/transition systems*, *P/T systems* for short, allowing a place to contain more than one token and more than one token to be removed/added from/to places as a consequence of a firing.

Definition 2 *A P/T system is a tuple $M = (P, T, F, W, K, C_{in})$, where:*

i) (P, T, F) *is a net (see Definition 1);*

ii) $W : F \to \mathbb{N}$ *is a weight function;*

iii) $K : P \to \mathbb{N} \cup \{+\infty\}$ *is a capacity function;*

iv) $C_{in} : P \to \mathbb{N}_0$ *is the initial configuration.*

A *configuration* of a P/T system is a multiset over P; if we consider a linear order on the elements of P, then a configuration can be regarded as a vector (this fact will be used in Definition 3). The dynamic behaviour of a P/T system is analogous to the one of a Petri net, but considering the weight and capacity functions.

A P/T system with *maximal concurrency* M is such that for each $C, D \in \mathbb{C}_M$ if there is a $U \subseteq T$ such that $C[U\rangle D$, then $U \in U^{max}_{C,D}$.

Petri nets can be regarded as a specific kind of a P/T systems having $W : F \to \{1\}$ and $K : P \to \{1\}$ as weight and capacity functions, respectively. Moreover, a P/T system can be regarded as a 'compressed' Petri net so, given a P/T system it is always possible to create a Petri net modelling the same process of the P/T system.

Definition 3 *Given a P/T system $M = (P, T, F, W, K, C_{in})$ a vector $i : P \to \mathbb{Z}$ is a p-invariant of M if for all configurations C, D of M and all $t \in T$: if $C[t\rangle D$, then $C \cdot i = D \cdot i$ (in this case a configuration is regarded as a vector).*

2.4 Program Machines

Non-rewriting Turing machines were introduced by M. L. Minsky in [13] and then reconsidered in [14] under the name of *program machines*.

Formally a *program machine* with n counters ($n \in \mathbb{N}$) is defined as $M = (S, R, s_0, f)$, where S is a finite set of *states*, $s_0, f \in S$ are respectively called the *initial* and *final* states, R is the finite set of *instructions* of the form $(s, op(l), v, w)$, with $s, v, w \in S,\ s \neq f,\ op(l) \in \{l_+, l_-\},\ 1 \leq l \leq n$

A *configuration* of a program machine M with n counters is given by an element in the $n+1$-tuples $S \times \mathbb{N}_0^n$. Given two configurations $(s, l_1, \ldots, l_n)$, $(r'$,
$l'_1, \ldots, l'_n)$ we define a *computational step* as $(s, l_1, \ldots, l_n) \vdash (s', l'_1, \ldots, l'_n)$ if $(s, op(l), v, w) \in R$ and:

- if $op(l) = l_-,\ l = l_i$ and $l_i \neq 0$, then $s' = v,\ l'_i = l_i - 1,\ l'_j = l_j,\ j \neq i,\ 1 \leq j \leq n$;
  if $op(l) = l_-,\ l = l_i$ and $l_i = 0$, then $s' = w,\ l'_j = l_j,\ 1 \leq j \leq n$;
  (informally: in state s if the content of counter l is greater than 0, then subtract 1 from that counter and change state into v, otherwise change state into w)

- if $op(l) = l_+,\ l = l_i$, then $s' = v, l'_i = l_i + 1,\ l'_j = l_j,\ j \neq i,\ 1 \leq j \leq n$;
  (informally: in state s add 1 to counter l and change state into v).

The reflexive and transitive closure of $\vdash$ is indicated by $\vdash^*$.

A *computation* is a finite sequence of transitions between configurations of a program machine M starting from the initial configuration $(s_0, l_1, \ldots, l_n)$ with $l_1 \neq 0,\ l_j = 0,\ 2 \leq j \leq n$. If the last of such configurations has f as state, then we say that M *accepted* the number l_1. The set of numbers accepted by M is defined as $L(M) = \{l_1 \mid (s_0, l_1, \cdots, l_n) \vdash^* (f, l''_1, \cdots, l''_n)\}$.

2.5 From P to P

Throughout this paper we only consider systems whose set of configurations is finite and such that some elements of this set are *final*, i.e. there is no transition from a final state to any other state.

The results present in Section 5 are related to the simulation of one system performed by another one. A simulation is a relation from the set of subsets of the configurations of a system, the *simulating* one, to the set of configurations of another (different) system, the *simulated* one.

This relation (mapping) induces a partition in the set of configurations of the simulating system and associates the final configurations of this system to the final configurations of the simulated one.

Now we formally define a *simulation* between two such systems.

Given a set A we define with $Sub(A) = \{A_1, \ldots, A_t\}, A_i \subseteq A, 1 \leq i \leq t$ a *subdivision* of A into subsets. If $Sub(A)$ is such that $\bigcup_{i=1}^{t} A_i = A$ and $A_p \cap A_q = \emptyset, p \neq q, 1 \leq p, q \leq t$, then $Sub(A)$ is a *partition* of A indicated with $Part(A)$.

Let S, S' two different systems with $\mathbb{C} = \{c_1, c_2, \ldots, c_v\}$ and $\mathbb{C}' = \{c'_1, c'_2, \ldots, c'_{v'}\}$ their respective (finite) sets of configurations.

A computation of a system S is a finite sequence of configurations (in which it can be that not all configurations of the system are present).

Given the set of configurations $\mathbb{C}$ of a system S it is possible to distinguish one subset $F_{\mathbb{C}}$ in it: the set of *final* configurations (all sharing a certain termination criterion). The last configuration in all computations of S is an element of $F_{\mathbb{C}}$.

Let us denote with $\Rightarrow$ ($\Rightarrow'$) the transition from one configuration to another in a computation of S (S'). Moreover, let $\Rightarrow^*$ ($\Rightarrow'$) the reflexive and transitive closure of $\Rightarrow$ ($\Rightarrow'$).

We will say the S can simulate S' if there is a relation $Sim \subseteq Sub(\mathbb{C}) \times \mathbb{C}'$ such that:

1. for each $c'_{g'}, c'_{h'} \in \mathbb{C}', g' \neq h'$ such that $c'_{g'} \Rightarrow' c'_{h'}$ there are $G, H \in Sub(\mathbb{C})$ such that $Sim(G) = c'_{g'}$ and $Sim(H) = c'_{h'}$ and $c_g \Rightarrow c_h$ with $c_g \in G$ and $c_h \in H$;

2. for each $c'_{f'} \in F_{\mathbb{C}'}$ there is $F \in Sub(\mathbb{C}), F = F_{\mathbb{C}}$ such that $Sim(F) = c'_{f'}$.

So, if $b' \Rightarrow' e', e' \in F_{\mathbb{C}'}$, then there are $B, E \in Sub(\mathbb{C}), E = F_{\mathbb{C}}$ such that $Sim(B) = b', Sim(E) = e'$ and $b \Rightarrow^* e$ with $b \in B$ and $e \in F_{\mathbb{C}}$. In this case we can also write $B \Rightarrow^* E$.

S is called the *simulating* system and the S' the *simulated* system.

Notice that neither all elements in $Sub(\mathbb{C})$ are in relation with an element $e' \in \mathbb{C}'$, nor for each element in $\mathbb{C}'$ there is $E \in Sub(\mathbb{C})$ such that $Sim(E) = e'$.

Lemma 1 *Give two systems S and S' the simulation relation $Sim \subseteq Sub(\mathbb{C}) \times \mathbb{C}'$ induces a partition in $\mathbb{C}$.*

So we can say that if S can simulate S', then there is a relation $Sim \subseteq Part(\mathbb{C}) \times \mathbb{C}'$.

Given two systems it is possible to have several relations defining different simulations.

3 Maximal Parallelism, Priorities and Indication of Emptiness

Several proofs of the computational completeness of P system using symbol objects are based on the simulation of program machines and use either maximal parallelism [16], or priorities [22] or indication of emptiness of the counters (of the simulated program machine) [6]. With *indication of emptyness* we mean that in the P system there is a configuration associated to the one(s) of the program machine in which one of the counters is zero (i.e. is empty). For some of these variants (for instance, [7, 6]) it is proved that when these elements are missing, then the computational power is reduced to the one of partially blind program machines.

These facts suggests that maximal parallelism, priorities and the indication of emptiness are different aspects of the same feature, that is that they are equivalent. To our knowledge there is not direct proof of this (an indirect proof would be to consider one computational device and show that with either maximal parallelism, priorities or indication of emptiness it is computationally complete).

In this section we prove that, for P/T systems maximal concurrency, priorities and indication of emptiness are equivalent.

In [3] it is indicated that P/T systems with maximal concurrency (there called *maximum strategy*) can perform the test on zero (addressed by us as the *0-test*) simulating the rule (s, l_-, v, w) (addressed by us as the *0-rule*) of a program machine. This simulation is performed by the net underlying the P/T system with maximal concurrency depicted in Figure 1. The same P/T system simulates the 0-rule if the firing of transition t_2 has priority on the firing of transition t_5 (and maximal concurrency is not present). In this P/T system the number of tokens that can be present in p_c is unbounded. It is important for us to notice that in the P/T system simulating the program machine there are as many such P/T systems performing the 0-test as many 0-rules present in the simulated program machine. Moreover, as the program machine can be in only one state per time, then at most one place p_s in a P/T system performing a 0-test can have a token.

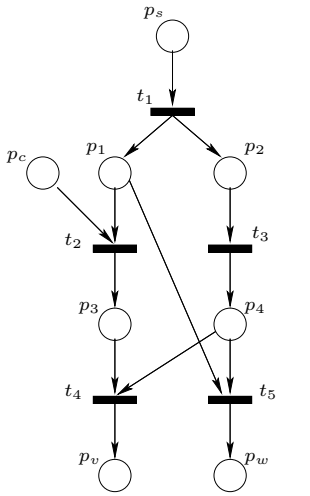


Figure 1: Net underlying the P/T system for the 0-test

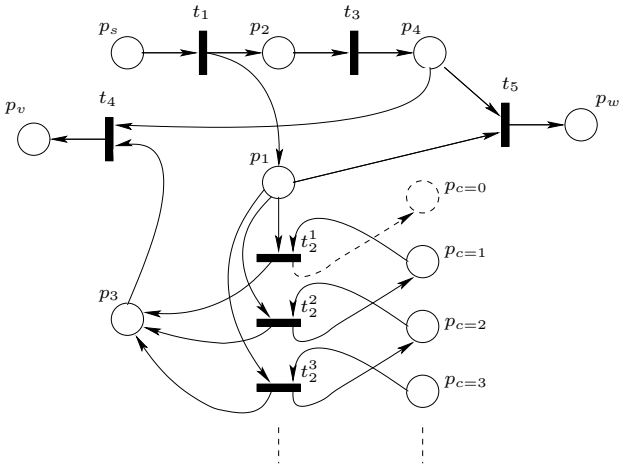


Figure 2: Net underlying the Petri net for the 0-test

If we transform the net underlying the P/T system present in Figure 1 into a Petri net, then we obtain what depicted in Figure 2 (without considering the dashed place and edge) where, if we consider the presence of priorities (and the absence of maximal parallelism), all the $t_2^i, i \in \mathbb{N}$ have priority on t_5. In Figure 2 if the counter c of the simulated program machine has value $i, i \in \mathbb{N}$, then a place $p_{c=i}$ has a token (we will discuss the case $i = 0$ in a while).

It should be clear that neither the just mentioned P/T system, nor the Petri net perform the 0-test if maximal parallelism and priorities are not considered. In the net underlying the P/T system in Figure 1 it can be, for instance, that if at least one token is present in p_c and one token is present in p_s, the system can fire t_1, t_3 and t_5 in sequence reaching a configuration having one token in p_w.

As in the net underlying the P/T system in Figure 1 the place p_c can contain arbitrary many tokens, then the net underlying the Petri net in Figure 2 the number of transitions t_2^i and of places $p_{c=i}, i \in \mathbb{N}$ is infinite.

The CGMC and the SCG (in case of priorities) for all the permitted initial configurations of the net underlying the Petri net depicted in Figure 2 are represented in Figure 3. The graphs depicted in Figure 3 are such that, once in the root, the system can only evolve toward the configuration at the bottom. So we can write a 'reduced' graph. This is done in Figure

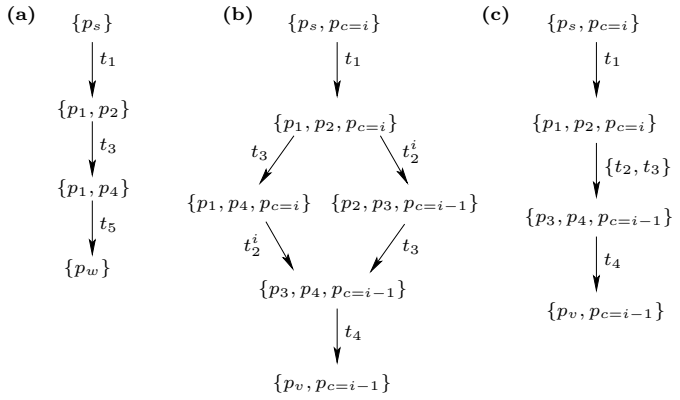


Figure 3: **(a)**, **(b)** SCG for the 0-test with priorities, **(a)**, **(b)** CG for the 0-test with maximal parallelism, $i \in \mathbb{N}$

4.a and Figure 4.c.

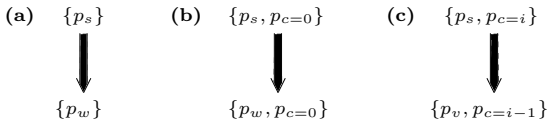


Figure 4: 'Reduced' (S)CGs, $i \in \mathbb{N}$

As in each configuration of a program machine a counter can have only one value, a Petri net simulating it would have $\sum_{i=1}^{\infty} p_{c=i} = 1$ as p-invariant (where $p_{c=i}$ are the places in the net depicted in Figure 2). If in the same net we consider the dashed place $p_{c=0}$ and the dashed edge incoming this place, then its 'behaviour' is not changed, but it is actually enlarged to model a program machine also when a counter c is empty. The configuration graphs are changed adding the place $p_{c=0}$ to every configuration in Figure 3.a. This means that its 'reduced' configuration graph changes into the one depicted in Figure 4.b.

If we now consider the 'reduced' configuration graphs depicted in Figure 4.b and Figure 4.c, then we can create another net underlying a Petri net implementing the 0-test. This is depicted in Figure 5. Because of the edges from $p'_{c=0}$ to $t_2^{'0}$ and vice versa, what depicted in Figure 5 it is not a net underlying a Petri net as defined in Definition 1 (as point 4 is not satisfied). We can overcome this simply removing the edge from $t_2^{'0}$ to $p'_{c=0}$ and adding

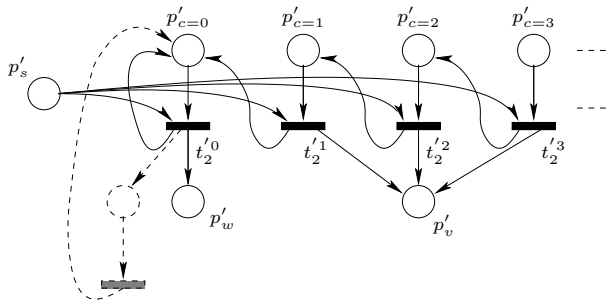


Figure 5: Net underlying the 'reduced' Petri net for the 0-test

the dashed place, transition and edges.

The net underlying the Petri net depicted in Figure 5 can be regarded as a rewriting of the one depicted in Figure 2. Such rewriting implements the 0-test without maximal concurrency and priorities but with an infinity of places and transitions.

Recalling (from Section 2.3) that a Petri net can be regarded as a specific kind of a P/T system we can say that:

Theorem 1 *For a P/T system*

1. *finite number of places + unbounded number of tokens in a place + maximal concurrency;*

2. *finite number of places + unbounded number of tokens in a place + priorities;*

3. *infinite number of places (that is, indication of emptiness)*

are similar ways to perform the 0-test.

It is important to notice that the net underlying the Petri net depicted in Figure 5 performs more than just the 0-test, it performs the i-test for $i \in \mathbb{N}_0$.

4 0-Test and 0-Gamble

In this section we will show that two ways to simulate the 0-rule are equivalent. The proof uses nets and building blocks.

The 0-test can be performed by nets different from the one depicted in Figure 1. Several papers on variant of P systems using symbol objects use another mapping from P systems to P/T systems) to simulate the 0-rule (s, l_-, v, w).

This procedure, addressed by us as the *0-gambling*, can be represented by the net underlying the P/T system depicted in Figure 6 where maximal concurrency or priorities (transition t_4 has priority on transition t_6) are present.

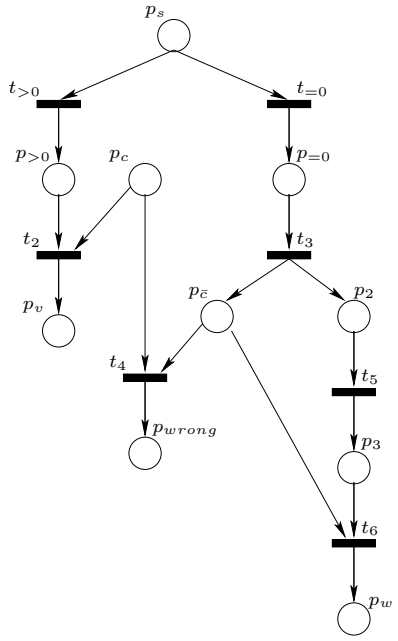


Figure 6: Net underlying the P/T system for the 0-gamble

Let us say that once in state p_s the system 'gambles' one of two cases: either the counter is empty or it is not. If the gamble is correct, then the system will evolve in state p_w or p_v respectively, in case of wrong gamble it will go into the state p_{wrong} or it will block, respectively.

It is interesting to notice what happens when the system gambles that the counter in not empty. In this case the places $p_{\bar{c}}$ and p_2 get a token and then either p_{wrong} or into p_{s_0} will get a token.

The presence of the place $p_{\bar{c}}$ is of interest for our discussion. Following what indicated in [8] we can name this place 'conflicting counter'. Similar concept has been used also in [5]. Even if explicitly defined in the just

0-test	**c**-**b**, **c**-**d**, **b**-**b**, **d**-**b**
0-gamble	**a**-**b**, **a**-**c**, **b**-**b**, **c**-**b**, **c**-**d**, **d**-**b**

Table 1: Pairs of building blocks for the 0-test and the 0-gamble (with **d**)

mentioned papers, the concept of 'conflicting counter' was already implicitly present in all the proofs of P systems using symbol objects generating $\mathbb{N}\cdot\mathsf{RE}$ and using the 0-gamble ([16], for instance).

Some of these proofs were so made that once in p_w the computation could still enter the p_{wrong} place (for instance Theorem 1 in [7]).

At this point we can wonder on the relation between the 0-test (Figure 1) and the 0-gamble (Figure 6).

Before studying this further, let us introduce the nets depicted in Figure 7. We call these nets *building blocks*.

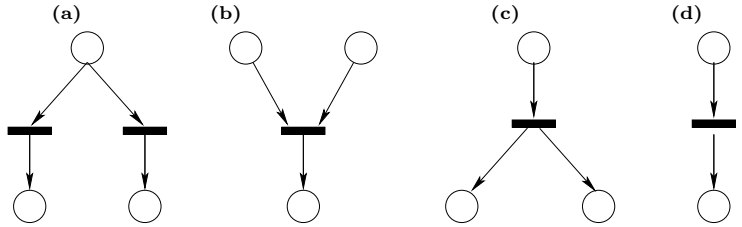


Figure 7: Building blocks: **(a)** nondeterminism, **(b)** join, **(c)** fork, **(d)** determinism

Both the 0-test and the 0-gamble are implemented by a combination of the building blocks. In the net underlying the P/T system for the 0-test (Figure 1) $\{p_c, p_1, p_3\}, \{p_3, p_4, p_v\}$ and $\{p_1, p_4, p_v\}$ are building blocks of type **b**; $\{p_s, p_1, p_2\}$ is a building block of type **c** and $\{p_2, p_4\}$ is a building block of type **d**. In the net underlying the P/T system for the 0-gamble (Figure 6) $\{p_s, p_{>0}, p_{=0}\}$ is a building block of type **a**; $\{p_{>0}, p_c, p_v\}, \{p_c, p_{\bar{c}}, p_{wrong}\}$ and $\{p_{\bar{c}}, p_3, p_w\}$ are building blocks of type **b**; $\{p_{=0}, p_{\bar{c}}, p_2\}$ is a building block of type **c** and $\{p_2, p_3\}$ is a building block of type **d**. In this figures we also see that the relative arrangement of pairs of building blocks is the one indicated in Table 1.

At this point we can say that:

0-test	**c-b**, **b-b**
0-gamble	**a-b**, **a-c**, **b-b**, **c-b**

Table 2: Pairs of building blocks for the 0-test and the 0-gamble (without **d**)

Theorem 2 *A necessary condition such that a system S with maximal parallelism or priorities using symbol objects can implement the 0-test (or the 0-gamble) is that there is a mapping from S to a net underlying a P/T system such that it is possible to have a sequence of sets of configurations in S associated to the subsequent firing of transitions present in the pairs of building blocks listed above.*

The previous theorem does not give a sufficient condition as the system could have features (limits in the number of tokens present in a place, limits on the firing of transitions, etc.) not allowing it to perform the 0-test even if its process can be described by the building blocks and that pairs of building blocks can be combined in the ways listed in Table 1.

As the relative arrangements of the pairs of building blocks present in the 0-test are part of the relative arrangements of building blocks present in the 0-gamble, then we can say that:

Corollary 1 *If a system with maximal parallelism or priorities using symbol objects can implement the 0-gamble, then it can implement the 0-test.*

Both in the 0-test (Figure 1) and the 0-gamble (Figure 6) the building block **d** can be substituted by the building block **c** followed by the building block **b** without affecting the behaviour of the P/T system. If we consider this the relative arrangements of pairs of building blocks indicated in Table 1 changes into the one indicated in Table 2.

5 Unifying Results

In this section we generalise the result obtained in Section 3 showing that maximal parallelism, priorities and indication of emptiness are equivalent concepts for systems using symbol objects. To do so we use the same technique used in Section 4.

0-test, non det.	**b-a**	0-gamble, non det.	**b-a**
non det., 0-test	**a-c**	non det., 0-gamble	**a-c**
fork, 0-test	**c-b**, **c-c**	fork, 0-gamble	**c-b**, **c-a**
0-test, fork	**b-c**	0-gamble, fork	**b-c**
fork, non det.	**c-a**	fork, non det.	**c-a**
non det., fork	**a-c**	non det., fork	**a-c**

Table 3: Pairs of building blocks for 0-test (0-gamble), fork and nondeterminism

The capability to simulate the 0-rule is only one of the operations needed for a system using symbol objects to simulate a program machine. The system has also to simulate the rule (s, l_+, v, w) (adding 1 to a counter) and to simulate nondeterminism (see Section 2.4). These two operations can be simulated by the building blocks **c** and **a** respectively. These operations: 0-rule, addition of 1 and nondeterminism, can be performed by the system in any order. If we consider that the simulation of the 0-test can be followed by the simulation of nondeterminism, then it has to be possible that the building block **b** (as in Figure 1 both p_v and p_w are in such building block) is followed by the building block **a**. Reasoning in a similar way we obtain the pairs of building blocks listed in Table 3.

This implies that:

Corollary 2 *A necessary condition such that a system S with maximal parallelism or priorities using symbol objects can simulate a program machine is that there is a mapping from S to a net underlying a P/T system such that it is possible to have a sequence of sets of configurations in S associated to any subsequent firing of transitions present in the pairs of building blocks listed in Table 1 and Table 3.*

Reasoning in a similar way we can say that

Corollary 3 *A necessary condition such that a system S using symbol objects can simulate a P/T system with indication of emptiness is that there is a mapping from S to a net underlying a P/T system such that it is possible to have a sequence of sets of configurations in S associated to any subsequent firing of transitions present in the pairs of building blocks listed in Table 4.*

Now we are ready to state the main result of this paper.

inside ind. empty	**b-c**
ind. empty., non det.	**c-a**
non det., ind. empty.	**a-b**
fork, ind. empty.	**c-b**
ind. empty, fork	**c-c**
fork, non det.	**c-a**
non det., fork	**a-c**

Table 4: Pairs of building blocks for indication of emptiness, fork and non-determinism

Theorem 3 *Let $X = \{$maximal parallelism, priorities, indication of emptiness$\}$ and let D with $x \in X$ be a computationally complete device using symbol objects. Let D' be similar to D but with $x' \in X, x' \neq x$.*

1. *If $x =$ 'maximal parallelism' and $x' =$ 'priorities', then D' with x' is also computationally complete (similar if $x =$ 'priorities' and $x' =$ 'maximal parallelism');*

2. *If $x =$ 'maximal parallelism' and $x' =$ 'indication of emptiness', then necessary condition for D' with x' to be computationally complete is the presence of a mapping from D' to the net underlying a P/T system such that it is possible to have a sequence of sets of configurations in D' associated to the subsequent firing of the transitions present in the pair of building blocks **c-c** (similar if $x =$ 'priorities');*

3. *If $x =$ 'indication of emptiness' and $x' =$ 'maximal parallelism', then necessary condition for D' with x' to be computationally complete is the presence of a mapping from D' to the net underlying a P/T system such that it is possible to have a sequence of sets of configurations in D' associated to the subsequent firing of the transitions present in the pairs of building blocks **b-b** and **b-a** (similar if $x' =$ 'priorities');*

In the previous theorem if $x' =$ 'indication of emptiness' more than being able to arrange the building blocks as indicated, D' needs a finite way to address the infinite number of places that are created.

6 Final Remarks

How can the results obtained in the previous sections be of any use in the study of the computational power of a system using symbol objects?

Given a computability system S using symbol objects one can prove that it is computationally complete using Theorem 3, and not trying the simulation of a computationally complete device. As Theorem 3 states only a necessary condition it has also to be proved that there are no limitations affecting the work of S.

More relevant consequences are present in case a system S using symbol objects cannot fulfil what stated in Theorem 3. Not verifying the necessary condition present in that theorem S cannot be computationally complete.

At the present time we do not know if the building blocks **a**, **b** and **c** represent a base for nets underlying a P/T systems generating $\mathbb{N}\cdot\mathsf{RE}$. For sure they do not represent a base for a general net.

Our intention is to go further on the line of this research trying to give an answer to the following questions:

How computationally powerful is a P/T system having an underlying net composed by proper subsets of the building blocks depicted in Figure 7? (in this respect the computational differences between program machine and partially blind program machine [9] and the infinite hierarchy described in [15] are going to be of help)

How the computational power is affected if we limit the kind of relative arrangements of pairs of building blocks?

What happens if we allow the P/T system having as underlying net the one depicted in Figure 2 to have as p-invariant $\sum_{i=1}^{\infty} p_{c=i} = n, n \in \mathbb{N}_0$? (which means that more than one place of the kind $p_{c=i}$ can have a token or that some of them have more than one token)

Is it possible to extend these results to systems not using symbol objects?

Our ultimate question on this subject is: is it possible to create an hierarchy of computational power based on building blocks, their combinations and the functions W and K (see Definition 2)?

7 Acknowledgements

This work has been supported by the research grant NAL/01143/G of The Nuffield Foundation.

References

[1] A. Alhazov, C. Martín-Vide, Gh. Păun (Eds.): *Preproceedings of the Workshop on Membrane Computing, WMC-2003*. Rovira i Virgili University (2003).

[2] F. Bernardini, M. Gheorghe: On the power of minimal symport/antiport. *PreProcedings Workshop on Membrane Computing, WMC-2003*. Tarragona (2003) and *Technical Report* **28/03**, Research Group on Mathematical Linguistics, Universitat Rovira i Virgili, Tarragona, 72–83.

[3] H.-D. Burkhard: Ordered firing in Petri nets. *Journal of Information Processing and Cybernetics* **17** (2–3) (1981), 71–86.

[4] R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science* **230** (2) (2005), 251–266.

[5] R. Freund, M. Oswald: P systems with activated/prohibited membrane channels. In [19], 261–269.

[6] P. Frisco: The conformon-P system: A molecular and cell biology-inspired comutability model. *Theoretical Computer Science* **312** (2-3) (2004), 295–319.

[7] P. Frisco: About P systems with symport/antiport. To appear in *Soft Computing* (2005).

[8] P. Frisco, H. J. Hoogeboom: Simulating counter automata by P systems with symport/antiport. In: [19], 288–301.

[9] S. A. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science* **7** (1978), 311–324.

[10] J. E. Hopcroft, D. Ullman: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley (1979).

[11] O. H. Ibarra, Z. Dang, O. Egecioglu: Catalytic P systems, semilinear sets, and vector addition systems. *Theoretical Computer Science* **312** (1-2) (2004), 379–399.

[12] C. Martín-Vide, A. Păun, G. Păun: On the power of P systems with symport rules. *Journal of Universal Computer Science* **8** (2002), 317–331.

[13] M. L. Minsky: Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Annals of Mathematics* **74** (3) (1961), 437–455.

[14] M. L. Minsky: *Computation: finite and infinite machines.*Prentice-Hall (1967).

[15] B. Monien: Two-way multihead automata over a one-letter alphabet. *Informatique Théorique et Applications* **14** (1) (1980), 67–82.

[16] A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing* **20**(3) (2002), 295–306.

[17] Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, **1** (61) (2002), 108–143.

[18] Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin (2002).

[19] Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing: International Workshop, WMC-CdeA 2002, Curtea de Arges, Romania, August 19-23, 2002. Revised Papers. *Lecture Notes in Computer Science* **2597**, Springer-Verlag, Berlin (2002).

[20] Z. Qi, J. You: P systems and Petri nets. In [1], 387–403.

[21] W. Reisig, G. Rozenberg (Eds.): Lectures on Petri Nets I: Basic Models. *Lecture Notes in Computer Science* **1491**, Springer-Verlag, Berlin (1998).

[22] P. Sosík: The power of catalysts and priorities in membrane systems. *Grammars* **6** (1) (2003), 13–24.

[23] Gy. Vaszil: On the size of P systems with minimal symport/antiport. In: *Pre-proceedings of the Fifth Workshop on Membrane Computing (WMC5).* Milan, Italy (2004), 422–431.

[24] The P Systems Web Page: `http://psystems.disco.unimib.it`

[25] S. Dal Zilio, E. Formenti: On the dynamics of PB systems. In: [1], 197–208.

A Simulator for Conformon-P Systems

Pierluigi FRISCO, Ranulf T. GIBSON

Dept. of Comp. Sci.
School of Eng., C. S., and Math.
University of Exeter
Harrison Building, North Park Road
Exeter, EX4 4QF, U.K.
E-mail: P.Frisco@exeter.ac.uk,
rtg@ranulf.co.uk

Abstract

A simulator of conformon-P systems is presented together with an initial study of an evolution program optimising processes described by conformon-P systems.

1 Introduction

The compartmentalization created by membranes present inside (eucaryotic) cells has been of inspiration to G. Păun [13] for the definition of distributed and parallel theoretical computational models called *Membrane systems* (or *P systems*).

Despite the biological motivations of these devices the first investigations about them had a solely computability focus: the computational aspects of numerous variants of P systems has been (and is) deeply studied. In the later years the attention of researchers on P systems widened to other fields [15, 4] ([16] is the most updated source of information on P systems). The study of biological processes through variants of P systems is one of these new lines of research.

Conformon-P systems [5] are the variant of P systems considered in this paper. They are characterised by a rather simple definition that fits quite well with the modelling of (biological) systems at any scale. Given a conformon-P system modelling a biological process the conformons present in it can represent chemicals, molecules or entire systems, the interaction

between these conformons can describe the interaction between what is represented by them. For instance, two conformons representing two different chemicals can interact to create a molecule represented by another conformon; a conformon representing a molecule can interact with another representing a system (i.e., an organelle or an entire cell), etc.

In this paper we present a simulator of conformon-P systems (Section 4) and its application to the modelling of chemical reactions (the first ten chemical reactions involved in glycolysis are our test case, Section 4.2).

We also report initial results about an evolution program optimising processes described by conformon-P systems (Section 5). Here the target is to create a (evolution) program that can be of help in the study of not well understood processes.

2 Basic Definitions

Let V be a finite alphabet and $\mathbb{N}$ the set of natural numbers and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. A *conformon* is an element of $V \times \mathbb{N}_0 \cup \{+\infty\}$, denoted by $[X, x]$. We will refer to X as the *name* of the conformon $[X, x]$ and to x as the *value* of $[X, x]$. The symbol X will also refer to the conformon itself; the context will help the reader to understand when we refer only to the name aspect of the conformon or to the whole conformon. Moreover let $r = \langle A, e, B \rangle,\ A, B \in V,\ e \in \mathbb{N}$, be a *rule* (also indicated as $A \xrightarrow{e} B$) defining the passage of (part of the) value from one conformon to another so that:

$$([A, a], [B, b]) \Rightarrow ([A, a - e], [B, b + e])$$

with $a, b \in \mathbb{N}_0$, $a \geq e$ indicating that $[A, a]$ and $[B, b]$ *interact* according to r. Informally this means that e is subtracted from the value of the conformon (with name) A and e is added to the value of the conformon (with name) B only if the value of A is at least e.

A *multiset* M (over V) is a function $M : V \to \mathbb{N}_0 \cup \{+\infty\}$; for $d \in V$, $M(d)$ defines the *multiplicity* of d in the multiset M. We will indicate this also with $(d, M(d))$. In case the multiplicity of an element of a multiset is 1 we will indicate just the element. The *support* of a multiset M is the set $supp(M) = \{d \in V \mid M(d) > 0\}$. Informally we will say that an element belongs to a multiset M if it belongs to the support of M. The *size* of a multiset is defined by the function $|\cdot| : (V \to \mathbb{N} \cup +\{\infty\}) \to \mathbb{N} \cup \{+\infty\}$, where for M multiset over V, $|M| = \sum_{a \in supp(M)} M(a)$. Intuitively the size of a multiset M gives the number of elements in $supp(M)$ counted with their multiplicity.

Let $M_1, M_2 : V \to \mathbb{N}_0$ be two multisets. The *union* of M_1 and M_2 is the multiset $M_1 \cup M_2 : V \to \mathbb{N}_0$ defined by $(M_1 \cup M_2)(d) = M_1(d) + M_2(d)$, for all $d \in V$. The *difference* $M_1 \backslash M_2$ is here defined only when M_2 is *included* in M_1 (which means that $M_1(d) \geq M_2(d)$ for all $d \in V$) and it is the multiset $M_1 \backslash M_2 : V \to \mathbb{N}_0$ given by $(M_1 \backslash M_2)(d) = M_1(d) - M_2(d)$ for all $d \in V$.

A *conformon-P system* of *degree* $m, m \geq 1$, is a construct $\Pi = (V, \mu, fin, ack, L_1, \ldots, L_m, R_1, \ldots, R_m)$, where:

V is an alphabet;

$\mu = (N, E)$ is a *directed labelled graph* underlying Π, where:

- $N \subset \mathbb{N}$ contains *vertices*; for simplicity we define $N = \{1, \ldots, m\}$. Each vertex in N defines a region (or *membrane*) of the system Π;
- $E \subseteq N \times N \times pred(\mathbb{N}_0)$ defines directed labeled *edges* between vertices, indicated by (i, j, p) where $p \in pred(n) = \{\geq n, \leq n \mid n \in \mathbb{N}_0\}$ set of *predicates*. So, for instance, an edge can be $(3, 5, \geq 2)$ where the first two elements (3 and 5) indicate nodes, the third one the predicate associated to the edge. The semantics of a predicate $p \in pred(n)$ is the following: given $x \in \mathbb{N}_0$, $p(x)$ may be either $(\geq n)(x)$ indicating $x \geq n$, or $(\leq n)(x)$ indicating $x \leq n$.

$fin \in N$ defines the *final membrane*;

$ack \in N$ the *acknowledgement membrane.*

L_i over $V \times \mathbb{N}_0$, contain conformons associated to region i, $i \in \mathbb{N}$;

R_i are finite sets of rules for conformons interaction associated to region i, $i \in \mathbb{N}$.

Two conformons associated to a membrane i may interact according to a rule r associated to the same membrane such that the multiset of conformons M_i changes into M_i'. So, for $i \in N$, $[A, a], [B, b] \in M_i$ and $r = \langle A, e, B \rangle \in R_i$, $A, B \in V$, $a, b, e \in \mathbb{N}_0$, we have that $M_i' = (M_i \backslash \{[A, a], [B, b]\}) \cup \{[A, a - e], [B, b + e]\}$.

A conformon $[X, x]$ associated to a membrane i may *pass* to a membrane j if $(i, j, p) \in E$ and $p(x)$ holds changing the multisets of conformons M_i and M_j to M_i' and M_j', respectively. In this case $M_i' = M_i \backslash \{[X, x]\}$ and $M_j' = M_j \cup \{[X, x]\}$. The fact that the passage of an object to a membrane

is regulated by some features present in the compartments themselves is already discussed by others in literature when P systems with electrical charge and variable thickness have been considered [14] or only communication was used to compute [12, 11].

The interaction of two conformons according to a rule and the passage of a conformon from one membrane to another are the only *operations* that may be performed by a conformon-P system. A conformon present in a membrane may be involved in one of these two operations or none of them.

A *configuration* of Π is an m-tuple $(M_1, \ldots, M_m)$ of multisets over $V \times \mathbb{N}_0$. The m-tuple $(L_1, \ldots, L_m)$, $supp(L_{ack}) = \emptyset$, is called *initial configuration* (so in the initial configuration the acknowledge membrane does not contain any conformon) while any configuration having $supp(M_{ack}) \neq \emptyset$ is called *final configuration*. For two configurations $(M_1, \ldots, M_m), (M'_1, \ldots, M'_m)$ of Π we write $(M_1, \ldots, M_m) \Rightarrow (M'_1, \ldots, M'_m)$ indicating a *transition* from $(M_1, \ldots, M_m)$ to $(M'_1, \ldots, M'_m)$ that is the parallel application of one or no operation to all the conformons associated to each membrane of μ. In other words in any configuration in which $supp(L_{ack}) \neq \emptyset$ any conformon associated to a membrane can either interact with another conformon associated to the same membrane or pass to another membrane or remain in the same membrane. If no operation is applied to a multiset M_i, then $M_i = M'_i$. The reflexive and transitive closure of $\Rightarrow$ is indicated by $\Rightarrow^*$.

A *computation* is a finite sequence of transitions between configurations of a system Π starting from $(L_1, \ldots, L_m)$. The result of a computation is given by the multisets of conformons associated to membrane fin when any conformon is associated to membrane *ack*. When this happens the computation is halted, that is no other operation is performed even if it could. When a conformon is associated to the acknowledge membrane the number of conformons (counted with their multiplicity) associated to membrane fin define the *number generated* by Π, indicated by $L(\Pi)$.

Conformon-P systems have been proved to be computationally complete [5].

In the following we will use the concept of *module*: a group of membranes with conformons and interaction rules in a conformon-P system able to perform a specific task. A module is not a conformon-P systems as it lacks of the specification of a final and of an acknowledgement membrane.

For a better understanding of the systems presented in this paper figures representing them are provided. In these figures membranes are depicted as enclosed compartments having their label written in **bold** on their right-upper corner. Rules related to a membrane are written inside it; conformons

associated to a membrane in one of the possible configurations of the system are written in normal font.

As a module is supposed to be part of a bigger system it will have edges coming from or going to vertices not defined in the module itself. For this reason in the figures representing modules we depict these vertices with a dashed line; modules, on the other hand, will be indicated by a unique enclosed compartment with a thicker line. Such modules will have a *label* written in **bold** on its right upper corner indicating the kind of module. A subscript is added to differentiate labels referring to the same kind of module present in one system.

In the paper we will use the following labels associated to modules (described in [5]:

spl a *splitter*, a module that when a conformon $[X, x]$ with $x \in \{x_1, \ldots, x_h\}$, $x_i < x_{i+1},\ 1 \leq i \leq h-1$, is associated to a specific membrane of it, it may pass such a conformon to other specific membranes according to its value x.

Inc[x]/Dec[x] a *increaser/decreaser*, a module that when a conformon $[X, x]$ with $x \geq 1$ is associated to a specific membrane of it may decrease or increase the value of such conformon to q, so that $[X, q]$ may pass to another specific membrane.

Edges outgoing a splitter can also have predicates of the form $= n$ for $n \geq 0$ while edges outgoing a increaser/decreaser can also have predicates of the form $= n$ for $n \geq 1$.

Some edges of the modules presented in Section 3 have predicates of the kind $[A, a]$ (a conformon). This is a shorthand for a *separator* module: when conformons of type $[X_i, x],\ 1 \leq i \leq h,\ x \geq 1$ are associated to a specific membrane of it, may pass them to specific different membranes according to their name content. So if there is an edge between membrane 1 and membrane 2 having $[A, a]$ as predicate, it means that only the conformon $[A, a]$ can pass from membrane 1 to membrane 2.

If a membrane contains both the rules $A \xrightarrow{x} B$ and $B \xrightarrow{x} A$, then this is abbreviated with $A \overset{x}{\leftrightarrow} B$.

3 Some Modules

The modelling of chemical reactions with conformon-P systems can be performed in a more direct way if some phenomena (present during chemical reactions) are defined by modules.

Lemma 1 *(Bounded increase) Let* $[A, a]$ *and* $[B, b]$ *two conformons and* x, y,
$a' \in \mathbb{N}$ *such that* $b + x = y \geq 1, a' = a - x \geq 1, y \geq x, a \geq x$ *and* $b \leq y$. *There is a module that allows* $[A, a]$ *and* $[B, b]$ *to interact such that* $[A, a']$ *and* $[B, y]$ *are produced.*

Proof. Figure 1 represents a detailed module for bounded increase. Once in membrane 1 conformons A and B can exchange x value. At any time a conformon with a value bigger equal than y can pass from membrane 1 to $\mathbf{spl_1}$ and from here to membrane 4 only if it is $[B, y]$. In a similar way from membrane 1 conformons with value smaller equal to a' can pass to $\mathbf{spl_2}$ and from here only $[A, a']$ can pass into membrane 5. □

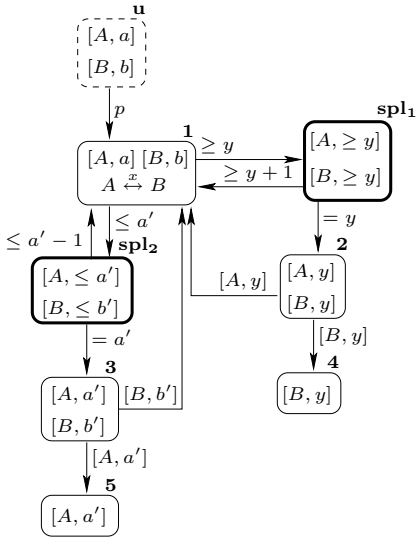


Figure 1: A module for bounded increase

The previous lemma states that the conformon A can pass x value to B even if A could pass $z > x$ to B.

The interaction between two conformons defined in Section 2 cannot model in a direct way the reaction of several chemicals. We prefer to have a module that model this phenomenon.

Lemma 2 *(Multiple interaction) There is a module that when all the conformons* $[X_i, x_i], x_i \geq 1, 0 \leq i \leq n$ *are associated to a specific membrane*

can let the conformons $[Y_j, y_j], y_j \geq 1, 0 \leq j \leq m$ *pass to another specific membrane.*

Proof. Figure 2 represents a detailed module for multiple interaction in which when two conformons A and B are present in membrane 1, then they can interact in a way that conformons C and D will be associated to membrane 4. What represented in Figure 2 can be easily generalised to more conformons in membrane 1 and more conformons in membrane 4, also the values of the conformons can differ.

From membrane 1 a conformon with the sum of the values of all the conformons that have to be present can pass to membrane 2. Here these conformons interact with one of the Y conformons. This last conformon then passes to the other membranes so to create (adding 1 to their value) the remaining Y conformons.

□

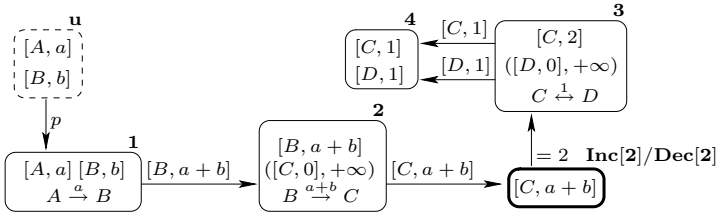


Figure 2: A module for multiple interaction

Notice that in the previous lemma i and j can be different and that the decrease of value of the X_i conformons has to be equal to the increase of value of the Y_j conformons. Anyhow the Y_j can pass to an increaser/decreaser and have their values changed independently from the values of the X_i.

We redefine the interaction between conformons in the following way: $\langle M_1;$
$V_1; M_2; T_2\rangle$ where M_1 and M_2 are vectors of name of conformons, while V_1 and T_2 are vectors with values in $\mathbb{N}_0$. Vectors M_1 and V_1 have the same dimension; similarly for M_2 and T_2.

The meaning of $\langle M_1; V_1; M_2; T_2\rangle$ is: the conformons with names in M_1 can pass the (respectively) values in V_1 to the conformons with names in M_2 and the value of the conformons in M_2 can have at most what is indicated in T_2 (respectively).

For instance let us assume that we want to model the chemical reaction $2H_2 + O_2 \rightharpoonup 2H_2O$ and that we use conformons $[2H_2, m]$ and $[O_2, n]$ indicating that there are m molecules of $2H_2$ and n molecules of O_2. The result of the interaction of these conformons has to create pairs of conformons of the kind $[H_2O, 1]$ indicating one molecule of H_2O (so we do not want to create conformons of the kind $[H_2O, p]$ with $p \geq 2$).

The rule $\langle (H_2, O_2); (2, 1); (H_2O, H_2O); (1, 1) \rangle$ represents what just we described: every H_2 conformon loses 2 units of value and every conformon O loses 1 unit of value, at the same time two conformons with names H_2O get 1 as value. This implies that if a conformon $[H_2O, 1]$ is already present, its value is not going to be increase further (because it is already 1, the maximum value indicated by the rule).

Considering that some chemical reactions are reversible, then the combination of the interactions $\langle M_1; V_1; M_2; T_2 \rangle$ and $\langle M_2; V_2; M_1; T_1 \rangle$ leads to the *general chemical interaction* $\langle M_1; V_1; T_1; M_2; V_2; T_2 \rangle$.
So, for instance, let us assume that we want to model the (reversible) reaction $2SO_2 + O_2 \rightleftharpoons 2SO_3$ and that we use conformons $[SO_2, 2]$ and $[O_2, 1]$ indicating two molecules of SO_2 and one of O_2, respectively, with several copies of each of these conformons. Moreover, there are also conformons with name SO_3 whose value indicates the number of the SO_3 molecule. For some reason we do not want that the value of these last conformons to go above 4.

The rule $\langle (SO_2, O_2);\ (2, 1);\ (2, 1);\ (SO_3);\ (2);\ (4) \rangle$ represents what just described: the value of the conformon SO_3 increases of two units every time the value of one of the conformons with name SO_2 decreases of 2 together with the decrease of 1 of the value of one of the conformons with name O_2. The value of a conformon SO_3 cannot go above 4. A conformon with this name can lose two units and while a conformon SO_2 gets two units and a conformon O_2 gets one unit. The value of a conformon SO_2 cannot go above 2 and the value of a conformon O_2 cannot go above 1.

4 The Simulator

It is not difficult to devise a simulator for conformon-P systems based on an object-oriented programming language. Conformons, interaction rules, membranes and the movement of a conformon from a membrane to another can easily be modelled by objects. This, together with the desire to allow the code to be run on different platforms, leads us to choose Java as the

programming language. Similar choice has been taken in [10, 2] in the creation of a simulator for P systems.

We also wanted to have a simulator that could have the definition of any conformon-P system as input. In this case the choice for the format of the input file has been XML (Section 4.1). The main advantage in the use of XML is the presence of libraries designed to process an XML file. Similar choice has been taken in [2] in the creation of a simulator for P systems.

The simulator program can simulate any conformon-P systems as defined in Section 2, but, as in our tests we modelled chemical reactions (Section 4.2) the implementation of some modules is embedded in the code.

The simulation of the conformon-P system defined in the input file is divided into x time steps. In each time step the simulator implements a transition of the conformon-P system. After the x^{th} time step an output file, describing the configuration of the conformon-P system after this time step, is produced. It is possible to define the format of the output file which can contain the membrane structure, the conformons, the interaction rules, the passage rules and any combination of them.

We will not go into the details of the implementation, but we describe how the simulation of the operations is performed.

Looping through all membranes in the system, either interaction rules or passage rules are simulated first (50% chance).

When rules are simulated one rule per time is chosen at random. A chosen rule can be applied at most c times where c is the number of conformons present in the membrane in that time unit. For each application of the rule it is first checked if all the needed conformons (on the left side of a rule) are available (random choice between the conformons present in the membrane). In an affirmative case the chosen conformons are selected for the rule. If a randomly generated number is smaller than the probability associated to the rule, then the selected conformons are 'tagged'. A 'tagged' conformon is not available to any other operation in that time unit.

When passage rules are simulated each conformon traverses in a random order all the links. If the link can be applied to the conformon, then there is 50% chance of the conformon to pass to another membrane.

It should be clear that this algorithm allows a conformon not to be involved in any interaction or passage rule even if it could.

4.1 The Input File

Appendix A contains an XML file with the definition of a conformon-P system.

The majority of the XML tags are self-explanatory. The ones we believe not self-explanatory are:

`Equation`: indicates the conformons involved in a rule, the valued that are passed and the maximum threshold that can be reached;

`Conformon1`, `Conformon1PassValues`, `Conformon1Limit`, `Conformon2`, `Conformon2PassValues`, `Conformon2Limit`: list, eventually divided by a pipe (`|`), what present in M_1, V_1, T_1, M_2, V_2 and T_2, respectively, in a general chemical interaction;

`Condition`: the allowed values are `le` (less or equal), `ge` (greater or equal) and `e` (equal);

`MustBeId`: it refers to a Separator (see Lemma 2 in [5]) and indicates that only conformons with the specified name verify the predicate.

The tags: `Multiplicity`, `Id` (for a rule), `Reversible`, `Conformon1Limit`, `Conformon2Limit` and `MustBeId` are optional.

In the input file only the conformons present with a value different than 0 in the initial configuration have to be indicated, the remaining conformons are created at runtime by the simulator. In Appendix A, for instance, conformon C is not defined as conformon present in the initial configuration but it appears as argument in an interaction rule.

4.2 The Toy Simulation

The chemical process that we decided to simulate are the first ten chemical reactions of glycolysis [1]. Glycolysis is a very well understood process in a cell and can be an excellent benchmark to test the validity of our simulator.

In this process for each molecule of glucose present at the begin, two molecules of pyruvate are produced. This can only happen only if two molecules of ATP are also available at the begin. The net effect of these ten steps is that for each molecule of glucose and two molecules of ATP, two molecules of NADH, four of ATP and two of pyruvate are created.

This can be expressed by:

$$glucose + 2\ ATP \rightharpoonup 2\ NADH + 4\ ATP + 2\ pyruvate$$

Two of the ten chemical reactions are not reversible (as they involve ATP) and each of them involves an enzyme.

At the end of Section 3 we discussed how with conformon-P systems it is possible to model a (reversible) chemical reaction. Now we describe

how a chemical reaction catalyzed by an enzyme has been modelled. Let us consider the first chemical reaction in glycolysis:

$$glucose + ATM \overset{hexokinase}{\rightharpoonup} glucose6P + ADP + H^{+} \quad (1)$$

catalyzed by *hexokinase*.

We described a chemical reaction not using the enzyme and we associated to it a very low probability. Moreover, we described the reversible chemical reaction

$$glucose + hexokinase \rightleftharpoons glucoseHexokinase \quad (2)$$

and the irreversible one

$$glucoseHexokinase + ATP \rightharpoonup glucose6P + ADP + H^{+} + hexokinase \quad (3)$$

The probability associated to reaction (3) is much bigger than the one associated to reaction (1). Lacking a precise analysis of the dissociation rate (between substrate and enzyme) we gave a random (high) probability to reaction (2).

Appendix B indicates the description of the previous three reactions in the input XML file.

Running the simulator on the first ten chemical reactions present in glycolysis led to the expected result: for each molecule of glucose and two molecules of ATP present at the begin two molecules of NADT, four of ATP and 2 of pyruvate have been created in around 100 time units. This result is promising but not unexpected: the simulated process is quite small (a few chemicals and just ten chemical reactions) and moreover the probabilities associated to each rule were not related to the modelled process.

5 An Evolution Program

Our intention is to create a tool that could be used (for instance by biologist) to simulate known processes and as an help in the study of processes not entirely defined.

We consider the scenario in which a researcher is trying to discover all the steps present in a process. At some stage during this research some parts of the process are known, others are partially known, while others are unknown. We would like to provide such a researcher with a tool that could easily model known part of the process under study but that could also be used to get predictions on what is unknown.

For instance in glycolysis the ratio glucose:ATP present at the beginning of the reaction is 1:2. A simulator using a 1:1 ratio would not have results comparable with the experiments. Knowing part of what happens during glycolysis an evolution program (EP) could be used to approximate the proper ratio (or any other missing part of the process under study).

Evolution programs are used to search for the absolute maximum (minimum) value of high dimensional functions. In the following the way they work is sketched, the reader is referred to [9] for a better description of EPs.

A *population* of candidate solutions (*individuals*) is initial created. Afterwards the individuals *evolve* following a finite and discrete timing (*generations*). In each generation the following operations are performed:

evaluation: each individual is given as input to the function and the returned value (*fitness value*) is associated to it;

selection: depending on their fitness value some individual are selected, the selected individuals form another (new) population;

mutation: some individuals in the new population undergo a modification. If, for instance, an individual is a vector of numbers a modification could be to change one of the numbers.

crossover: some individuals in the new population are used to create others individuals. If, again, each individual is a vector of numbers, then the numbers defining two individuals A and B could be used to create two new individuals C and D.

The individual that during the evolution had the maximum (minimum) fitness is what returned by the EP.

We implemented an EP that searches for the minimum number of conformons (i.e. the minimum sum of the values of some *input* conformons) that can produce in the minimum number of generations a certain *output* conformon in a specified conformon-P system. The EP has two files in input: the one describing the conformon-P system to simulate (glycolysis), and the other one specifying the input conformons (glucose and ATP) and the output ones (pyruvate) and other parameters used during the evolution process. The output of the EP is the minimum value of the input conformons that succeeded in producing the desired output in the minimum number of iterations of the simulated conformon-P system.

The specifications for the EP are given in Appendix C.

The results obtained by the EP indicated that, indeed, a ratio 1:2 for glucose:ATP is the best one for the process we considered. This result is good but it is not surprising as the process we modelled is quite simple and the value of only two elements has to be optimised.

6 Final Remarks

The research reported in this paper is definitely a promising starting point. We are going to extend both the simulator and the evolution program trying to apply them to real world case studies.

The simulator can be extended to include modules arising by other processes. We can also envisage a simulator able to rearrange the conformon-P system given as input in order to perform better (more efficient, accurate, etc.) simulations.

We also think to extend the functionalities of a simulator including the plot of the number of the conformons present in the system during the simulation (similar studies have been reported in [2, 3]).

At the present time the simulator does not consider issues such as the concentration of chemicals (as done in [2]) or other factors influencing a chemical reaction. These elements will be added to the simulator in a later stage.

Of course it is not our intention to simulate a chemical process performing the simulation of each single chemical reaction present in it. We consider to include into the simulator algorithms approximating such processes [6, 7, 8].

It is hard for us to think to a 'universal' evolution program for conformon-P systems as each process can have different representations or function to optimise. Depending on the process under study the evolution program can be extended to include the evolution of rules, links and the membranes themselves. Similarly the function to optimise can be related to any of the variable aspects (number of initial conformons, time steps, number of usage of certain rules, etc.) present in a conformon-P system.

Acknowledgements

This work has been supported by the research grant NAL/01143/G of The Nuffield Foundation.

A An Input File

```
<XML version="1.0">
  <Psystem>
  <Membrane>
    <Id>M1</Id>
    <Conformon>
      <Id>A</Id>
      <Value>2</Value>
      <Multiplicity>3</Multiplicity>
    </Conformon>
    <Conformon>
      <Id>B</Id>
      <Value>1</Value>
    </Conformon>
    <Rule>
      <Id>r1</Id>
      <Equation>
        <Conformon1>A</Conformon1>
        <Conformon2>B</Conformon2>
        <Conformon1PassValues>1</Conformon1PassValues>
        <Conformon2PassValues>1</Conformon2PassValues>
        <Reversable>false</Reversable>
      </Equation>
      <Probability>1</Probability>
    </Rule>
    <Rule>
      <Id>r2</Id>
      <Equation>
        <Conformon1>A|B</Conformon1>
        <Conformon2>C</Conformon2>
        <Conformon1PassValues>1|1</Conformon1PassValues>
        <Conformon2PassValues>1</Conformon2PassValues>
        <Conformon1Limit>1|1</Conformon1Limit>
        <Conformon2Limit>2</Conformon2Limit>
        <Reversable>true</Reversable>
      </Equation>
      <Probability>0.8</Probability>
    </Rule>
    <Link>
      <Linkto>M2</Linkto>
      <Value>6</Value>
      <Condition>le</Condition>
      <Probability>1</Probability>
      <MustBeId>C</MustBeId>
    </Link>
    <Link>
      <Linkto>M3</Linkto>
      <Value>1</Value>
      <Condition>be</Condition>
      <Probability>0.1</Probability>
    </Link>
  </Membrane>
  <Membrane>
    <Id>M2</Id>
  </Membrane>
```

```
  <Membrane>
    <Id>M3</Id>
  </Membrane>
  </Psystem>
```

B Three Chemical Reactions

```
\scriptsize
    <Rule>
      <Equation>
        <Conformon1>Glucose|ATP|Enz.Hexokinase</Conformon1>
        <Conformon2>Glucose-6-phosphate|ADP|H+|Enz.Hexokinase</Conformon2>
        <Conformon1PassValues>1|1|1</Conformon1PassValues>
        <Conformon2PassValues>1|1|1|1</Conformon2PassValues>
        <Reversable>false</Reversable>
      </Equation>
      <Probability>0.0000001</Probability>
    </Rule>
    <Rule>
      <Equation>
        <Conformon1>Glucose|Enz.Hexokinase</Conformon1>
        <Conformon2>Glucose.Hexokinase--Attached</Conformon2>
        <Conformon1PassValues>1|1</Conformon1PassValues>
        <Conformon2PassValues>1</Conformon2PassValues>
        <Reversable>true</Reversable>
      </Equation>
      <Probability>0.8</Probability>
    </Rule>
    <Rule>
      <Equation>
        <Conformon1>Glucose.Hexokinase--Attached</Conformon1>
        <Conformon2>Glucose-6-phosphate|ADP|H+|Enz.Hexokinase</Conformon2>
        <Conformon1PassValues>1</Conformon1PassValues>
        <Conformon2PassValues>1|1|1|1</Conformon2PassValues>
        <Reversable>false</Reversable>
      </Equation>
      <Probability>1</Probability>
    </Rule>
```

C Specifications of the Evolution Program

An individual of the EP is a vector with the values of the input conformons.

The fitness function to minimize is:

`((<time steps taken to reach the final configuration> * <SimulatorPenalty>) * <SimulatorWeighting>) + (<sum values input conformons> * <ChromosomeValueWeighting>)`

where the sum of the elements of an individual are `<sum values input conformons>` while `<time steps taken to reach the final configura-`

tion> indicates how many times steps an individual needed to reach the final configuration.This value can be at most SimulatorIterations (see below).

The file with the specifications of the EP is an XML file whose tags are:

SimulatorIterations: number of iterations of the simulator for each individual;

SimulatorPenalty: in case an individual does not reach TargetConformon then in its fitness SimulatorIterations is multiplied by SimulatorPenalty;

SimulatorWeighting: how much weight is given to the number of generations needed to reach TargetConformon (see below);

ChromosomeValueWeighting: how much weight is given to the number of conformons present in the initial configuration (see below);

PopulationSize: size of the population (number of individuals);

UpperConformonLimit: the maximum value that the conformons in the initial configuration can have. The bigger the number the bigger the range of values and the smaller the probability to guess a smaller value (if this leads to a fast solution);

Generations: number of generations the EP runs;

NumberOfChromosomesToMutate: how many individual mutate in each generation;

HowManyMutationsPerChromosome: how many values of the input conformons are mutated;

NumberOfCrossoversToPerform: half of the crossovers performed in each generation (if 1, it means that 2 individuals are chosen to crossover);

TournamentSelection: how many individual are picked up to be selected for the next population according to tournament selection (see [9]);

SimulationAverageRuns: for each individual simulator is run Simulation AverageRuns times (and each time for SimulatorIterations iterations). The time steps taken to reach the final configuration is the average of the time steps in the SimulationAverageRuns simulations;

`StartConformon:` the name of the conformon present in the `StartMembrane` membrane defined in the conformon-P system to simulate that represent the input conformons (this tag can be repeated if there are more than one conformon);

`StartMembrane:` membrane defined in the conformon-P system to simulate containing the input conformons;

`TargetConformon:` it has two sub-tags:

`Id:` name of target output conformon present in `TargetMembrane`;

`Value:` value of target output conformon present in `TargetMembrane`;

`TargetMembrane:` membrane defined in the conformon-P system to simulate where the output conformons should be present.

`SimulatorWeighting` and `ChromosomeValueWeighting` are used to indicate what is more important in a solution. If we are looking for solutions having fast (small number of time steps) simulation time, then `SimulatorWeighting` will be smaller than `ChromosomeValueWeighting`. In this way if two individuals reach the `TargetConformon` using the same initial configuration but one individual uses less time steps than the other, then the former gets an higher fitness value. If we are looking for solutions having a small number of conformons in the initial configuration, then `ChromosomeValueWeighting` will be smaller than `SimulatorWeighting`.

The output of the EP is on screen: it just indicates the progress of the simulation and at the end of it the fitness of the best individual with the value of each input conformon.

References

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular biology of the cell*. Garland Publishing, **fourth edition** (2002).

[2] L. Bianco: Introduction to Psim. Downloaded from [16].

[3] M. Cavaliere, I.I. Ardelean: Modelling respiration in bacteria and respiration/photosynthesis interaction in Cyanobacteria by using a P system simulator. In [4] (2005), 127–156.

[4] G. Ciobanu, Gh. Păun, M.J. Prez-Jimnez (Eds.): *Applications of Membrane Computing*. Springer-Verlag, Berlin (2005).

[5] P. Frisco: The conformon-P system: A molecular and cell biology-inspired comutability model. *Theoretical Computer Science* **312** (2-3) (2004), 295–319.

[6] D. T. Gillespie: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *J.Comp. Phys* **22** (1976), 403–434.

[7] D. T. Gillespie: Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem* **81** (1977), 2340–2361.

[8] T. Lu, D. Volfson, L. Tsimring, J. Hasty: Cellular growth and division in the gillespie algorithm. *IEE Systems Biology* **1** (2004), 121–127.

[9] Z. Michalewicz: *Genetic algorithms + data structures = evolution programs*. Springer-Verlag, Berlin (1999).

[10] I.A. Nepomuceno-Chamorro: A Java simulator for basic transition P systems. *Journal of Universal Computer Science* **10** (5) (2004), 620–629.

[11] A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing* **20** (3) (2002), 295–306.

[12] A. Păun, Gh. Păun, G. Rozenberg: Computing by communication in networks of membranes. *International Journal of Foundations of Computer Science* **13** (2002), 779–798.

[13] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, **1** (61) (2000), 108–143.

[14] Gh. Păun: P systems with active membranes: Attacking NP complete problems. *Journal of Automata, Languages and Combinatorics* **1** (6) (2001), 75–90.

[15] Gh. Păun: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin (2002).

[16] The P Systems Web Page `http://psystems.disco.unimib.it`

On the Power of Dissolution in P Systems with Active Membranes

Miguel A. GUTIÉRREZ-NARANJO,
Mario J. PÉREZ-JIMÉNEZ,
Agustín RISCOS-NÚÑEZ,
Francisco J. ROMERO-CAMPERO

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: {magutier,marper,ariscosn,fran}@us.es

Abstract

In this paper we study the membrane dissolution rules in the framework of P systems with active membranes but without using electrical charges. More precisely, we prove that the polynomial computational complexity class associated with the class of recognizer P systems with active membranes, without polarizations and without dissolution is equal to the standard complexity class **P**. Furthermore, we demonstrate that if we consider dissolution rules, then the resulting complexity class contains the class **NP**.

1 Introduction

Membrane Computing is inspired by the structure and functioning of living cells, and it provides a new non–deterministic model of computation which starts from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations. The devices of this model are called *P systems*.

Roughly speaking, a P system consists of a cell-like membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules in a synchronous non–deterministic maximally parallel manner.

In this paper we work with P systems with active membranes. This model was introduced in [11], abstracting the way of obtaining new membranes through the process of *mitosis* (membrane division) and providing a tool able to construct an exponential workspace in linear time. In these devices membranes have polarizations, one of the "electrical charges" $0, -, +$, and several times the problem was formulated whether or not these polarizations are necessary in order to obtain polynomial solutions to **NP**–complete problems. The last result is that from [1], where one proves that two polarizations suffice.

P systems with active membranes have been successfully used to design (uniform) solutions to well-known **NP**–complete problems, such as SAT [20], *Subset Sum* [17], *Knapsack* [18], *Bin Packing* [19], *Partition* [4], and the *Common Algorithmic Problem* [21].

The present paper can be considered as a contribution to the interesting problem of characterizing the tractability in terms of descriptional resources required in membrane systems.

Specifically, in the framework of recognizer P systems with membrane division but without using polarizations we prove the following: (a) the class of problems which can be solved in a polynomial time by a family of such P systems *without dissolution* is equal to class **P**, and (b) the class of problems which can be solved in a polynomial time by a family of such P systems *with dissolution* contains the class **NP**. Hence, we show a surprising role of the –apparently "innocent"– operation of membrane dissolution, as it makes the difference between efficiency and non–efficiency for P systems with membrane division and without polarization.

The paper is organized as follows. In the next section some preliminary ideas about recognizer membrane systems and polynomial complexity classes are introduced. In Section 3 we present a characterization of the class **P** through the polynomial complexity class associated with recognizer P systems with active membranes, without polarization and without dissolution. In Section 4 we show that every **NP**–complete problem can be solved in a semi–uniform way by families of recognizer P systems using membrane dissolution rules and division for elementary and non–elementary membranes. Conclusions and some final remarks are given in Section 5.

2 Preliminaries

2.1 Recognizer P Systems

In the structure and functioning of a cell, biological *membranes* play an essential role. The cell is separated from its environment by means of a *skin membrane*, and it is internally compartmentalized by means of *internal membranes*.

The main *syntactic* ingredients of a cell–like membrane system (P system) are the *membrane structure*, the *multisets*, and the *evolution rules*.

- A *membrane structure* consists of several membranes arranged hierarchically inside a main membrane (the *skin*), and delimiting *regions* (the space in–between a membrane and the immediately inner membranes, if any). Each membrane identifies a region inside the system. When a membrane has no membrane inside, it is called *elementary*. A membrane structure can be considered as a rooted tree, where the nodes are called *membranes*, the root is called *skin*, and the leaves are called *elementary membranes*.

- Regions defined by a membrane structure contain objects corresponding to chemical substances present in the compartments of a cell. The objects can be described by symbols or by strings of symbols, in such a way that *multisets of objects* are placed in the regions of the membrane structure.

- The objects can evolve according to given *evolution rules*, associated with the regions (hence, with the membranes).

The *semantics* of the cell–like membrane systems is defined through a non–deterministic and synchronous model (a global clock is assumed) as follows:

- A *configuration* of a cell–like membrane system consists of a membrane structure and a family of multisets of objects associated with each region of the structure. At the beginning, there is a configuration called the *initial configuration* of the system.

- In each time unit we can transform a given configuration in another configuration by applying the evolution rules to the objects placed inside the regions of the configurations, in a non–deterministic, maximally parallel manner (the rules are chosen in a non–deterministic

way, and in each region all objects that can evolve must do it). In this way, we get *transitions* from one configuration of the system to the next one.

- A *computation* of the system is a (finite or infinite) sequence of configurations such that each configuration –except the initial one– is obtained from the previous one by a transition.
- A computation which reaches a configuration where no more rules can be applied to the existing objects and membranes, is called a *halting computation.*
- The result of a halting computation is usually defined through the multiset associated with a specific output membrane (or the environment) in the final configuration.

In this paper we use membrane computing as a framework to address the resolution of decision problems. In order to solve this kind of problems and having in mind that solving them is equivalent to recognizing the language associated with them, we consider P systems as *recognizer language* devices.

Definition 2.1 *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where: (a)* Π *is a P system with working alphabet* Γ*, with* p *membranes labelled with* $1, \ldots, p$*, and initial multisets* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *associated with them; (b)* Σ *is an (input) alphabet strictly contained in* Γ *and the initial multisets are over* $\Gamma - \Sigma$*; (c)* i_Π *is the label of a distinguished (input) membrane.*

The computations of a P system with input in the form of a multiset over Σ are defined in a natural way, but the initial configuration of (Π, Σ, i_Π) must be the initial configuration of the system Π to which we add the input multiset. More formally,

Definition 2.2 *Let* (Π, Σ, i_Π) *be a P system with input. Let* Γ *be the working alphabet of* Π*,* μ *the membrane structure, and* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *the initial multisets of* Π*. Let* m *be a multiset over* Σ*. The* initial configuration of (Π, Σ, i_Π) with input m *is* $(\mu, \mathcal{M}_1, \ldots, \mathcal{M}_{i_\Pi} \cup m, \ldots, \mathcal{M}_p)$.

Let (Π, Σ, i_Π) be a P system with input. Let Γ be the working alphabet of Π, μ the membrane structure, and $\mathcal{M}_1, \ldots, \mathcal{M}_p$ the initial multisets of Π. Let m be a multiset over Σ. Then we denote $\mathcal{M}_j^* = \{(a, j) : \ a \in \mathcal{M}_j\}$, for $1 \leq j \leq p$, and $m^* = \{(a, i_\Pi) : \ a \in m\}$.

Let us recall that a decision problem X is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (the elements are called *instances*) and θ_X is a predicate (a total boolean function) over I_X.

Definition 2.3 *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of P systems with input. A* polynomial encoding *from X to $\mathbf{\Pi}$ is a pair (cod, s) of polynomial time computable functions over I_X such that for each instance $w \in I_X$, $s(w)$ is a natural number and $cod(w)$ is an input multiset for the system $\Pi(s(w))$.*

Polynomial encodings are stable under polynomial time reductions [20]. More precisely, the following proposition holds.

Proposition 2.1 *Let X_1, X_2 be decision problems. Let r be a polynomial time reduction from X_1 to X_2. Let (cod, s) be a polynomial encoding from X_2 to $\mathbf{\Pi}$. Then $(cod \circ r, s \circ r)$ is a polynomial encoding from X_1 to $\mathbf{\Pi}$.*

Definition 2.4 *A* recognizer P system *is a P system with input and external output such that:*

1. *The working alphabet contains two distinguished elements* `yes` *and* `no`.
2. *All computations halt.*
3. *If $\mathcal{C}$ is a computation of the system, then either object* `yes` *or object* `no` *(but not both) must have been released into the environment, and only in the last step of the computation.*

In recognizer P systems, we say that a computation is an *accepting computation* (respectively, *rejecting computation*) if the object `yes` (respectively, `no`) appears in the environment associated with the corresponding halting configuration.

2.2 Recognizer P Systems with Active Membranes and without Polarizations

A particularly interesting class of membrane systems are the systems with active membranes, where the membrane division can be used in order to solve computationally hard problems in polynomial or even linear time, by a space–time trade-off.

In this paper we work with a variant of P systems with active membranes that does not use polarizations.

Definition 2.5 *A P system with active membranes and without polarizations is a P system with* Γ *as working alphabet, with* H *as the finite set of labels for membranes, and where the rules are of the following forms:*

(a) $[\, a \to u \,]_h$ *for* $h \in H$, $a \in \Gamma$, $u \in \Gamma^*$. *This is an object evolution rule, associated with a membrane labelled with* h*: an object* $a \in \Gamma$ *belonging to that membrane evolves to a string* $u \in \Gamma^*$.

(b) $a\,[\ \ ]_h \to [\, b \,]_h$ *for* $h \in H$, $a, b \in \Gamma$. *An object from the region immediately outside a membrane labelled with* h *is introduced in this membrane, possibly transformed into another object.*

(c) $[\, a \,]_h \to b\ [\ \ ]_h$ *for* $h \in H$, $a, b \in \Gamma$. *An object is sent out from membrane labelled with* h *to the region immediately outside, possibly transformed into another object.*

(d) $[\, a \,]_h^\alpha \to b$ *for* $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in \Gamma$*: A membrane labelled with* h *is dissolved in reaction with an object. The skin is never dissolved.*

(e) $[\, a \,]_h \to [\, b \,]_h\ [\, c \,]_h$ *for* $h \in H$, $a, b, c \in \Gamma$. *An elementary membrane can be divided into two membranes with the same label, possibly transforming some objects.*

These rules are applied according to the following principles:

- All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non–deterministic way), but any object which can evolve by one rule of any form, must evolve.
- If at the same time a membrane labelled with h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. Of course, this process takes only one step.
- The rules associated with membranes labelled with h are used for all copies of this membrane. At one step, a membrane can be the subject of *only one* rule of types (b)-(e).

Let us note that in this framework we work without cooperation, without priorities, with cell division rules for elementary membranes, and without changing the labels of membranes. But we can use dissolution or not.

We denote by $\mathcal{AM}^0_{-d}$ (respectively, $\mathcal{AM}^0_{+d}$) the class of all recognizer P systems with active membranes without polarizations and without using dissolution (respectively, using dissolution).

2.3 Polynomial Complexity Classes in Recognizer P Systems

Definition 2.6 *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(w))_{w \in I_X}$ be a family of recognizer membrane systems without input.*

- *$\mathbf{\Pi}$ is sound with regard to X if for each instance of the problem $w \in I_X$, if there exists an accepting computation of $\Pi(w)$, then $\theta_X(w) = 1$.*
- *$\mathbf{\Pi}$ is complete with regard to X if for each instance of the problem $w \in I_X$, if $\theta_X(w) = 1$, then every computation of $\Pi(w)$ is an accepting computation.*

These concepts can be extended to families of recognizer P systems with input membrane.

Definition 2.7 *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer P systems with input. Let (cod, s) be a polynomial encoding from X to $\mathbf{\Pi}$.*

- *We say that the family $\mathbf{\Pi}$ is sound with regard to (X, cod, s) if the following holds: for each instance of the problem $w \in I_X$, if there exists an accepting computation of $\Pi(s(w))$ with input $cod(w)$, then $\theta_X(w) = 1$.*
- *We say that the family $\mathbf{\Pi}$ is complete with regard to (X, cod, s) if the following holds: for each instance of the problem $u \in I_X$, if $\theta_X(u) = 1$, then every computation of $\Pi(s(u))$ with input $cod(u)$ is an accepting computation.*

The first results about *solvability* of **NP**–complete problems in polynomial time (even linear) by membrane systems were given by Gh. Păun [10], C. Zandron, C. Ferretti and G. Mauri [23], S.N. Krishna and R. Rama [7], and A. Obtułowicz [8] in the framework of P systems that lack an input membrane. Thus, the constructive proofs of such results need to design *one* system for *each* instance of the problem.

This method for solving problems provides an algorithmic solution of *specific purpose* in the following sense: if we wanted to apply such a method

to some decision problem in a laboratory, then the system constructed to solve a concrete instance is useless when trying to solve another instance.

Now, we formalize these ideas in the following definition.

Definition 2.8 *Let $\mathcal{R}$ be a class of recognizer P systems without input membrane. A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family, $\mathbf{\Pi} = (\Pi(w))_{w \in I_X}$, of P systems of type $\mathcal{R}$, and we denote this by $X \in \mathbf{PMC}^*_{\mathcal{R}}$, if:*

- $\mathbf{\Pi}$ *is polynomially uniform by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(w)$ from the instance $w \in I_X$.*
- $\mathbf{\Pi}$ *is polynomially bounded, that is, there exists a polynomial function $p(n)$ such that for each $w \in I_X$, all computations of $\Pi(w)$ halt in at most $p(|w|)$ steps.*
- $\mathbf{\Pi}$ *is sound and complete with regard to X.*

Next, we propose to solve a decision problem through a family of P systems constructed in polynomial time by a Turing machine, and verifying that each element of the family processes, in a specified sense, all the instances of *equivalent size*. We say that these solutions are *uniform solutions*.

Definition 2.9 *Let $\mathcal{R}$ be a class of recognizer P systems with input membrane. A decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$, of P systems from $\mathcal{R}$, and we denote this by $X \in \mathbf{PMC}_{\mathcal{R}}$, if the following is true:*

- *The family $\mathbf{\Pi}$ is polynomially uniform by Turing machines.*
- *There exists a polynomial encoding (cod, s) from I_X to $\mathbf{\Pi}$ such that*
  - *The family $\mathbf{\Pi}$ is polynomially bounded with regard to (X, cod, s); that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and, moreover, it performs at most $p(|u|)$ steps.*
  - *The family $\mathbf{\Pi}$ is sound and complete with regard to (X, cod, s).*

It is easy to see that the classes $\mathbf{PMC}^*_{\mathcal{R}}$ and $\mathbf{PMC}_{\mathcal{R}}$ are closed under polynomial–time reduction and complement (see [15] for details).

3 Characterizing the Tractability by Recognizer P Systems with Active Membranes

Let Π be a recognizer P systems with active membranes without polarizations and without dissolution. Let R be the set of rules associated with Π.

Each rule can be considered, in a certain sense, as a *dependency* between the object triggering the rule and the object or objects produced by its application.

We can consider a general pattern for rules of types $(a), (b), (c), (e)$ in the form $(a, h) \to (a_1, h')(a_2, h') \dots (a_s, h')$, where:

- The rules of type (a) correspond to the case $h = h'$ and $s \geq 1$.
- The rules of type (b) correspond to the case $h = f(h')$ and $s = 1$.
- The rules of type (c) correspond to the case $h' = f(h)$ and $s = 1$.
- The rules of type (e) correspond to the case $h = h'$ and $s = 2$.

If h is the label of a membrane, then $f(h)$ denotes the label of the father of the membrane labelled with h. We adopt the convention that the father of the skin membrane is the environment (and we denote by *environment* the label associated with the environment of the system).

For example, let us consider a general rule $(a, h) \to (a_1, h') \dots (a_s, h')$. Then we can interpret that from the object a in membrane labelled with h we can *reach* the objects $a_1, \dots, a_s$ in membrane labelled with h'.

Next, we formalize these ideas in the following definition.

Definition 3.1 *Let Π be a recognizer P system with active membranes without polarizations and without dissolution. Let R be the set of rules associated with Π. The dependency graph associated with Π is the directed graph $G_\Pi = (V_\Pi, E_\Pi)$ defined as follows:*

$$V_\Pi = VL_\Pi \cup VR_\Pi,$$

$$\begin{aligned} VL_\Pi = \{(a,h) \in \Gamma \times H : {} & \exists u \in \Gamma^* \ ([a \to u]_h \in R) \vee \\ & \exists b \in \Gamma \ ([a]_h \to [\ ]_h b \in R) \vee \\ & \exists b \in \Gamma \ \exists h' \in H \ (h = f(h') \wedge a[\ ]_{h'} \to [b]_{h'} \in R) \vee \\ & \exists b, c \in \Gamma \ ([a]_h \to [b]_h [c]_h \in R)\}, \end{aligned}$$

$$VR_\Pi = \{(b,h) \in \Gamma \times H : \exists a \in \Gamma\ \exists u \in \Gamma^*\ ([a \to u]_h \in R \wedge b \in alph(u)) \vee$$

$$\exists a \in \Gamma\ \exists h' \in H\ (h = f(h') \wedge [a]_{h'} \to [\ ]_{h'} b \in R) \vee$$

$$\exists a \in \Gamma\ (a[\ ]_h \to [b]_h \in R) \vee$$

$$\exists a, c \in \Gamma\ ([a]_h \to [b]_h[c]_h \in R)\},$$

$$E_\Pi = \{((a,h),(b,h')) : \exists u \in \Gamma^*\ ([a \to u]_h \in R \wedge b \in alph(u) \wedge h = h') \vee$$

$$([a]_h \to [\ ]_h b \in R\ \wedge\ h' = f(h)) \vee$$

$$(a[\ ]_{h'} \to [b]_{h'} \in R\ \wedge h\ = f(h')) \vee$$

$$\exists c \in \Gamma\ ([a]_h \to [b]_h[c]_h \in R \wedge h = h')\}.$$

Proposition 3.1 *Let Π be a recognizer P system with active membranes without polarizations and without dissolution. Let R be the set of rules associated with Π. There exists a Turing machine that constructs the dependency graph, G_Π, associated with Π, in polynomial time (that is, in a time bounded by a polynomial function depending on the total number of rules and the maximum length of the rules).*

Proof. A deterministic algorithm that, given a P system Π with the set R of rules, constructs the corresponding dependency graph, is the following:

```
Input:  Π (with R as its set of rules)
V_Π ← ∅;  E_Π ← ∅
for each rule r ∈ R of Π do
  if r = [a → u]_h ∧ alph(u) = {a_1, ..., a_s} then
    V_Π ← V_Π ∪ ⋃_{j=1}^{s} {(a,h),(a_j,h)};  E_Π ← E_Π ∪ ⋃_{j=1}^{s} {((a,h),(a_j,h))}
  if r = [a]_h → [ ]_h b then
    V_Π ← V_Π ∪ {(a,h),(b,f(h))};
    E_Π ← E_Π ∪ {((a,h),(b,f(h)))}
  if r = a[ ]_h → [b]_h then
    V_Π ← V_Π ∪ {(a,f(h)),(b,h)};
    E_Π ← E_Π ∪ {((a,f(h)),(b,h))}
```

`if` $r = [a]_h \rightarrow [b]_h[c]_h$ `then`
$V_\Pi \leftarrow V_\Pi \cup \{(a,h)), (b,h), (c,h)\}$;
$E_\Pi \leftarrow E_\Pi \cup \{((a,h)), (b,h)), ((a,h), (c,h))\}$

The running time of this algorithm is bounded by $O(|R| \cdot q)$, where q is the value $\max\{length(r) : \ r \in R\}$. □

Proposition 3.2 *Let* $\Pi = (\Gamma, \Sigma, H, \mathcal{M}_1, \ldots, \mathcal{M}_p, R_1, \ldots, R_p, i_\Pi)$ *be a recognizer P system with active membranes without polarizations and without dissolution. Let* Δ_Π *be defined as follows:*

$$\Delta_\Pi = \{(a,h) \in \Gamma \times H : \ \textit{there exists a path (within the dependency graph) from } (a,h) \textit{ to } (\texttt{yes}, environment)\}.$$

Then, there exists a Turing machine that constructs the set Δ_Π *in polynomial time (that is, in a time bounded by a polynomial function depending on the total number of rules and the maximum length of the rules).*

Proof. We can construct the set Δ_Π from Π as follows:

- We construct the dependency graph G_Π associated with Π.
- Then we consider the following algorithm:

`Input:` $G_\Pi = (V_\Pi, E_\Pi)$
$\Delta_\Pi \leftarrow \emptyset$
`for each` $(a,h) \in V_\Pi$ `do`
`if reachability` $(G_\Pi, (a,h), (yes, environment)) = yes$ `then`
$\Delta_\Pi \leftarrow \Delta_\Pi \cup \{(a,h)\}$

The running time of this algorithm is of the order $O(|V_\Pi| \cdot |V_\Pi|^2)$, hence[1] it

[1]The Reachability Problem is the following: *given a (directed or undirected) graph, G, and two nodes a, b, determine whether or not the node b is reachable from a, that is, whether or not there exists a path in the graph from a to b.* It is easy to design an algorithm running in polynomial time solving this problem. For example, given a (directed or undirected) graph, G, and two nodes a, b, we consider a depth–first–search with source a, and we check if b is in the tree of the computation forest whose root is a. The total running time of this algorithm is $O(|V|+|E|)$, that is, in the worst case is quadratic in the number of nodes. Morover, this algorithm needs to store a linear number of items (it can be proved that there exists another polynomial time algorithm which uses $O(\log^2(|V|))$ space).

is of the order $O(|\Gamma|^3 \cdot |H|^3)$. □

Next, given a family of recognizer P systems solving a decision problem, we will characterize the acceptance of an instance of the problem, w, using the set $\Delta_{\Pi(s(w))}$ associated with the system $\Pi(s(w))$, that processes the given instance w. More precisely, the instance is accepted by the system if and only if there is an object in the initial configuration of the system $\Pi(s(w))$ with input $cod(w)$ such that there exists a path in the associated dependency graph reaching the object `yes` in the environment.

Proposition 3.3 *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ be a family of recognizer P systems with input membrane solving X, according to Definition 2.9. Let (cod, s) be the polynomial encoding associated with that solution. Then, for each instance w of the problem X the following assertions are equivalent:*

(a) $\theta_X(w) = 1$ (that is, the answer to the problem is `yes` for w).

(b) $\Delta_{\Pi(s(w))} \cap ((cod(w))^* \cup \bigcup_{j=1}^{p} \mathcal{M}_j^*) \neq \emptyset$, where $\mathcal{M}_1, \ldots, \mathcal{M}_p$ are the initial multisets of the system $\Pi(s(w))$.

Proof. Let $w \in I_X$. Then $w \in L_X$ if and only if there exists an accepting computation of the system $\Pi(s(w))$ with input multiset $cod(w)$. But this condition is equivalent to the following: in the initial configuration of $\Pi(s(w))$ with input multiset $cod(w)$ there exists an object $a \in \Gamma$ in a membrane labelled with h such that in the dependency graph the node $(\mathtt{yes}, environment)$ is reachable from (a, h).

Hence, $\theta_X(w) = 1$ if and only if $\Delta_{\Pi(s(w))} \cap \mathcal{M}_1^* \neq \emptyset$, or $\ldots$, or $\Delta_{\Pi(s(w))} \cap \mathcal{M}_p^* \neq \emptyset$, or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$. □

Theorem 3.1 $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0_{-d}}$.

Proof. We have $\mathbf{P} \subseteq \mathbf{PMC}_{\mathcal{AM}^0_{-d}}$ because the class $\mathbf{PMC}_{\mathcal{AM}^0_{-d}}$ is closed under polynomial time reduction. Next, we show that $\mathbf{PMC}_{\mathcal{AM}^0_{-d}} \subseteq \mathbf{P}$. Let $X \in \mathbf{PMC}_{\mathcal{AM}^0_{-d}}$ and let $\mathbf{\Pi} = (\Pi(n))_{n\in\mathbf{N}}$ be a family of recognizer P systems with input membrane solving X, according to Definition 2.9. Let (cod, s) be the polynomial encoding associated with that solution.

We consider the following deterministic algorithm:

```
Input:  An instance w of X
  - Construct the system Π(s(w)) with input multiset cod(w).
  - Construct the dependency graph G_{Π(s(w))} associated with
Π(s(w)).
  - Construct the set Δ_{Π(s(w))} according to Proposition 3.2
    answer ← no; j ← 1
    while j ≤ p ∧ answer = no do
       if Δ_{Π(s(w))} ∩ M_j^* ≠ ∅ then
          answer ← yes
       j ← j + 1
    endwhile
    if Δ_{Π(s(w))} ∩ (cod(w))^* ≠ ∅ then
       answer ← yes
```

On one hand, the answer of this algorithm is `yes` if and only if there exists a pair (a, h) belonging to $\Delta_{\Pi(s(w))}$ such that the symbol a appears in the membrane labelled with h in the initial configuration (with input the multiset $cod(w)$).

On the other hand, a pair (a, h) belongs to $\Delta_{\Pi(s(w))}$ if and only if there exists a path from (a, h) to $(\texttt{yes}, environment)$, that is, if and only if we can obtain an accepting computation of $\Pi(s(w))$ with input $cod(w)$. Hence, the algorithm above described solves the problem X.

The cost to determine whether or not $\Delta_{\Pi(s(w))} \cap M_j^* \neq \emptyset$ (or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$) is of the order $O(|\Gamma|^2 \cdot |H|^2)$.

Hence, the running time of this algorithm can be bounded by $f(|w|) + O(|R| \cdot q) + O(p \cdot |\Gamma|^2 \cdot |H|^2)$, where f is the (total) cost of a polynomial encoding from X to $\mathbf{\Pi}$, R is the set of rules of $\Pi(s(w))$, H is the set of labels for membranes of $\Pi(s(w))$, p is the number of (initial) membranes of $\Pi(s(w))$, and $q = \max\{length(r) : r \in R\}$. But from Definition 2.9 we have that all involved parameters are polynomials in $|w|$. That is, the algorithm is polynomial in the size $|w|$ of the input. □

Now, we consider *division rules for non–elementary membranes*, that is, rules of the following form $[\,[\,]_{h_1}[\,]_{h_2}\,]_{h_0} \rightarrow [\,[\,]_{h_1}\,]_{h_0}\,[\,[\,]_{h_2}\,]_{h_0}$, where h_0, h_1, h_2 are labels: if the membrane with label h_0 contains other membranes than those with labels h_1, h_2, then such membranes and their contents are duplicated and placed in both new copies of the membrane h_0; all membranes and

objects placed inside membranes h_1, h_2, as well as the objects from membrane h_0 placed outside membranes h_1 and h_2, are reproduced in the new copies of membrane h_0. We denote by $\mathcal{AM}^0_{-d,+ne}$ the class of all recognizer P systems with active membranes without polarization, without membrane dissolution rules, and using division rules for elementary and non–elementary membranes.

If $\Pi \in \mathcal{AM}^0_{-d,+ne}$, then we define the dependency graph associated with Π as the directed graph G_Π from Definition 3.1, that is, the division rules for non–elementary membranes do not add any node or edge to the dependency graph.

Then, the proof of Theorem 3.1 provides the following result:

Theorem 3.2 $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^0_{-d,+ne}}$.

Now, we study similar characterizations of **P** dealing with semi–uniform solutions in the framework of recognizer P systems with active membranes without polarizations and without dissolution.

Proposition 3.4 *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(w))_{w \in I_X}$ be a family of recognizer P systems without input membrane solving X, according to Definition 2.8. Then, for each instance w of the problem X the following assertions are equivalent:*

(a) $\theta_X(w) = 1$ (*that is, the answer to the problem is* `yes` *for* w).

(b) $\Delta_{\Pi(w)} \cap (\bigcup_{j=1}^{p} \mathcal{M}_j^*) \neq \emptyset$, *where* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *are the initial multisets of the system* $\Pi(w)$.

Proof. Let $w \in I_X$. Then $w \in L_X$ if and only if there exists an accepting computation of the system $\Pi(w)$. But this condition is equivalent to the following: in the initial configuration of $\Pi(w)$ there exists an object $a \in \Gamma$ in a membrane labelled with h such that in the dependency graph the node $(\texttt{yes}, environment)$ is reachable from (a, h).

Hence, $\theta_X(w) = 1$ if and only if $\Delta_{\Pi(w)} \cap \mathcal{M}_1^* \neq \emptyset$, or $\ldots$, or $\Delta_{\Pi(w)} \cap \mathcal{M}_p^* \neq \emptyset$. □

Theorem 3.3 $\mathbf{P} = \mathbf{PMC}^*_{\mathcal{AM}^0_{-d}}$.

Proof. We have $\mathbf{P} \subseteq \mathbf{PMC}^*_{\mathcal{AM}^0_{-d}}$ because the class $\mathbf{PMC}^*_{\mathcal{AM}^0_{-d}}$ is closed under polynomial time reduction. Next, we show that $\mathbf{PMC}^*_{\mathcal{AM}^0_{-d}} \subseteq \mathbf{P}$. For that, let $X \in \mathbf{PMC}^*_{\mathcal{AM}^0_{-d}}$. Let $\mathbf{\Pi} = (\Pi(w))_{w \in I_X}$ be a family of recognizer P systems without input membrane solving X, according to Definition 2.8.

We consider the following deterministic algorithm:

```
Input:  An instance w of X
  - Construct the system Π(w).
  - Construct dependency graph G_Π(w) associated with Π(w).
  - Construct the set Δ_Π(w) according to Proposition 3.2
    answer ← no; j ← 1
    while j ≤ p  ∧  answer = no do
       if Δ_Π(w) ∩ M*_j ≠ ∅ then
          answer ← yes
       j ← j + 1
    endwhile
```

On one hand, the answer of this algorithm is `yes` if and only if there exists a pair (a, h) belonging to $\Delta_{\Pi(w)}$ such that the symbol a appears in the membrane labelled with h in the initial configuration.

On the other hand, a pair (a, h) belongs to $\Delta_{\Pi(w)}$ if and only if there exists a path from (a, h) to $(\texttt{yes}, environment)$, that is, if and only if we can obtain an accepting computation of $\Pi(w)$. Hence, the algorithm above described solves the problem X.

The cost to determine whether or not $\Delta_{\Pi(w)} \cap M^*_j \neq \emptyset$ (for $1 \leq j \leq p$) is of the order $O(|\Gamma|^2 \cdot |H|^2)$.

Hence, the running time of this algorithm can be bounded by $O(|R| \cdot q) + O(p \cdot |\Gamma|^2 \cdot |H|^2)$, where R the set of rules of $\Pi(w)$, H is the set of labels for membranes of $\Pi(w)$, p is the number of (initial) membranes of $\Pi(w)$, and $q = \max\{length(r) : r \in R\}$. From Definition 2.8 we have that all involved parameters are polynomials in $|w|$. That is, the algorithm is polynomial in the size $|w|$ of the input. □

Bearing in mind that division rules for non–elementary membranes do not provide any novelty in the dependency graph, we have:

Theorem 3.4 $\mathbf{P} = \mathbf{PMC}^*_{\mathcal{AM}^0_{-d,+ne}}$.

4 Solving Problems by Recognizer P Systems with Active Membranes, without Polarizations and with Dissolution

In this section we show that the class of decision problems solvable in polynomial time in a semi–uniform way by families of recognizer P systems with active membranes, without polarization, with membrane dissolution rules and using division rules for elementary and non–elementary membranes, contains the standard complexity class **NP**.

For that, we describe a family of such recognizer membrane systems which solves the Subset Sum problem in linear time and in a semi–uniform way.

The Subset Sum problem is the following one: *Given a finite set A, a weight function, $w : A \to \mathbf{N}$, and a constant $k \in \mathbf{N}$, determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$.*

Proposition 4.1 *The Subset Sum problem belongs to the class* $\mathbf{PMC}^{*}_{\mathcal{AM}^{0}_{+d,+ne}}$.

Proof. We will use a tuple $u = (n, (w_1, \ldots, w_n), k)$ to represent an instance of the problem, where n stands for the size of $A = \{a_1, \ldots, a_n\}$, $w_i = w(a_i)$, and k is the constant given as input for the problem.

We propose here a solution to this problem based on a brute force algorithm implemented in the framework of P systems with active membranes, without polarizations, with dissolution, and using division for elementary and non–elementary membranes.

The idea of the design is better understood if we divide the solution to the problem into several stages:

- *Generation stage*: for every subset of A, a membrane is generated via membrane division.
- *Weight calculation stage*: in each membrane the weight of the associated subset is calculated. This stage will take place in parallel with the previous one.
- *Checking stage*: for each membrane it is checked whether or not the weight of its associated subset is exactly k. This stage cannot start before the previous ones are over.

- *Output stage*: when the previous stage has been completed in all membranes, the system sends out the answer to the environment.

For each instance $u = (n, (w_1, \ldots, w_n), k)$ of the Subset Sum problem we consider the P system with active membranes, without polarization, without input membrane $\Pi(u)$ defined as follows:

• Working alphabet:

$$\begin{aligned}\Gamma(u) &= \{d_0, \ldots, d_{2n+1}, a_1, \ldots, a_n, e_1, \ldots, e_n\} \\ &\cup \{b, s, c, \underline{c}, z_0, \ldots, z_{2n+k+5}, \texttt{yes}, \texttt{no}\}.\end{aligned}$$

• Initial membrane structure: $\mu = [\,[\,[\,[\ldots[\,[\,]_0\,]_1\ldots]_k\,]_{k+1}\,]_{k+2}\,]_{k+3}$.

• Initial Multisets: $w_0 = d_0$, $w_{k+2} = z_0$ and $w_i = \emptyset$, if $i \in \{1, \ldots, k, k+1, k+3\}$.

• The set of evolution rules, $R(u)$, consists of the following rules:

(a) $[d_{2i} \to a_{i+1} d_{2i+1}]_0$ for $i \in \{0, \ldots, n\}$,
$[d_{2i+1} \to d_{2i+2}]_0$ for $i \in \{0, \ldots, n-1\}$.

The goal of the counter d_i is to control the apparition of the object a_j only in the odd steps. The importance of these objects will be explained in the next set of rules.

(b) $[a_i]_0 \to [e_i]_0\,[b]_0$,
$[e_i \to s^{w_i}]_0$ for i $\in \{1, \ldots, \mathrm{n}\}$.

The object a_i triggers the rule for division of elementary membranes. After the division, in one membrane is placed the object e_i and in the other the object b. The object b remains inactive whereas the object e_i evolves in the next step to as many objects s as the weight w_i.

(c) $[\,[\,]_i\,[\,]_i\,]_{i+1} \to [\,[\,]_i\,]_{i+1}\,[\,[\,]_i\,]_{i+1}$ for $i \in \{1, \ldots, k\}$.

This is the set of rules for the division of non-elementary membranes. These three first set of rules produce a membrane structure with 2^n branches. On each of the leaves of the tree we have a membrane with as many objects s as the weight of a possible subset, S, of A.

(d) $[d_{2n+1}]_0 \to b$,
$[s]_i \to c$ for $i \in \{1, \ldots, k+1\}$.

When the generation stage has finished, the object d_{2n+1} dissolves the membrane with label 0. At this point, the elements s start to

dissolve membranes. If there are enough objects s, all the membranes with labels $1, \ldots, k+1$ are dissolved. Otherwise, the branch remains inactive.

(e) $[c \to \underline{c}]_{k+1}$.

It is a waiting step and the key of the computation. If in a branch the codified weight of the subset, w_S, is less than k, the membrane remains inactive. Otherwise all the membranes of the branch are dissolved until reaching the membrane with label $k+1$. If $w_S = k$ in this membrane, there are no objects s that dissolve it and the object $\underline{c}$ remains in the membrane. On the contrary, if $w_S > k$, the membrane is dissolved in the same step in which $\underline{c}$ is produced and $\underline{c}$ goes to the membrane with label $k+2$.

(f) $[z_i \to z_{i+1}]_{k+2}$ for $i \in \{0, \ldots, 2n+k+4\}$,
$[\underline{c}]_{k+1} \to \texttt{yes}$,
$[yes]_{k+2} \to \texttt{yes}$,
$[z_{2n+k+5}]_{k+2} \to \texttt{no}$.

If one of the subsets of A has weight k, then an object $\underline{c}$ appears in a membrane with label $k+1$. This object dissolves the membrane and sends an object `yes` to the membrane with label $k+2$. In this membrane we keep a counter z_i along the computation. If an object $\underline{c}$ has sent an object `yes` to this membrane, this object will dissolve the membrane in the next step preventing that the object z_{2n+k+5} remains in the membrane. Otherwise, if the object $\underline{c}$ is never produced, we get an object z_{2n+k+5} in the membrane with label $k+2$. In the following step this membrane is dissolved and an element `no` is sent to the membrane with label $k+3$.

(g) $[\texttt{no}]_{k+3} \to \texttt{no}\,[\,]_{k+3}$,
$[\texttt{yes}]_{k+3} \to \texttt{yes}\,[\,]_{k+3}$.

From above, we know that the membrane with label $k+3$ is reached by one and only one of the objects `yes` and `no`. This rules send that object to the environment in the last step of the computation.

□

Theorem 4.1 $\mathbf{NP} \cup \mathbf{co-NP} \subseteq \mathbf{PMC}^{*}_{\mathcal{AM}^{0}_{+d,+ne}}$.

Proof. It suffices to make the following observations: the Subset Sum problem is **NP**–complete, belonging to the class $\mathbf{PMC}^{*}_{\mathcal{AM}^{0}_{+d,+ne}}$, and this class is stable under polynomial-time reduction and closed under complement. □

Remark A. Alhazov et al. in [2] showed that $\mathbf{SAT} \in \mathbf{PMC}^{*}_{\mathcal{AM}^{0}_{+d,+ne}}$. Hence the result in Theorem 4.1 can also be deduced from this remark.

5 Conclusions

A conjecture known in the membrane computing area under the name of the P–conjecture affirms that $\mathbf{P} = \mathbf{PMC}_{\mathcal{AM}^{0}}$, where $\mathcal{AM}^{0}$ is the class of all recognizer P systems with active membranes and without polarization.

In this paper a partial negative answer to the P–conjecture is given when we use semi–uniform solutions and membrane division rules for elementary and non–elementary membranes.

We have used the concept of *dependency graph* that initially was defined to help to design strategies that allow to choose short computations of recognizer confluent membrane systems. In this paper we work with dependency graphs associated with a variant of recognizer P systems with active membranes. In this way we are able to characterize accepting computations of these systems through the reachability of a distinguished node of the graph from other nodes associated with the initial configuration.

We have also shown that it is possible to solve in polynomial time and in a uniform way through recognizer P systems with active membranes without polarizations and without dissolution *only* problems which are tractable in the standard sense. Moreover, if in this framework we consider membrane dissolution rules, then we can solve **NP**–complete problems in polynomial time, in a semi–uniform way and using division for elementary and non–elementary membranes.

Acknowledgement

The authors wish to acknowledge the support of the project TIC2002-04220-C03-01 of the Ministerio de Ciencia y Tecnología of Spain, cofinanced by FEDER funds.

References

[1] A. Alhazov, R. Freund, Gh. Păun: P systems with active membranes and two polarizations. In: Gh. Păun, A. Riscos, A. Romero, F. Sancho, (Eds.): *Proceedings of the Second Brainstorming Week on Membrane Computing* **Report RGNC 01/04** (2004), 20—35.

[2] A. Alhazov, L. Pan, Gh. Păun: Trading polarizations for labels in P systems with active membranes. *Acta Informaticae* **41** (2-3) (2004), 111–144.

[3] M.R. Garey, D.S. Johnson: Computers and Intractability. *A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York (1979).

[4] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A fast P system for finding a balanced 2-partition. *Soft Computing* **9** (9) (2005), 673–678.

[5] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving SAT with membrane creation. *Accepted paper for CiE* (2005). In: S. Barry Cooper, B. Lowe, L. Torenvliet (Eds.): *New Computational Paradigms, Amsterdam* , **Report ILLC X-2005-01**, University of Amsterdam, 82–91.

[6] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution for QSAT with Membrane Creation. In this volume.

[7] S.N. Krishna, R. Rama: A variant of P systems with active membranes: Solving NP–complete problems. *Romanian Journal of Information Science and Technology* 2 (4) (1999), 357–367.

[8] A. Obtułowicz: Deterministic P systems for solving SAT problem. *Romanian Journal of Information Science and Technology* **4** (1–2) (2001), 551–558.

[9] A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing* **20** (3) (2002), 295–305.

[10] Gh. Păun: P systems with active membranes: Attacking **NP**–complete problems. *Journal of Automata, Languages and Combinatorics*, 6 (1) (2001), 75–90.

[11] Gh. Păun: Computing with membranes: Attacking **NP**–complete problems. In: I. Antoniou, C. Calude, M.J. Dinneen (Eds.): *Unconventional Models of Computation.* Springer–Verlag (2000), 94–115.

[12] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108–143.

[13] Gh. Păun, M.J. Pérez-Jiménez: Recent computing models inspired from biology: DNA and membrane computing. *Theoria* **18** (46) (2003), 72–84.

[14] Gh. Păun: Further open problems in membrane computing. In: Gh. Păun, A. Riscos, A. Romero, F. Sancho (Eds.): Proceedings of the Second Brainstorming Week on Membrane Computing. *Report RGNC* **01/04**, University of Seville (2004), 354–365.

[15] M.J. Pérez-Jiménez: An approach to computational complexity in Membrane Computing. In: G. Mauri, Gh. Păun, M. J. Pérez-Jiménez, Gr. Rozenberg, A. Salomaa (Eds.): Membrane Computing, 5th International Workshop, WMC5, Revised Selected and Invited Papers. *Lecture Notes in Computer Science* **3365** (2005), 85–109.

[16] M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear time solution to the Knapsack problem using active membranes. In: C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. *Lecture Notes in Computer Science* **2933** (2004), 250–268.

[17] M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset-Sum problem by active membranes. *New Generation Computing*, in press.

[18] M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear solution for the Knapsack problem using active membranes. In C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. *Lecture Notes in Computer Science* **2933** (2004), 250–268.

[19] M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving the BIN PACKING problem by recognizer P systems with active membranes. In Gh. Păun, A. Riscos, A. Romero and F. Sancho (Eds.): *Proceedings of the Second Brainstorming Week on Membrane Computing Report*, RGNC 01/04, University of Seville (2004), 414–430.

[20] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: A polynomial complexity class in P systems using membrane division. In: E.

Csuhaj-Varjú, C. Kintala, D. Wotschke, G. Vaszil (Eds.): *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003* (2003), 284–294.

[21] M.J. Pérez-Jiménez, F.J. Romero–Campero: Attacking the Common Algorithmic Problem by recognizer P systems. In: M. Margenstern (Ed.): Machines, Computations and Universality, MCU'2004, Saint Petesburg, Russia, September 2004, Revised Selected Papers. *Lecture Notes in Computer Science* **3354** (2005), 304–315.

[22] P. Sosík: The computational power of cell division. In *Natural Computing* **2** (3) (2003), 287–298.

[23] C. Zandron, C. Ferreti, G. Mauri: Solving NP-complete problems using P systems with active membranes. In: I. Antoniou, C. Calude, M. J. Dinneen (Eds.): *Unconventional Models of Computation* UMC'2K, Springer-Verlag (2000), 289–301.

[24] ISI Web Page: `http://esi-topics.com/erf/october2003.html`

[25] The P Systems Web Page: `http://psystems.disco.unimib.it/`

A Linear Solution for QSAT with Membrane Creation

Miguel A. GUTIÉRREZ-NARANJO,
Mario J. PÉREZ-JIMÉNEZ,
Francisco J. ROMERO-CAMPERO

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mail: {magutier,marper,fran}@us.es

Abstract

The usefulness of P systems with membrane creation for solving **NP** problems has been previously proved (see [3, 4]), but, up to now, it was an open problem whether such P systems can solve **PSPACE** problems. In this paper we give an answer to this question by presenting a uniform family of P system with membrane creation which solve the QSAT-problem in linear time.

1 Introduction

The power of P systems as a tool for efficiently solving **NP** problems was been widely proved. Many examples have been proposed in the frame work of P systems with active membranes with two polarizations and three polarizations and in the framework of P systems with Membrane Creation.

The complexity class of **NP** problems deals with the *time* needed to solve a problem, i.e., **NP** is the class of problems which can be *solved* by a non-deterministic one-tape Turing machine program where the number of steps is polynomially bounded (see [1]). The key of such solutions is the creation of an exponential amount of workspace (membranes) in polynomial time.

When we consider the resources needed in a computation, we obviously have to consider the *time*, i.e. the number of steps of our device, but in

practice, we also need to consider the amount of memory or storage required by the computation. If we consider a Turing machine computation, the *space* is the number of distinct tape squares visited by the write-read head of the machine. Since the number of visited squares cannot be greater than the number of steps in the computation, we have that, if the number of steps is polynomially bounded, then the number of visited squares is also polynomially bounded. Therefore, any problem solvable in polynomial time is also solvable in polynomial space.

PSPACE (respectively, **NPSPACE**) is the class of decision problems that are solvable by a deterministic (respectively, non–deterministic) Turing machine using a polynomial amount of space. These complexity classes are closed under polynomial time reduction. Savitch's theorem says that each non–deterministic Turing machine using $f(n)$ space can be simulated by a deterministic Turing machine using only $f(n)^2$ space (for time complexity, such a simulation seems to require an exponential increase in time). Bearing in mind that a Turing machine running in $f(n) \geq n$ time can use at most $f(n)$ space we have $\mathbf{P} \subseteq \mathbf{PSPACE}$ and $\mathbf{NP} \subseteq \mathbf{NPSPACE}$. So, $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{NPSPACE} = \mathbf{PSPACE}$. It is unknown whether any of these containment are strict.

A decision problem in **PSPACE** such that every problem in **PSPACE** is polynomial time reducible to it, is called **PSPACE**–complete. If a **PSPACE**–complete problem belongs to **P** (respectively, **NP**), then $\mathbf{P} = \mathbf{PSPACE}$ (respectively, $\mathbf{NP} = \mathbf{PSPACE}$).

In this paper, we present a polynomial time solution to the QSAT problem, a well known **PSPACE**-complete problem (see L.J. Stockmeyer and A.R. Meyer in [15]) using a family of recognizer P systems with membrane creation. This result shows that all **PSPACE** problems can be solved in polynomial time by P systems with membrane creation.

The paper is organized as follows: first recognizer P systems are briefly described. In Section 3 P systems with membrane creation are recalled with a short discussion about their semantics. A linear–time solution to the QSAT problem is presented in the following section, with a short overview of the computation. Finally, some conclusions are given in the last section.

2 Recognizer P Systems

Recognizer P systems were introduced in [14] and are the natural framework to study and solve decision problems, since deciding whether an instance has an affirmative or negative answer is equivalent to deciding if a string belongs

or not to the language associated with the problem.

In the literature, recognizer P systems are associated in a natural way with P systems with *input*. The data related to an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation (`yes` or `no`) is sent to the environment. In this way, P systems with input and external output are devices which can be seen as black boxes, in which the user provides the data before the computation starts and the P system sends to the environment the output in the last step of the computation. Another important feature of P systems is the non-determinism. The design of a family of recognizer P system has to consider it, because all possibilities in the non-deterministic computations have to output the same answer. This can be summarized in the following definitions.

Definition 2.1 *A* P system with input *is a tuple* (Π, Σ, i_Π)*, where: (a)* Π *is a P system, with working alphabet* Γ*, with* p *membranes labelled by* $1, \ldots, p$*, and initial multisets* $w_1, \ldots, w_p$ *associated with them; (b)* Σ *is an (input) alphabet strictly contained in* Γ*; the initial multisets are over* $\Gamma - \Sigma$*; and (c)* i_Π *is the label of a distinguished (input) membrane.*

Let m be a multiset over Σ. The *initial configuration of* (Π, Σ, i_Π) *with input* m is $(\mu, w_1, \ldots, w_{i_\Pi} \cup m, \ldots, w_p)$.

Definition 2.2 *A* recognizer P system *is a P system with input,* (Π, Σ, i_Π)*, and with external output such that:*

1. *The working alphabet contains two distinguished elements* `yes`*,* `no`*.*
2. *All its computations halt.*
3. *If* $\mathcal{C}$ *is a computation of* Π*, then either the object* `yes` *or the object* `no` *(but not both) must have been released into the environment, and only in the last step of the computation. We say that* $\mathcal{C}$ *is an accepting computation (respectively, rejecting computation) if the object* `yes` *(respectively,* `no`*) appears in the environment associated to the corresponding halting configuration of* $\mathcal{C}$*.*

Definition 2.3 *Let* $\mathcal{F}$ *be a class of recognizer P systems. We say that a decision problem* $X = (I_X, \theta_X)$ *is solvable in polynomial time by a family* $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbb{N}}$*, of* $\mathcal{F}$*, and we denote this by* $X \in \mathbf{PMC}_{\mathcal{F}}$*, if the following is true:*

- *The family* $\mathbf{\Pi}$ *is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine constructing* $\Pi(n)$ *from* $n \in \mathbb{N}$ *in polynomial time.*

- *There exists a pair* (cod, s) *of polynomial-time computable functions over* I_X *such that:*

  - *for each instance* $u \in I_X$, $s(u)$ *is a natural number and* $cod(u)$ *is an input multiset of the system* $\Pi(s(u))$*;*
  - *the family* $\mathbf{\Pi}$ *is polynomially bounded with regard to* (X, cod, s)*, that is, there exists a polynomial function* p*, such that for each* $u \in I_X$ *every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is halting and, moreover, it performs at most* $p(|u|)$ *steps;*
  - *the family* $\mathbf{\Pi}$ *is sound with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if there exists an accepting computation of* $\Pi(s(u))$ *with input* $cod(u)$*, then* $\theta_X(u) = 1$*;*
  - *the family* $\mathbf{\Pi}$ *is complete with regard to* (X, cod, s)*, that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

In the above definition we have imposed to every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation of a system with the *same* input must always give the *same* answer.

It can be proved that $\mathbf{PMC}_{\mathcal{F}}$ is closed under polynomial–time reduction and complement, see [14]. In this paper we will deal with the class $\mathcal{MC}$ of recognizer P systems with membrane creation.

3 P Systems with Membrane Creation

In this section we recall the description of cellular devices (P systems) with membrane creation.

Basically, a P system[1]. consists of a hierarchical membrane structure where each membrane has associated a multiset of objects and a set of rules expressing how these objects can evolve. The *membrane structure* of a P system is a hierarchical arrangement of membranes embedded in a *skin* membrane, which separates the system from its *environment*. A membrane without any membrane inside is called *elementary*. Each membrane

[1] A layman-oriented introduction can be found in [9], a comprehensive monograph in [8], and the latest information about P systems is available at [16].

determines a *region* (the space enclosed between the membrane and the membranes immediately inside it), which can contain a multiset of *objects*. Associated with the regions there are *rules* that can transform and move those objects.

There are two ways of producing new membranes in living cells: *mitosis* (membrane division) and *autopoiesis* (membrane creation, see [6]). Both ways of generating new membranes have given rise to different variants of P systems: *P systems with active membranes*, where the new workspace is generated by membrane division, and *P systems with membrane creation*, where the new membranes are created from objects. Both models have been proved to be universal, but up to now there is no theoretical result proving that these models simulate each other in polynomial time. P systems with active membranes have been successfully used to design solutions to **NP**-complete problems, as SAT [14], Subset Sum [11], Knapsack [12], Bin Packing [13], and Partition [5], but as Gh. Păun pointed out in [10] *"membrane division was much more carefully investigated than membrane creation as a way to obtain tractable solutions to hard problems"*. The first results in this way have recently appeared, showing that **NP** problems can also be solved in this framework (see [3, 4]).

Recall that a *P system with membrane creation* is a construct of the form $\Pi = (O, H, \mu, w_1, \ldots, w_m, R)$, where:

1. $m \geq 1$ is the initial degree of the system; O is the alphabet of *objects* and H is a finite set of *labels* for membranes;

2. μ is a *membrane structure* consisting of m membranes labelled (not necessarily in a one-to-one manner) with elements of H and $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;

3. R is a finite set of *rules*, of the following forms:

   (a) $[a \rightarrow v]_h$ where $h \in H$, $a \in O$, and v is a string over O describing a multiset of objects. These are *object evolution rules* associated with membranes and depending only on the label of the membrane.

   (b) $a[\,]_h \rightarrow [b]_h$ where $h \in H$, $a, b \in O$. These are *send-in communication rules*. An object is introduced in the membrane possibly modified.

(c) $[a]_h \to [\,]_h\, b$ where $h \in H$, $a, b \in O$. These are *send-out communication rules*. An object is sent out of the membrane possibly modified.

(d) $[a]_h \to b$ where $h \in H$, $a, b \in O$. These are *dissolution rules*. In reaction with an object, a membrane is dissolved, while the object specified in the rule can be modified.

(e) $[a \to [v]_{h_2}]_{h_1}$ where $h_1, h_2 \in H$, $a \in O$, and v is a string over O describing a multiset of objects. These are *creation rules*. In reaction with an object, a new membrane is created. This new membrane is placed inside of the membrane of the object which triggers the rule and has associated an initial multiset and a label.

Rules are applied according to the following principles:

- Rules from (a) to (d) are used as usual in the framework of membrane computing, that is, in a maximally parallel way. In one step, each object in a membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can evolve by a rule of any form must do it (with the restrictions below indicated).

- Rules of type (e) are used also in a maximally parallel way. Each object a in a membrane labelled with h_1 produces a new membrane with label h_2 placing in it the multiset of objects described by the string v.

- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin membrane is never dissolved.

- All the elements which are not involved in any of the operations to be applied remain unchanged.

- The rules associated with the label h are used for all membranes with this label, irrespective of whether or not the membrane is an initial one or it was obtained by creation.

- Several rules can be applied to different objects in the same membrane simultaneously. The exception are the rules of type (d) since a membrane can be dissolved only once.

We denote by $\mathcal{MC}$ the class of recognizer P systems with membrane creation.

4 Solving QSAT in Linear Time

In this section we design a family of recognizer P systems with membrane creation (and using dissolution rules) which solves the QSAT problem (the quantified satisfiability problem).

Given a boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, with boolean variables $x_1, \ldots, x_n$, the sentence $\varphi^* = \exists x_1 \forall x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$ (where Q_n is $\exists$ if n is odd, and Q_n is $\forall$, otherwise) is said to be the (existential) *fully quantified* formula associated with $\varphi(x_1, \ldots, x_n)$.

We say that φ^* is satisfiable if there exists a truth assignment, σ, over $\{i : \ 1 \leq i \leq n \wedge i \text{ odd}\}$ such that each extension, σ^*, of σ over $\{1, \ldots, n\}$ verify $\sigma^*(\varphi(x_1, \ldots, x_n)) = 1$.

The QSAT problem is the following one: *Given a boolean formula $\varphi(x_1, \ldots, x_n)$ in conjunctive normal form, determine whether or not the (existential) fully quantified formula $\varphi^* = \exists x_1 \forall x_2 \ldots Q_n x_n \varphi(x_1, \ldots, x_n)$ is satisfiable.*

It is well known that QSAT is a **PSPACE**–complete problem [7].

Next, we provide a polynomial time solution of QSAT by a family of recognizer P systems with membrane creation and using dissolution rules, according to Definition 2.3. We will address the resolution via a brute force algorithm, in the framework of recognizer P systems with membrane creation, which consists in the following phases:

- Generation and Evaluation Stage: Using membrane creation we will generate all possible truth assignments associated with the formula and evaluate it on each one.
- Checking Stage: In each membrane we check whether or not the formula evaluates true on the truth assignment associated with it.
- Output Stage: The system sends out to the environment the right answer according to the previous stage.

Let us consider the pair function $\langle\ ,\ \rangle$ defined by $\langle n, m\rangle = ((n+m)(n+m+1)/2) + n$. This function is polynomial-time computable (it is primitive recursive and bijective from $\mathbb{N}^2$ onto $\mathbb{N}$). For any given boolean formula, $\varphi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$, in conjunctive normal form, with n variables and m clauses, we construct a P system $\Pi(\langle n, m\rangle)$ processing the (existential) fully quantified formula φ^* associated with φ. The family presented here is

$$\mathbf{\Pi} = \{(\Pi(\langle n, m\rangle), \Sigma(\langle n, m\rangle), i(\langle n, m\rangle)) \mid (n, m) \in \mathbb{N}^2\}.$$

For each element of the family, the input alphabet is

$$\Sigma(\langle n,m\rangle) = \{x_{i,j}, \overline{x}_{i,j} \mid 1 \leq i \leq m, 1 \leq j \leq n\}$$

the input membrane is $i(\langle n,m\rangle) = t$, and the P system

$$\Pi(\langle n,m\rangle) = (\Gamma(\langle n,m\rangle), \{a,t,f,1,\ldots,m\}, \mu, w_a, w_t, R(\langle n,m\rangle))$$

is define as follows:

• Working alphabet:

$$\begin{array}{rcl}
\Gamma(\langle n,m\rangle) & = & \Sigma(\langle n,m\rangle) \\
& \cup & \{z_{j,c} \mid j \in \{0,\ldots n\},\, c \in \{\wedge,\vee\}\,\} \\
& \cup & \{z_{j,c,l} \mid j \in \{0,\ldots,n-1\},\, c \in \{\wedge,\vee\}\, l \in \{t,f\}\} \\
& \cup & \{x_{i,j,l}, \overline{x}_{i,j,l} \mid j \in \{1,\ldots,n\},\, i \in \{1,\ldots,m\},\, l \in \{t,f\}\} \\
& \cup & \{x_{i,j} \mid j \in \{1,\ldots,n\},\, i \in \{1,\ldots,m\}\} \\
& \cup & \{r_i, r_{i,t}, r_{i,f} \mid i \in \{1,\ldots,m\}\} \\
& \cup & \{d_1,\ldots,d_m, q, t_0,\ldots,t_4, ans_0,\ldots,ans_5, \mathtt{yes}, \mathtt{no}\} \\
& \cup & \{yes_\vee, yes^*, no_\vee, \underline{no}_\vee, yes_\wedge, no_\wedge, no^*, yes_\wedge, \underline{yes}_\wedge, no_\vee, \underline{no}_\wedge\} \\
& \cup & \{YES, NO\}.
\end{array}$$

• Initial membrane structure: $\mu = [\,[\,]_{<t,\vee>}\,]_s$.

• Initial multiset: $w_s = \emptyset$, $w_{<t,\vee>} = \{z_{0,\wedge,t} z_{0,\wedge,f}\}$.

• Input membrane: $[\,]_{<t,\vee>}$.

• The set of evolution rules, $R(\langle n,m\rangle)$, consists of the following rules (recall that λ denotes the empty string and if c is $\wedge$ then $\overline{c}$ is $\vee$ and if c is $\vee$ then $\overline{c}$ is $\wedge$):

1. $\left.\begin{array}{l} [z_{j,c} \to z_{j,c,t}, z_{j,c,f}]_{<l,\overline{c}>} \\ [z_{j,c,l} \to [z_{j+1,\overline{c}}]_{<l,c>}]_{<l',\overline{c}>} \end{array}\right\}$ for $\begin{array}{l} l, l' \in \{t,f\}, \qquad c \in \{\vee,\wedge\}, \\ j \in \{0,\ldots,n-1\}. \end{array}$

The goal of these rules is to create one membrane for each truth assignment to the variables of the formula. Firstly, the object $z_{j,c}$ evolves to two objects, one for the assignment *true* (the object $z_{j,c,t}$), and a second one for the assignment *false* (the object $z_{j,c,f}$). In a second step these objects will create two membranes. The new membrane with t in its label represents the assignment $x_{j+1} = true$; on the other hand, the new membrane with f in its label represents the assignment $x_{j+1} = false$.

2. $\left.\begin{array}{l} [x_{i,j} \to x_{i,j,t} x_{i,j,f}]_{<l,c>} \\ [\overline{x}_{i,j} \to \overline{x}_{i,j,t} \overline{x}_{i,j,f}]_{<l,c>} \\ [r_i \to r_{i,t} r_{i,f}]_{<l,c>} \end{array}\right\}$ for $\begin{array}{lll} l \in \{t,f\} & & i \in \{1,\ldots,m\}, \\ c \in \{\vee,\wedge\} & & j \in \{1,\ldots,n\}. \end{array}$

These rules duplicate the objects representing the formula so it can be evaluated on the two possible assignments, $x_j = true$ $(x_{i,j,t}, \overline{x}_{i,j,t})$ and $x_j = false$ $(x_{i,j,f}, \overline{x}_{i,j,f})$. The objects r_i are also duplicated $(r_{i,t}, r_{i,f})$ in order to keep track of the clauses that evaluate true on the previous assignments to the variables.

3. $$\left.\begin{array}{l} x_{i,1,t}[\,]_{<t,c>} \to [r_i]_{<t,c>}, \\ \overline{x}_{i,1,t}[\,]_{<t,c>} \to [\lambda]_{<t,c>} \\ x_{i,1,f}[\,]_{<f,c>} \to [\lambda]_{<f,c>}, \\ \overline{x}_{i,1,f}[\,]_{<f,c>} \to [r_i]_{<f,c>} \end{array}\right\} \text{for } \begin{array}{l} i \in \{1,\ldots,m\}, \\ c \in \{\vee, \wedge\}. \end{array}$$

According to these rules the formula is evaluated in the two possible truth assignments for the variable that is being analyzed. The objects $x_{i,1,t}$ (resp. $\overline{x}_{i,1,f}$) get into the membrane with t in its label (resp. f) being transformed into the objects r_i representing that the clause number i evaluates true on the assignment $x_{j+1} = true$ (resp. $x_{j+1} = false$). On the other hand, the objects $\overline{x}_{i,1,t}$ (resp. $x_{i,1,t}$) get into the membrane with f in its label (resp. t) producing no objects. This represents that these objects do not make the clause true in the assignment $x_{j+1} = true$ (resp. $x_{j+1} = false$).

4. $$\left.\begin{array}{l} x_{i,j,l}[\,]_{<l,c>} \to [x_{i,j-1}]_{<l,c>} \\ \overline{x}_{i,j,t}[\,]_{<l,c>} \to [\overline{x}_{i,j-1}]_{<l,c>} \\ r_{i,t}[\,]_{<l,c>} \to [r_i]_{<l,c>} \end{array}\right\} \text{for } \begin{array}{lll} l \in \{t,f\}, & & i \in \{1,\ldots,m\}, \\ c \in \{\vee,\wedge\}, & & j \in \{2,\ldots,n\}. \end{array}$$

In order to analyze the next variable the second subscript of the objects $x_{i,j,l}$ and $\overline{x}_{i,j,l}$ are decreased when they are sent into the corresponding membrane labelled with l. Moreover, following the last rule, the objects $r_{i,l}$ get into the new membranes to keep track of the clauses that evaluate true on the previous truth assignments.

5. $$\left.\begin{array}{l} [z_{n,c} \to d_1 \ldots d_m q]_{<l,\overline{c}>} \end{array}\right\} \text{for } l \in \{t,f\} \text{ and } c \in \{\vee,\wedge\}.$$

At the end of the generation stage the object z_n will produce the objects $d_1, \ldots, d_m$ and yes_0, which will take part in the checking stage.

6. $$\left.\begin{array}{lll} [d_i \to [t_0]_i]_{<l,c>}, & & \\ r_{i,t}[\,]_i \to [r_i]_i, & [r_i]_i \to \lambda & \\ [t_s \to t_{s+1}]_i, & [t_2]_i \to t_3 & \end{array}\right\} \text{for } \begin{array}{l} i \in \{1,\ldots,m\}, \\ s \in \{0,1\}, \quad c \in \{\vee,\wedge\}. \end{array}$$

Following these rules each object d_i creates a new membrane with label i where the object t_0 is placed; this object will act as a counter. The object r_i gets into the membrane labelled with i and dissolves it preventing the counter, t_i, from reaching the object t_2. The fact that the object t_2 appears in a membrane with label i means that there is no object r_i, that is, the clause number i does not evaluate true on the truth assignment associated

with the membrane; therefore neither does the formula evaluate true on the associated truth assignment.

7. $$\left.\begin{array}{ll} [q \to [ans_0]_a]_{<l,c>} & \\ t_3[\,]_a \to [t_4]_a, & [t_4]_a \to \lambda \\ [ans_h \to ans_{h+1}]_a, & [ans_5]_a \to \texttt{yes} \\ [ans_5 \to \texttt{no}]_{<l,c>} & \end{array}\right\} \text{for } \begin{array}{l} l \in \{t,f\}, \quad c \in \{\vee,\wedge\}, \\ h = 0,\ldots,4. \end{array}$$

The object q creates a membrane with label a where the object ans_0 is placed. The object ans_h evolves to the object ans_{h+1}; at the same time the objects t_3 can get into the membrane labelled with a and dissolve it preventing the object `yes` from being sent out from this membrane.

8. $$\left.\begin{array}{ll} [yes]_{<l,c>} \to yes_{\overline{c}} & [no]_{<l,c>} \to no_{\overline{c}} \\ [yes_\vee]_{<l,\vee>} \to yes^*, & [no_\vee \to \underline{no}_\vee]_{<l,\vee>} \\ [yes^* \to yes_\wedge]_{<l,\wedge>}, & [\underline{no}_\vee]_{<l,\vee>} \to no_\wedge \\ [\underline{no}_\vee \to \lambda]_{<l,\wedge>}, & [yes_\vee \to \lambda]_{<l,\wedge>} \\ [no_\wedge]_{<l,\wedge>} \to no^*, & [yes_\wedge \to \underline{yes}_\wedge]_{<l,\wedge>} \\ [no^* \to no_\vee]_{<l,\vee>}, & [\underline{yes}_\wedge]_{<l,\wedge>} \to yes_\vee \\ [\underline{no}_\wedge \to \lambda]_{<l,\vee>}, & [\underline{yes}_\wedge \to \lambda]_{<l,\vee>} \\ [yes^*]_s \to \texttt{yes}\,[\,]_s, & [no_\wedge]_s \to \texttt{no}\,[\,]_s \end{array}\right\} \text{for } l \in \{t,f\}.$$

This set of rules controls the output stage. After the evaluation stage, from each working membrane we obtain an object *yes* or *no* depending on whether the truth assignment associated with this membrane satisfies or not the formula. On the contrary to the SAT problem, in QSAT it is not enough that one truth assignment satisfies the formula, but the final answer is YES if an appropriate combination of truth assignments according to the quantifiers $\exists$ and $\forall$ are founded.

4.1 An Overview of the Computation

First of all we define a polynomial encoding of the QSAT problem in the family $\mathbf{\Pi}$ constructed in the previous section. Given a boolean formula in conjunctive normal form, $\varphi = C_1 \wedge \cdots \wedge C_m$ such that $Var(\varphi) = \{x_1, \ldots, x_n\}$, and being φ^* the (existential) fully quantified formula associated with it, we define $s(\varphi^*) = \langle n, m \rangle$ (recall the bijection mentioned in the previous section) and $cod(\varphi^*) = \{x_{i,j} \,:\, x_j \in C_i\} \cup \{\overline{x}_{i,j} \,:\, \neg x_{i,j} \in C_i\}$.

Next we describe informally how the recognizer P system with membrane creation $\Pi(s(\varphi^*))$ with input $cod(\varphi^*)$ works.

In the initial configuration we have the input multiset $cod(\varphi)$ and the

objects $z_{0,\wedge,t}$ and $z_{0,\wedge,f}$ placed in the input membrane (membrane labelled with $< t, \vee >$). In the first step of the computation the object $z_{0,\wedge,t}$ creates a new membrane with label $< t, \vee >$ which represents the assignment $x_1 = true$ and the object $z_{0,\wedge,f}$ creates a new membrane with label $< f, \vee >$ which represents the assignment $x_1 = false$. The second component of the labels, i.e., $\wedge$ and $\vee$ will be used in the output stage.

In these two new membranes the object $z_{1,\vee}$ is placed. At the same time the input multiset representing the formula φ is duplicated following the two first rules in group 2. In the next step, according to the rules in group 3, the formula is evaluated on the two possible truth assignments for x_1. In the same step the rules in group 4 decrease the second subscript of the objects representing the formula ($x_{i,j,l}, \overline{x}_{i,j,l}$ with $j \geq 2$) in order to analyze the next variable. Moreover, at the same time, the object $z_{1,c}$ produces the object $z_{1,c,t}$ and $z_{1,c,f}$ ($c \in \{\wedge, \vee\}$) and the system is ready to analyze the next variable. And so the generation and evaluation stages goes until all the possible assignments to the variables are generated and the formula is evaluated on each one of them. Observe that it takes two steps to generate the possible assignments for a variable and to evaluate the formula on them; therefore the generation and evaluation stages take $2n$ steps.

The checking stage starts when the object $z_{n,c}$ produces the objects $d_1, \ldots, d_m$ and the object q. In the first step of the checking stage each object d_i, for $i = 1, \ldots, m$, creates a new membrane labelled with i where the object t_0 is placed, and the object q creates a new membrane with label a placing the object yes_0 in it.

The objects $r_{i,t}$, which indicate that the clause number i evaluates true on the truth assignment associated with the membrane, are sent into the membranes by the last rule in group 4 so the system keeps track of the clauses that are true. The objects $r_{i,t}$ get into the membrane with label i and dissolves it in the following two steps preventing the counter t_2 from dissolving the membrane and producing the object t_3 according to the last rule in group 6. If for some i there is no object r_i (this means that the clause i does not evaluate true on the associated assignment) the object t_2 will dissolve the membrane labelled with i producing the object t_3 that will get into the membrane with label a where the object ans_h evolves following the rules in 7. The object t_4 dissolves the membrane with label a preventing the production of the object ans_5. Therefore the checking stage takes 6 steps.

Finally the output stage takes place according to the rules in group 8. If some object ans_5 is present in any membrane with label $< l, c >$, ($l \in \{t, f\}$, $c \in \{\wedge, \vee\}$), this means that there exists at least one clauses not satisfied by

the truth assignment associated to the membrane, and by the last rule in group 7, we obtain *no* in this membrane. Otherwise, the object ans_5 will be inside the membrane with label a, it will dissolve the membrane, and send *yes* to the working membrane.

At this point, in each of the 2^n working membranes we have an object *yes* or *no* depending on if the associated truth assignment satisfies or not the formula φ. In the last steps we control the flow of the objects *yes* and *no* from the working membranes to the environment. Basically, the process is the following. If there are one object *yes* inside a membrane with $\vee$ in its label, this object dissolves the membrane and sends out another *yes*. If this does not happen, i.e., if two objects *no* are inside a membrane with label $\vee$, the membrane is dissolved and *no* is sent out. Analogously, if there are one object *no* inside a membrane with $\wedge$ in its label, this object dissolves the membrane and sends out another *no*. Otherwise, if two objects *yes* are inside a membrane with label $\vee$, the membrane is dissolved and *yes* is sent out.

Hence, the family $\mathbf{\Pi}$ of recognizer P systems with membrane creation using dissolution rules solves in polynomial time QSAT according to Definition 2.3. So, we have the following result.

Theorem 4.1 $QSAT \in \mathbf{PMC}_{\mathcal{MC}}$

Corrolary 4.1 $\mathbf{PSPACE} \subseteq \mathbf{PMC}_{\mathcal{MC}}$

Proof. It suffices to make the following remarks: the QSAT problem is **PSPACE**–complete, $QSAT \in \mathbf{PMC}_{\mathcal{MC}_{+d}}$, and the complexity class $\mathbf{PMC}_{\mathcal{MC}_{+d}}$ is closed under polynomial time reduction. □

5 Conclusions and Future Work

P systems are computational devices whose power has to be studied in a deeper extent. In the last months, several paper have explored this power, both in the framework of P systems with active membranes and P systems with membrane creation. These papers have shown that **NP**-complete problems are solvable (in polynomial time) by families of recognizer of P systems in such P system models, according to Definition 2.3. In this paper we have shown that **PSPACE**–complete problems can also be solved (in polynomial time) by families of recognizer P systems with membrane creation, in a uniform way.

Both models (active membranes and membrane creation) have been proved to be universal, but up to now there is no theoretical result proving that these models simulate each other in polynomial time. The specific techniques for designing solutions to concrete problems (generation, evaluation, checking, and output stages) are quite different, so the simulation of one model in the other one is not a trivial question. This seems an interesting open problem to be considered in the future.

Acknowledgement

This work is supported by Ministerio de Ciencia y Tecnología of Spain, by *Plan Nacional de I+D+I (2000–2003)* (TIC2002-04220-C03-01), cofinanced by FEDER funds. F.J. Romero-Campero also acknowledges a FPU fellowship from the same Ministerio.

References

[1] M.R. Garey, D.S. Johnson: *Computers and Intractability. A Guide to the theory of NP-Completeness.* W.H. Freeman and Company (1979).

[2] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: Towards a programming language in cellular computing. *Electronic Notes in Theoretical Computer Science. Proceedings of the 11th Workshop on Logic, Language, Information and Computation (WoLLIC'2004)*, Elsevier B.V. **123** (2005), 93–110.

[3] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution of Subset Sum Problem by using Membrane Creation. In: J. Mira, J.R. Alvarez (Eds.): Mechanisms, symbols and models underlying using cognition, First International Work-Conference on the interplay between Natural and Artificial Computation, IWINAC 2005. *Lecture Notes in Computer Science* **3561** (2005), 258–267.

[4] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving SAT with Membrane Creation. In: S. Barry Cooper, B. Lowe, L. Torenvliet (Eds.): *Computability in Europe 2005, CiE 2005: New Computational Paradigms* , Report ILLC X-2005-01, University of Amsterdam, 82–91.

[5] M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A fast P system for finding a balanced 2-partition. *Soft Computing* **9** (9) (2005), 673–678.

[6] P.L. Luisi: The chemical implementation of autopoiesis. In: G.R. Fleishaker et al. (Eds.): *Self-Production of Supramolecular Structures*. Kluwer, Dordrecht (1994).

[7] C.H. Papadimitriou: *Computational Complexity*. Addison-Wesley (1994).

[8] Gh. Păun: *Membrane Computing: An Introduction*, Springer-Verlag, Berlin (2002).

[9] Gh. Păun, M.J. Pérez-Jiménez: Recent computing models inspired from biology: DNA and membrane computing. *Theoria* **18** (46) (2003), 72–84.

[10] Gh. Păun: Further open problems in membrane computing. In: (Gh. Păun, A. Riscos-núñez, A. Romero-Jiménez, F. Sancho-Caparrini (Eds.): *Proceedings of the Second Brainstorming Week on Membrane Computing*. Report RGNC 01/04, University of Seville (2004), 354–365.

[11] M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset-Sum problem by active membranes. *New Generation Computing* **23** (4) (2005), 367–384.

[12] M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear solution for the Knapsack problem using active membranes. In: C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. *Lecture Notes in Computer Science* **2933** (2004), 250–268.

[13] M.J. Pérez-Jiménez, F.J. Romero-Campero: Solving the BIN PACKING problem by recognizer P systems with active membranes. In: Gh. Păun, A. Riscos-núñez, A. Romero-Jiménez, F. Sancho-Caparrini (Eds.): *Proceedings of the Second Brainstorming Week on Membrane Computing*. Report RGNC 01/04, University of Seville (2004), 414–430.

[14] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: A polynomial complexity class in P systems using membrane division. In: E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszil (Eds.): *Proceedings*

of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003, 284–294.

[15] L.J. Stockmeyer, A.R. Meyer: Word problems requiring exponential time. Proc. 5th ACM Symp. on the Theory of Computing (1973), 1–9.

[16] P Systems Web Page `http://psystems.disco.unimib.it/`

Boolean Circuits and a DNA Algorithm in Membrane Computing

Mihai IONESCU[*], **Tseren-Onolt ISHDORJ**[†]

Research Group on Mathematical Linguistics
Rovira i Virgili University
Pl. Imperial Tàrraco 1
43005 Tarragona, Spain
E-mail: mi@urv.net,
tserenonolt.ishdorj@estudiants.urv.es

Abstract

In the present paper we propose a way to simulate Boolean gates and circuits in the framework of P systems with active membranes using inhibiting/de-inhibiting rules. This new approach on the simulation of Boolean gates has the advantage of a self-embedded synchronization, an extra system to solve this problem not being needed. Moreover, the number of membranes and objects we use for the simulation of boolean gates is only two. **NP**-complete problems, particularly `CIRCUIT-SAT`, are also considered here. In addition, we simulate a DNA-like'way of (experimentally) solving `SAT` problem using the tools given by polarization, merging and separation in P systems.

1 Introduction

P systems are a class of distributed parallel computing devices of a biochemical type, which can be seen as a general computing architecture where various types of objects can be processed by various operations.

[*]The work of first author was supported by the fellowship "Formación de Profesorado Universitario" from the Spanish Ministry of Education, Culture and Sport.

[†]The second author was supported by the grant 2002CAJAL-BURV4 from Rovira i Virgili University, Spain.

In membrane computing, P systems with active membranes have a special place, because they provide biologically inspired tools to solve computationally hard problems. In [6] the computational power of a class of P systems using catalytic and non-cooperative inhibiting/de-inhibiting rules was introduced and explored, and in [7] such a controlling mechanism was investigated in the framework of P systems with active membranes.

Boolean circuits are well-known classical computing devices, which incorporate features of parallelism. Various possibilities to simulate Boolean circuits by P systems with promoters/inhibitors, mobile catalysts, and weak priorities for rules were considered in [8].

In this paper, we propose a model to simulate Boolean circuits by inhibiting/de-inhibiting P systems with active membranes (AID P Systems). The idea behind the simulation of such a circuit is to construct a global AID P system for the whole circuit having distributed sub-AID P systems for each gate. The sub-AID P systems work in a parallel manner producing a unique output as the result of the computation of the whole system. One can see a correspondence between the concept of *inhibition* (which means *blocking* the execution of a rule in a membrane) and the term *switch-off* frequently used in the theory of circuits, and conversely, between *de-inhibition* and *switching on* some parts of the circuits.

Using this model to simulate Boolean circuits, we do not need an extra system to coordinate the entrance of the two inputs in an AND or an OR gate as presented in [8]. Here, the inputs wait for each other and produce the right result when the circuit is simulated. We say that the system has a self-embedded synchronization. As a consequence of the ability of AID P systems to simulate specific Boolean circuits, we also consider the Boolean `CIRCUIT-SAT` problem (which is **NP**-complete).

The second proposal of this paper is the construction of a DNA-like'system for solving `SAT` problem, following the original idea of Lipton presented in [13], this time, using the properties of merging and separation operations from membrane computing. The motivation behind this proposal is to create a model as close as possible to the experimental results on solving the problem mentioned above by using DNA strands (see [4], [23]). We only say here that the main idea in such a simulation is to consider the test tubes used in the experiments as being membranes, and, of course, the DNA-strands contained in such a test tube as being the objects inside membranes. For a detailed presentation of this concept we refer the reader to Section 6.

In the following section we will recall some of the notions regarding P systems with active membranes, boolean gates, circuits, and the DNA way of solving SAT we will use in the next sections.

2 Preliminaries

We assume the reader to be familiar with the fundamentals of formal language theory and complexity theory (e.g., from [17, 24, 25]), as well as with the basics of membrane computing (e.g., from [20]).

2.1 P Systems with Active Membranes

Informally speaking, in P systems with polarizations and active membranes the following types of rules are used:

(a) multiset rewriting rules,

(b) rules for introducing objects into membranes,

(c) rules for sending objects out of membranes,

(d) rules for dissolving membranes,

(e) rules for dividing elementary membranes, and

(h) rules for separating membranes, see [2, 19, 22, 15, 16].

The rules of type (a_0) are applied in a parallel way (all objects which can evolve by such rules have to evolve), while the rules of types (b), (c), (d), (e), (g), (h) are used sequentially, in the sense that one membrane can be used by at most one rule of these types at a time. In total, the rules are used in the non-deterministic maximally parallel manner: all objects and all membranes which can evolve, should evolve. Only halting computations give a result, in the form of the number (or the vector) of objects expelled into the environment during the computation.

In this paper we will make use only of some of the mentioned active membranes rules, so we define P systems with active membranes using only such rules.

A P system *with active membranes* (and electrical charges) is a construct

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

where:

- $m \geq 1$ (the initial degree of the system);

- O is the alphabet of *objects*;

- H is a finite set of *labels* for membranes;

- μ is a *membrane structure*, consisting of m membranes, labeled (not necessarily in a one-to-one manner) with elements of H;

- $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;

- R is a finite set of *developmental rules*, of the following forms:

  (a) $[\ a \to v]_h^e$,
  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, in the sense that the membranes are neither taking part in the application of these rules nor are they modified by them);

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

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

  (g) $[\ \ ]_h^{e_1}\ [\ \ ]_h^{e_2} \to [\ \ ]_h^{e_3}$,
  for $h \in H$, $e_1, e_2, e_3 \in \{+, -, 0\}$

  (merging rules for elementary membranes; in reaction of two membranes, they are merged into a single membrane; the objects of the former membranes are put together in the new membrane);

(h) $[\ ]_h^{e_1} \to [\ K]_h^{e_2} [\ \neg K]_h^{e_3}$,
for $h \in H$, $e_1, e_2, e_3 \in \{+, -, 0\}$, $K \subseteq O$

(separation rules for elementary membranes; the contents of membrane h is split into two membranes, the first one containing all objects from K and the second one containing all objects which are not in K).

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

(h') $[\ ]_{h_1}^{e_1} \to [\ K]_{h_2}^{e_2} [\ \neg K]_{h_3}^{e_3}$,
for $h_1, h_2, h_3 \in H$, $e_1, e_2, e_3 \in \{+, -, 0\}$.

The change of labels can also be considered for other types of rules.

P systems with active membranes without electrical charges were also considered and investigated (see [2, 3, 1, 12]). Let us consider now some rules of types without polarizations. They are of the following forms ("no electrical charges" means "neutral polarization"; as above, O is the alphabet of objects and H is the set of labels of membranes):

(a_0) $[\ a \to v]_h$, where $a \in O, v \in O^*$, and $h \in H$,

(b_0) $a[\ ]_h \to [\ b]_h$, where $a, b \in O$ and $h \in H$,

(c_0) $[\ a]_h \to [\ ]_h b$, where $a, b \in O$ and $h \in H$,

(g_0) $[\ ]_h [\ ]_h \to [\ ]_h$, where $h \in H$,

(h_0) $[\ ]_h \to [\ K]_h [\ \neg K]_h$, where $K \subseteq O$ and $h \in H$,

(i_0) $[\ [\ O]_h]_h \to [\ ]_h O$, where $h \in H$.

We recommend the reader unfamiliar with these rules to consult the references mentioned above for a better understanding of their functionality.

The subscript 0 indicates the fact that we do not use polarization for membranes. When the rules of a given type (α_0) are able to change the

label(s) of the involved membranes, we denote that type of rules by (α'_0). For example, the primed versions of merging and separation rules are of the following forms:

(g'_0) $[\ ]_{h_1}[\ ]_{h_2} \to [\ ]_{h_3}$, for $h_1, h_2, h_3 \in H$.

(h'_0) $[\ ]_{h_1} \to [\ K]_{h_2}[\ \neg K]_{h_3}$, for $h_1, h_2, h_3 \in H$.

To understand the difference of uniform construction and semi-uniform construction of P systems, we recall some notions about solving decidability problems in the membrane computing framework.

Given a decision question X, we say that it can be solved in polynomial (linear) time by recognizing P systems in a uniform way, if, informally speaking, we can construct in polynomial time a family of recognizing P systems Π_n, $n \in \mathbb{N}$, associated with the sizes n of instances $X(n)$ of the problem, such that the system Π_n, starting from the code of some Π_n, will always stop in a polynomial (linear, respectively) number of steps, sending out the object `yes` if the instance $X(n)$ has a positive answer and the object `no` if the instance $X(n)$ has a negative answer.

In [19], the complexity classes related to P systems are defined in the semi-uniform way: P systems are constructed starting not from the size n, but from an instance $X(n)$. For a clearer description of the difference between uniform and semi-uniform constructions, the reader is referred to [22].

2.2 Inhibiting/De-inhibiting (AID) P Systems with Active Membranes

The basic idea of the AID P systems model is that, when a rule (acting on the membranes or on the objects) is inhibited, then it can not be applied until another rule de-inhibits it. The application of a rule can inhibit other rules (and in particular may inhibit itself).

A P system *with active membranes and inhibiting/de-inhibiting mechanism*, in short AID P system, without electrical charges and without using catalysts, is a construct

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

where:

- $m \geq 1$ is the initial degree of the system;
- O is the alphabet of *objects*;
- H is a finite set of *labels* for membranes;
- I is a finite set of *labels* for rules;
- μ is a *membrane structure*, consisting of m membranes, labeled with elements of H;
- $w_1, \ldots, w_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ;
- R is a finite set of *developmental rules*. Here are some examples:

(b_0) $r : a[\ ]_h \rightarrow [\ b]_h \langle S \rangle$, for $r \in I, h \in H, a, b \in O, S \subseteq I$(communication rules; an object is introduced in the membrane during this process);

(c_0) $r : [\ a\ ]_h \rightarrow [\ ]_h b \langle S \rangle$, for $r \in I, h \in H, a, b \in O, S \subseteq I$(communication rules; an object is sent out of the membrane during this process).

The rules in R are written as $r_j : \neg r \langle S \rangle$ or as $r_j : r \langle S \rangle$, where $r_j \in I$ and r is a rule of type $(a_0) - (l_0)$ (replicative-distribution rules are: (k_0) $r : a[\ ]_{h_1}[\ ]_{h_2} \rightarrow [\ u]_{h_1}[\ v]_{h_2}$, for $h_1, h_2 \in H, a \in O, u, v \in O^*$ –for sibling membranes; (l_0) $r : [\ a[\ ]_{h_1}]_{h_2} \rightarrow [\ [\ u]_{h_1}]_{h_2} v$, for $h_1, h_2 \in H, a \in O, u, v \in O^*$ –for nested membranes) from [1, 2, 6]; S is a string that represents a subset of I.

The AID P systems works like general P systems with active membranes. The only difference consists in the fact, that, in each step, only the non-inhibited rules can be used. When a rule $r_j : r \langle S \rangle$ is applied, then the rules whose labels are specified in S are inhibited (if they were de-inhibited) or de-inhibited (if they were inhibited). Now, starting from an initial configuration, the system evolves according to the rules and objects present in the membranes, in a non-deterministic maximally parallel manner, and according to a universal clock. The system will make a successful computation if and only if it halts, meaning there is no applicable rule to the objects present in the halting configuration.

The result of a successful computation is the number of objects present in the output membrane (or in the environment) in a halting configuration of Π. If the computation never halts, then we will have no output.

2.3 Boolean Functions and Circuits

An n-ary *Boolean function* is a function

$$f : \{true, false\}^n \mapsto \{true, false\};$$

$\neg$ (negation) is a unary Boolean function (the other unary functions are: constant functions and identity function). We say that the Boolean expression φ with variables $x_1, \ldots, x_n$ *expresses* the n-ary Boolean function f if, for any n-tuple of truth values $t = (t_1, \cdots, t_n)$, $f(t)$ is *true* if $T \models \varphi$, and $f(t)$ is false if $T \not\models \varphi$, where $T(x) = t_i$ for $i = 1, \ldots, n$.

There are three primary boolean functions that are widely used:

(1) the NOT function - this is a just a negation; the output is the opposite of the input. The NOT function takes only one input, so it is called a unary function or operator. The output is true when the input is false, and vice-versa.

(2) the AND function - the output of an AND function is true only if all its inputs are true.

(3) the OR function - the output of an OR function is true if at least one of its inputs is true.

Both AND and OR can have any number of inputs, with a minimum of two.

Any n-ary Boolean function f can be expressed as a Boolean expression φ_f involving variables $x_1, \ldots, x_n$.

There is a potentially more economical way than these expressions for representing Boolean functions –namely *Boolean circuits*. A Boolean circuit is a graph $C = (V, E)$, where the nodes in $V = \{1, \ldots, n\}$ are called the *gates* of C. Graph C has a rather special structure. First, there are no cycles in the graph, so we can assume that all edges are of the form (i, j), where $i < j$. All nodes in the graph have in-degree (number of incoming edges) equal to 0, 1, or 2. Moreover, each gate $i \in V$ has a *sort* $s(i)$ associated with it, where

$$s(i) \in \{true, false, \vee, \wedge, \neg\} \cup \{x_1, x_2, \ldots\}.$$

If $s(i) \in \{true, false\} \cup \{x_1, x_2, \ldots\}$, then the in degree of i is 0, that is, i must not have any incoming edges. Gates with no incoming edges are called the *inputs* of C. If $s(i) = \neg$, then i has in-degree one. If $s(i) \in \{\vee, \wedge\}$, then the in-degree of i must be two. Finally, node n (the largest numbered gate

in the circuit, which necessarily has no outgoing edges) is called the *output gate* of the circuit.

This concludes our definition of the *syntax* of circuits. The *semantics* of circuits specifies a truth value for each appropriate truth assignment. We let $X(C)$ be the set of all Boolean variables that appear in the circuit C (that is, $X(C) = \{x \in X : s(i) = x \text{ for some gate } i \text{ of } C\}$). We say that a truth assignment T is appropriate for C if it is defined for all variables in $X(C)$. Given such a T, *the truth value of gate* $i \in V$, $T(i)$, is defined, by induction on i, as follows:

If $s(i) = true$ then $T(i) = true$, and similarly if $s(i) = false$. If $s(i) \in X$, then $T(i) = T(s(i))$. If now $s(i) = \neg$, there is a unique gate $j < i$ such that $(j, i) \in E$. By induction, we know $T(j)$, and then $T(i)$ is $true$ if $T(j) = false$, and vice-versa.

If $s(i) = \vee$, then there are two edges (j, i) and (j', i) entering i. $T(i)$ is then $true$ if only if at least one of $T(j)$, $T(j')$ is $true$. If $s(i) = \wedge$, then $T(i)$ is $true$ if only if both $T(j)$ and $T(j')$ are $true$, where (j, i) and (j', i) are the incoming edges. Finally, *the value of the circuit*, $T(C)$, is $T(n)$, where n is the output gate.

2.4 Brief Description of Solving SAT in DNA Computing

Lipton's DNA-based solution of the satisfiability problem [13] uses some of the basic operations in DNA Computing:

– *merge* (given test tubes N_1 and N_2, we consider their union, understood as a multiset),

– *separate* (given a test tube N and a word w over the alphabet A, C, T, G, produce two test tubes $+(N, w)$ and $-(N, w)$, where $+(N, w)$ consists of all strands in N which contain w as a (consecutive substring), while $-(N, w)$ is its negation), and

– *detect* (given a tube N, return $true$ if N contains at least one DNA strand, otherwise return $false$).

We begin with a graphical description of truth assignments. Assume that we are dealing with a propositional formula containing k variables. Consider the directed graph depicted in Fig. 1:

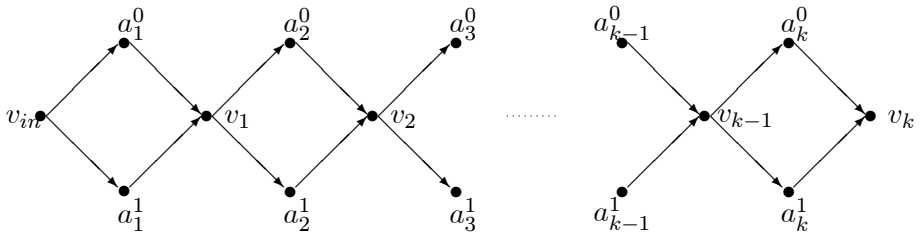


Figure 1: A graph associated with a truth assignment.

There are 2^k paths from v_{in} to v_{out}, none of the paths being Hamiltonian. One can observe that in each of the v_i nodes ($i \neq out$) there are two independent choices (0 or 1). The construction of the graph prevents the (unwanted) possibility of choosing for the same variable both 0 and 1 values. Moreover, the paths and the truth assignments for the variables $x_1, x_2, \ldots, x_k$ have a natural one-to-one correspondence.

Each vertex of the graph is encoded by a random oligonucleotide of length 20 and an arc between two vertices will be the Watson-Crick complementation of the last half and the first half of the start and end nodes, respectively. More precisely, consider the encodings s_i and s_j of two vertices such that there is an edge $e_{i,j}$ from the former to the latter. If $s_i = s_i's_i''$, where s_i' and s_i'' have equal length, and similarly, $s_j = s_j's_j''$, then the edge $e_{i,j}$ is encoded by the Watson-Crick complement of $s_i''s_j'$.

Now, having encoded all the possible truth assignments with the help of the operations mentioned above and strictly depending on the clauses given in the `SAT` problem, an algorithm (based on separation, merging, and, only in the end, detection) will select the right solution(s) of the given problem. An example is presented in Section 6 where we compare the two (DNA and P-based) ways of solving a particular, simple `SAT` problem.

3 Simulating Logical Gates

In this section we present AID P systems which simulate logical gates. We will consider that the input for a gate is given in the inner membrane, while the output will be computed and sent out to the outer region.

3.1 Simulation of AND Gate

Lemma 3.1 *Boolean AND gate can be simulated by AID P systems with rules of types* (b'_0) *and* (c'_0)*, using two membranes and two objects (only the input), in at most four steps.*

Proof. We construct the AID P system

$$\begin{aligned}
\Pi_{AND} &= (O, H, I, \mu, w_0, w_s, R), \text{with} \\
O &= \{0, 1\}, \\
\mu &= [\ [\ \ ]_0]_s, \\
w_0 &= w_s = \lambda, \\
H &= \{0, 1, s\}, \\
I &= \{r_i \mid 0 \leq i \leq 9\},
\end{aligned}$$

and the set R consisting of the following rules:

$r_1 : [\ 0]_0 \to [\ \ ]_1 0,$

$r_2 : [\ 0]_1 \to [\ \ ]_0 \lambda \langle r_2 r_8 \rangle,$

$r_3 : [\ 1]_0 \to [\ \ ]_1 1 \langle r_2 r_4 r_5 r_6 \rangle,$

$r_4 : [\ 1]_1 \to [\ \ ]_0 \lambda \langle r_2 r_8 \rangle,$

$r_5 : \neg[\ 0]_1 \to [\ \ ]_1 0 \langle r_5 r_7 \rangle,$

$r_6 : \neg[\ 1]_1 \to [\ \ ]_1 \lambda \langle r_4 r_6 r_9 \rangle,$

$r_7 : \neg 1[\ \ ]_1 \to [\ \lambda]_0 \langle r_4 r_6 r_7 r_8 \rangle,$

$r_8 : \neg[\ 0]_s \to [\ \ ]_s 0 \langle r_2 r_8 \rangle,$

$r_9 : \neg[\ 1]_s \to [\ \ ]_s 1 \langle r_2 r_5 r_9 \rangle.$

Initially, we place the input values x_1 and x_2 in the membrane with label 0 from the membrane structure. Depending on the value of the initial variables x_1 and x_2, the rules we apply for each of the four cases are: r_1, r_2, r_8 for 00, r_1, r_4, r_8 for 01, r_3, r_5, r_7, r_8 for 10, and r_3, r_6, r_9 for 11.

More precisely, if two symbols 1 are in membrane 0, in the first step, rule r_3 is applied, a symbol 1 is expelled and the membrane label is changed to 1. At the same time, according to the inhibition/de-inhibition concept, rules r_2 and r_4 are inhibited, while rules r_5 and r_6 are de-inhibited and ready to be used. In the second step we notice that only rule r_6 can be applied, thus, object 1, placed inside membrane labeled 1 is transformed, on its way out, into λ. One may notice that rule r_6, after having been applied, restores the original status of rules r_4 and itself, and also de-inhibits rule r_9. In the third step, rule r_9 performs and the right answer 1 is sent out the skin membrane, while rules r_2, r_5, and r_9 come back to their original status.

In other words, after these three steps, our system has sent out of the skin membrane the right answer (given the input 11) and comes back to its initial configuration, thus being ready for a new input.

In the case when the input is 01 or 10, we can start by using r_1 or r_3. Let us examine the second case. Rule r_3 sends 1 out of membrane 0 and changes its label to 1. At the same time, rules r_2 and r_4 are inhibited, while rules r_5 and r_6 are de-inhibited. The only rule we can use in the second step is r_5 which expels 0 out of membrane 1, inhibits itself and de-inhibits rule r_7. In this moment we have the following configuration of our system: $[\,[\,\,]_1 01]_s$. We now apply rule r_7 which transforms object 1 to λ on its way in the inner membrane and changes its label from 1 to 0. Rule r_7 de-inhibits the inhibited rule r_4, inhibits r_6 and itself, and de-inhibits rule r_8. The fourth step is the one in which the right answer 0 is sent out of the skin membrane, while the system gets back to its initial configuration.

Thus, our system gives the right answer, in four steps, when we have input 01. In the other two cases (when we have the input 01 and start by using first the rule r_1, or the input is 00) our system performs the rules mentioned above, the details being left to the reader. □

3.2 Simulation of OR Gate

Lemma 3.2 *A Boolean OR gate with fan-in at most 2 can be simulated by AID P systems with rules of types* (b'_0) *and* (c'_0)*, using two membranes and two objects (only the input), at most four steps.*

Proof. We construct the AID P system

$$\begin{aligned}
\Pi_{OR} &= (O, H, I, \mu, w_0, w_s, R), \text{with} \\
O &= \{0, 1\}, \\
\mu &= [\,[\;\;]_0]_s, \\
w_0 &= w_s = \lambda, \\
H &= \{0, 1, s\}, \\
I &= \{r_i \mid 0 \le i \le 9\},
\end{aligned}$$

and the following set of R of rules:

$r_1 : [\ 1]_0 \to [\;\;]_1 1,$

$r_2 : [\ 1]_1 \to [\;\;]_0 \lambda \langle r_2 r_8 \rangle,$

$r_3 : [\ 0]_0 \to [\;\;]_1 0 \langle r_2 r_4 r_5 r_6 \rangle,$

$r_4 : [\ 0]_1 \to [\;\;]_0 \lambda \langle r_2 r_8 \rangle,$

$r_5 : \neg[\ 1]_1 \to [\;\;]_1 1 \langle r_5 r_7 \rangle,$

$r_6 : \neg[\ 0]_1 \to [\;\;]_1 \lambda \langle r_4 r_6 r_9 \rangle,$

$r_7 : \neg 0[\;\;]_1 \to [\ \lambda]_0 \langle r_4 r_6 r_7 r_8 \rangle,$

$r_8 : \neg[\ 1]_s \to [\;\;]_s 1 \langle r_2 r_8 \rangle,$

$r_9 : \neg[\ 0]_s \to [\;\;]_s 0 \langle r_2 r_5 r_9 \rangle.$

As in the case of the AND gate, we place initial values x_1 and x_2 in the membrane labeled 0 from the membrane structure. The succession of rules we apply for each case is (as expected due to the duality of the system) the following: r_3, r_6, r_9 for 00, r_3, r_5, r_7, r_8 for 01, r_1, r_4, r_8 for 10, and r_1, r_2, r_8 for 11.

We only give here the details of the case when x_1 and x_2 are both 1. Our system has the following initial configuration: $[\,[\,11]_0]_s$. As mentioned above, the only rule we can apply is r_1, and our system evolves to the following configuration: $[\,[\,1]_1 1]_s$. The next rule we can apply is r_2 through which the object in membrane 1 is transformed into λ and the membrane label changes to 0, the system evolving to $[\,[\;\;]_0 1]_s$. After having applied rule r_2, rule r_8 is de-inhibited while rule r_2 is inhibited. We now can apply r_8, which sends out of the skin membrane the answer 1 and restores the initial configuration of the system inhibiting rule r_8 and de-inhibiting rule r_2.

We have shown how our systems expels, in three steps, the right answer, given the input 11.

The details of the behavior of the system in the other three cases are left to the reader. □

3.3 Simulation of NOT Gate

Lemma 3.3 *A Boolean (unary) NOT gate can be simulated by AID P systems with rules of type* (b_0) *in one step.*

Proof. We construct the AID P system

$$\begin{aligned}
\Pi_{NOT} &= (O, H, S, \mu, w_s, R), \text{with} \\
O &= \{0, 1\}, \\
\mu &= [\;\;]_s, \\
w_s &= x_1 x_2, \\
H &= \{s\}, \\
S &= \{r_0, r_1\}, \\
R &= \{r_0 : [\,0]_s \to [\;\;]_s 1, r_1 : [\,1]_s \to [\;\;]_s 0\}.
\end{aligned}$$

The correct simulation of the NOT gate is obvious. □

4 Simulating Circuits

We give now an example of how to construct a global AID P system which simulates a Boolean circuit, designed for evaluating a Boolean function, using the distributed sub-AID P systems in it, namely including Π_{AND}, Π_{OR} and Π_{NOT} constructed in the previous section.

4.1 An Example

We take into consideration the same example used in [8], namely we consider the function $f : \{0,1\}^4 \to \{0,1\}$ given by the formula

$$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2) \vee \neg(x_3 \wedge x_4).$$

The corresponding circuit is depicted in Fig. 2, its assigned membrane structure in Fig 3:

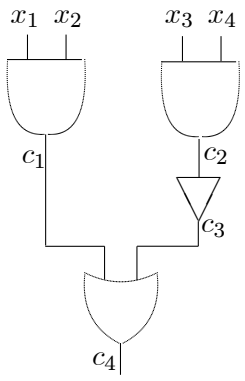


Figure 2: Boolean Circuit.

As shown in Fig. 3, the circuit has a tree as its underlying graph, with the leaves as input gates and the root as output gate.

We simulate this circuit with the P system

$$\Pi_C = (\Pi^{(1)}_{AND}, \Pi^{(2)}_{AND}, \Pi^{(3)}_{NOT}, \Pi^{(4)}_{OR})$$

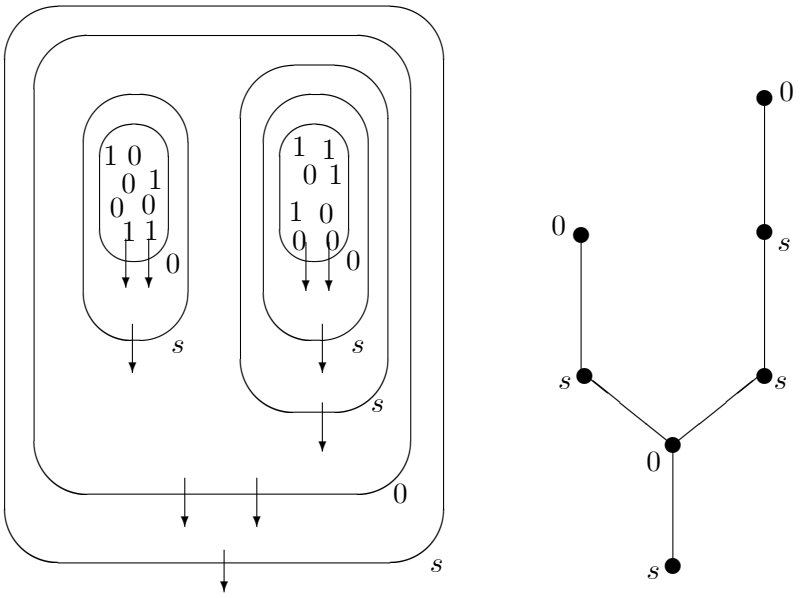


Figure 3: Membrane structure associated with circuit from Fig. 2.

constructed from the distributed sub-AID P systems which work in parallel in the global P system, and we obtain a unique result in the following way:

1. for every gate of the circuit with inputs from input gates, we have an appropriate P system simulating it, with the innermost membrane containing the input values;

2. for every gate which has at least one input coming as an output of a previous gate, we construct an appropriate P system to simulate it by embedding in a membrane the "environments" of the P systems which compute the gates at the previous level.

For the particular formula

$$(x_1 \wedge x_2) \vee \neg(x_3 \wedge x_4)$$

and the circuit depicted in Figure 2 we will have:

- $\Pi^{(1)}_{AND}$ computes the first AND_1 gate $(x_1 \wedge x_2)$ with inputs x_1 and x_2.

- $\Pi^{(2)}_{AND}$ computes the second AND_2 gate $(x_3 \wedge x_4)$ with inputs x_3 and x_4; these two P systems, $\Pi^{(1)}_{AND}$ and $\Pi^{(2)}_{AND}$, act in parallel.

- $\Pi^{(3)}_{NOT}$ computes the NOT gate $\neg(x_3 \wedge x_4)$ with input $(x_3 \wedge x_4)$. While $\Pi^{(3)}_{NOT}$ is working, the output value of the first AND_1 gate performs the rules that can be applied (in $\Pi^{(4)}_{OR}$) and at a point waits for the second input (namely, the output of $\Pi^{(3)}_{NOT}$) to come.

- after the second input has entered the inner membrane of the OR gate, the P system $\Pi^{(4)}_{OR}$ will be able to complete its task. The result of the computation for the OR gate (which is the result of the global P system), is sent into the environment of the whole system.

The idea we want to stress here is that, as noticed from the explanations given above, our system has a self-embedded synchronization. By this we mean that if either of the gates AND or OR receives only one (part of the) input from an upper level of the tree, the gate will wait for the other part of the input to come in order to expel the output. In that way, an extra synchronization system, as considered in [8], is not needed in AID P Systems.

Based on the previous explanations the following result holds:

Theorem 4.1 *Every Boolean circuit α whose underlying graph structure is a rooted tree, can be simulated by a P system, Π_α, in linear time. Π_α is constructed from AID P systems of type Π_{AND}, Π_{OR} and Π_{NOT}, by reproducing the structure of the tree associated to the circuit in the architecture of the membrane structure.*

Property 4.1 *Any Boolean circuit α, with n gates, can be simulated using at most $2n$ membranes.*

Proof. Let us consider the worst case in which our circuit contains only OR and AND gates. Then it is obvious that for the individual simulation of these gates we use $2n$ membranes (every gate is simulated by using two membranes). In fact, this coincides exactly with the upper bound stated in the property due to the embedded synchronization and the fact we do not need additional membranes in order to simulate it. □

5 CIRCUIT-SAT Effi ciency

There is an interesting computational problem related to circuits, called CIRCUIT-SAT. Given a circuit C, is there a truth assignment T appropriate to C such that $T(C) = true$? It is easy to argue that CIRCUIT-SAT is computationally equivalent to SAT, and thus presumably very hard.

We can now appeal to a well-known construction to reduce a CIRCUIT-SAT instance to a CNF formula. Given a circuit C, we will construct a CNF formula φ_C such that there is an assignment to the inputs of C producing an output 1 if and only if the formula φ_C is satisfiable. The formula φ_C will have $n + |C|$ variables, where $|C|$ denotes the number of gates in C; if C acts on inputs $x_1, \cdots, x_n$ and contains gates $g_1, \cdots, g_{|C|}$, then φ_C will have variable set $\{x_1, \cdots, x_n, g_1, \cdots, g_{|C|}\}$. For each gate $g \in C$, we define a set of clauses as follows:

1. if $c = \text{AND}(a, b)$, then add $(\neg c \vee a), (\neg c \vee b), (c \vee \neg a \vee \neg b)$;
2. if $c = \text{OR}(a, b)$, then add $(c \vee \neg a), (c \vee \neg b), (\neg c \vee a \vee b)$;
3. if $c = \text{NOT}(a)$, then add $(c \vee a), (\neg c \vee \neg a)$.

The formula φ_C is simply the conjunction of all the clauses over all the gates of C.

We assume below that C consists of gates from a standard complete basis such as AND, OR, NOT and that each gate has fan-in at most 2. Our results can easily be generalized to allow other gates (e.g., with a larger fan-in); the final bounds are interesting as long as the number of clauses per gate and the maximum fan-in in the circuit have constant upper bounds. Recall that a circuit C is a directed acyclic graph (DAG).

We define the underlying undirected graph as G_C:

Definition 5.1 *Given a circuit C with inputs $X = \{x_1, \cdots, x_n\}$ and gates $S = \{g_1, \cdots, g_s\}$, let $G_C = (V, E)$ be the undirected and unweighted graph with $V = X \cup S$ and $E = \{\{x, y\}$: x is an input to gate y or vice versa$\}$.*

Theorem 5.1 *For a circuit C with gates from {AND, OR, NOT}, the CIRCUIT-SAT instance for C can be solved by an AID P system.*

Proof. We only give a sketch of the proof.

We know that a propositional formula φ_C in CNF is simply the conjunction of all the clauses over all the gates of C. In our previous example, for the Boolean circuit considered in Section 4, φ_C is:

$$\begin{aligned}\varphi_C = \; & (\neg c_1 \vee x_1) \wedge (\neg c_1 \vee x_2) \wedge (c_1 \vee \neg x_1 \vee \neg x_2) \wedge \\ & (\neg c_2 \vee x_3) \wedge (\neg c_2 \vee x_4) \wedge (c_2 \vee \neg x_3 \vee \neg x_4) \wedge \\ & (c_2 \vee c_3) \wedge (\neg c_2 \vee \neg c_3) \wedge \\ & (\neg c_1 \vee c_4) \wedge (\neg c_3 \vee c_4) \wedge (\neg c_4 \vee c_1 \vee c_3).\end{aligned}$$

There are already known algorithms which solve `SAT` (written as Boolean propositional formula in CNF) with P systems with active membranes (see [1, 2, 7, 14, 15, 19]). Then our φ_C can be solved easily following the proof ideas from these papers.

We have left the technical details of the proof to the reader. □

6 A DNA-like Proposal to Solve SAT

For a better understanding of the proposed system we start first with an example and then we give the general details.

Let us begin with the example promised in Subsection 2.4, which was first considered in [13] and later mentioned in [21]. Starting from this example we will make a connection between the classical DNA way of solving satisfiability and the 'DNA-like' way of solving it with P systems using the tools of merging and separation of the membranes, and polarizations.

Consider the propositional formula

$$\alpha = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2).$$

Thus, we have two variables with the corresponding graph as depicted in Fig. 4:

As mentioned in Section 2.4, each of the four paths through this graph corresponds to one of the four truth assignments for the variables x_1 and x_2. The core of the procedure of solving SAT with DNA strands is the operation of *separation*. Let us denote by N_0 the initial test tube which contains all four paths (strands) from the initial to the final vertex of the graph. If we

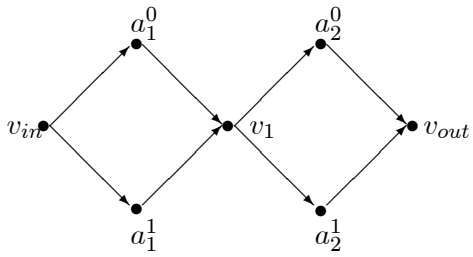


Figure 4: The graph associated with formula α.

apply the operation separate, forming the test tube $+(N_0, a_1^1)$, we get those truth assignments where x_1 assumes the value 1 (true).

A truth assignment is denoted by a two-bit sequence in the natural way. Thus, 01 stands for the assignment $x_1 = 0$, $x_2 = 1$. A similar notation is also used if there are more than two variables. This simple notation of bit sequences is extended to the DNA strands resulting from our basic graphs. Thus, the strand $v_{in}a_1^0v_1a_2^1v_{out}$ is simply denoted by 01.

Following the idea in [21], by $S(N, i, j)$ we denote the test tube of such strands in N where the i-th bit equals j, $j = 0, 1$. Thus, as we observed above, $S(N, i, j)$ results from N by the operation *separate*:

$$S(N, i, j) = +(N, a_i^j).$$

The tube of such strands in N, where the i-th bit equals the complement of j is also considered:

$$S^-(N, i, j) = -(N, a_i^j).$$

Here is the algorithm of solving SAT for the propositional formula α:

(1) $input(N_0)$

(2) $N_1 = S(N_0, 1, 1)$

(3) $N_1' = S^-(N_0, 1, 1)$

(4) $N_2 = S(N_1', 2, 1)$

(5) $merge(N_1, N_2) = N_3$

(6) $N_4 = S(N_3, 1, 0)$

(7) $N_4' = S^-(N_3, 1, 0)$

(8) $N_5 = S(N_4', 2, 0)$

(9) $merge(N_4, N_5) = N_6$

(10) $detect(N_6)$

The program is based on exhaustive search. The initial tube at step (1) contains all possible truth-assignments. The test tube at step (5) contains the assignments satisfying the *first clause* of the propositional formula α. (Either x_1 or x_2 must assume the value 1. At step (2) we have those assignments for which x_1 is 1. Of the remaining ones we still take, at step (4), those for which x_2 is 1.) The assignments in this tube, N_3, are filtered further to yield at step (9) those assignments that also satisfy the second clause of the propositional formula α.

Let us now consider the same propositional formula α and solve it using the new technique we propose here, namely P systems, eventually using polarizations and separation/merging rules following closely the principle of separation/merging as above.

We start our computation having in one membrane, labeled $(1, 1)$, all truth assignments plus the instances of the given problem (α) encoded as follows:

- truth assignments

  00 – $a^0_{1,1}, a^0_{1,2}$, $\quad 01$ – $a^0_{2,1}, a^1_{2,2}$

  10 – $a^1_{3,1}, a^0_{3,2}$, $\quad 11$ – $a^1_{4,1}, a^1_{4,2}$

  More precisely, $a^0_{1,2}$ says that first position - (1) in the subscript indicates the truth-assignment, the second position - (2) indicates the place in the assignment of the value indicated by the superscript - (0).

- clauses

  – $(x_1 \vee x_2)$ – $x^1_{1,1}, x^1_{1,2}$,

  – $(\neg x_1 \vee \neg x_2)$ – $x^0_{2,1}, x^0_{2,2}$.

  Here, by $x^1_{1,2}$ we understand that in the first instance of the formula – (1), variable x_2 (2) is not negated (1).

In the given example, one can imagine 12 objects (given by the sum of the 4 truth assignments (8 objects) and 4 variables of the propositional formula) of two types, a and x, floating in the membrane labeled $(1, 1)$, but, at any time, picking any of these objects we can precisely state which is its value and position/value in the set of string of truth assignments/clauses. In other words, a more clear image is that of some "strings" of two types floating inside that cell (membrane), an image very close to the one of the initial test tube N_0 having all truth assignments encoded as we previously saw.

This last image is also very close to the biological image of DNA in an eukaryotic cell enclosed in the *nuclear envelope* through *inner nuclear membrane* and *outer nuclear membrane*.

Coming back to our example, we now want to separate the membrane labeled $(1, 1)$ having the polarization 1 into two membranes –one containing the truth assignments which have 1 on the first position (10 and 11, encoded as $a^1_{3,1}, a^0_{3,2}$, and $a^1_{4,1}, a^1_{4,2}$, respectively), the variable which is on the first position on the first clause (namely $x^1_{1,1}$) and the variables from the other clause ($x^0_{2,1}$, and $x^0_{2,2}$), while the second contains the rest of the truth assignments (00 and 01) plus the second variable of the first clause ($x^1_{1,2}$). This step simulates the steps (2) and (3) from the DNA variant of the example. (Fig. **??** shows how the example is processed by the two techniques in parallel.)

This is done by applying rule r_1:

$r_1 : [\ ]^1_{1,1} \rightarrow [\ X_{1,1}]^0_{1,1}[\ X_{1,2}]^1_{1,2}$, where

- $X_{1,1} = \{11, 10, \neg x_1, \neg x_2, x_1\}$,
- $X_{1,2} = \neg X_{1,1} = \{x_2\}$.

(1) $input(N_0)$

(1_P) input membrane labeled $1, 1$

(2) $N_1 = S(N_0, 1, 1)$

(3) $N_1' = S^-(N_0, 1, 1)$

(2_P) separation of membrane labeled $1, 1$

(4) $N_2 = S(N_1', 2, 1)$

(3_P) separation of membrane labeled $1, 2$

(5) $merge(N_1, N_2) = N_3$

(4_P) merge between membranes $1, 1$ and $1, 2$ to membrane $2, 1$

(6) $N_4 = S(N_3, 1, 0)$

(7) $N_4' = S^-(N_3, 1, 0)$

(5_P) separation of membrane $2, 1$

(8) $N_5 = S(N_4', 2, 0)$

(6_P) separation of membrane $2, 2$

(9) $merge(N_4, N_5) = N_6$

(7_P) merge between membranes $2, 1$ and $2, 2$ to membrane $3, 1$

(10) $detect(N_6)$

(8_P) detect if there is at least one solution

(We remind the reader that in showing our procedure we start from an example and only after it we define the general framework).

We continue the computation by separating the membrane labeled $(1, 2)$ with polarization 1 into two membranes labeled $(1, 2)$ and $(1, 3)$ with polarizations 0 and 1, respectively. The first membrane will contain the truth assignments that have 0 on the first position and 1 on the second position

(so, only 01) plus the variable x_2 from the first clause. The second membrane is the negation of the above one, thus containing only the truth assignment 00.

A schematic way of solving the problem (by r_1 we mean the application of the general rule r_1, etc.) is depicted in Fig. 5:

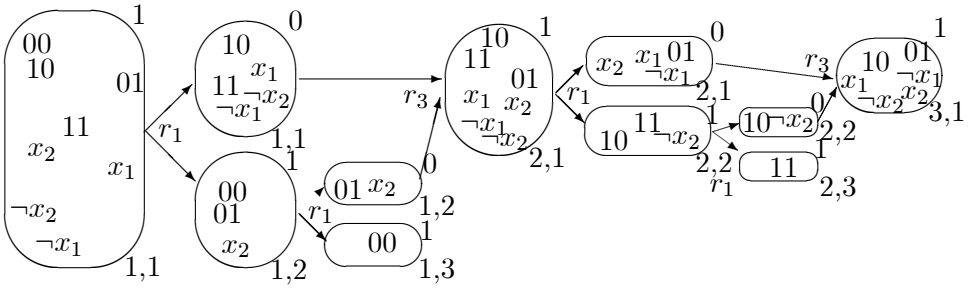


Figure 5: Schematic representation of solving α.

In the next step, membranes labeled $(1,1)$ and $(1,2)$ will merge and form membrane $(2,1)$. The rule of merging is given below:

- $[\ ]^0_{1,1}[\ ]^0_{1,2} \rightarrow [\ ]^1_{2,1}$.

Membrane labeled $(2,1)$ contains strings 10, 11, 01 plus variables x_1, x_2, $\neg x_1$, and $\neg x_2$.

In this phase of the computation, our procedure has computed the *first clause* of the formula and continues with the second one.

From the membrane labeled $(2,1)$ we separate, using the rule r_1 (applicable to membrane $(2,1)$), those truth assignments which begin with 0 from those that do not begin with 0. Thus, the membrane labeled $2,1$ (with polarization 0) will contain the truth assignment 01 and the variables x_1, x_2, and $\neg x_1$, while membrane labeled $(2,2)$ will contain the rest of the objects (namely, the truth assignments 10 and 11 plus the variable $\neg x_2$).

In this step we separate the second membrane produced in the previous step into two membranes: one (with polarization 0) containing the truth-assignments that have 0 on the second position (so only 10) and variable $\neg x_2$, and the other one containing only the string 11. We now merge the two last membranes having polarization 0 ((2, 1), and (2, 2)), thus completing the seventh step of the computation.

One can notice that the answer to our particular problem fbats into the membrane labeled (3, 1) (produced by the union of membranes labeled (2, 1) and (2, 2)). So, now, we are in the stage of *detecting* the result of our problem.

For detection, we will use the following rules:

$r_4 : [\ a^k_{i,j}]^1_{3,1} \rightarrow [\ \ ]^1_{3,1}1$
(where $a^k_{i,j} \in \{a^0_{2,1}, a^1_{2,2}, a^1_{3,1}, a^0_{3,2}\}$ is non-deterministically chosen),

$r_5 : [\ \ ]^1_{1,4}[\ \ ]^1_{2,4} \rightarrow [\ \ ]^1_{2,4}$,

$r_6 : [\ 1]^0_{0,0} \rightarrow [\ \ ]^1_{0,0}yes$.

In the last step of the computation, which is done through rule r_6, the correct answer *yes* is sent to the environment, meaning that our problem has at least one solution.

The example we considered here has, as we have seen, at least one solution to the given problem. In the general case, if the there is no solution to the given problem, the system will expel to the environment the answer `no`.

The problem with this solution is that we first have to produce 2^n truth assignments for variables. In membrane computing, this can be done in various ways –see, e.g., [16], [1], [15] –by using membrane division, separation, etc. However, the respective truth assignments are obtained in 2^n separate membranes. By merging operations, we could put together these truth assignments in a single membrane, but, in order not to "mix" them, we have to encode them separately. This, however, assumes using an exponential number of objects, and thus the system itself has exponential size. For the moment, we do not know how to overcome this diffi culty.

7 Final Remarks

In this paper we have introduced a new way of simulating Boolean gates and circuits, as an answer to a question formulated in [6]: simulate Boolean circuits with P systems using the inhibiting/de-inhibiting controlling mechanism of computation, as introduced and investigated in [6, 7]. This idea is very attractive because apart from using less biological resources (only two objects and two types of rules for the simulation of Boolean gates) than the previous simulations, we also proposed a system which has a self-embedded synchronization of the objects in the circuit without having to coordinate the computation like in other systems.

We have also proposed an approach to solving the `SAT` problem simulating, in P systems with active membranes, the way this problem is effectively solved in laboratories using DNA strands. Technical details of this proposal still remain to be fixed, but we hope that this is a step ahead in our way to the laboratory. In addressing the problem mentioned above we have used polarized/non-polarized P systems, while membranes are capable of merge/separate, changing or not-changing their labels. We found very natural to compare (and study the computational bridge between the) two notions from Natural Computing both having the tools of merging/separation already defined. Actually, such an attempt was already done in [11], but using different protocols of DNA computing and different operations with membranes in P systems.

In the end we invite the reader to investigate, using the new tools (uniformly way of solving `SAT`, but following Lipton's algorithm) presented above, other NP-hard problems, or to try to solve the proposed problem using P systems with different features.

References

[1] A. Alhazov, T.-O. Ishdorj: Membrane Operations in P Systems with Active Membranes. In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini (Eds.): *Second Brainstorming Week on Membrane Computing*. Technical report of Research Group on Natural Computing, University of Seville, TR 01/2004 (2004), 37-44.

[2] A. Alhazov, L. Pan, Gh. Păun: Trading Polarizations for Labels in P Systems with Active Membranes. *Acta Informatica* **41** (2-3) (2004), 111–144.

[3] A. Alhazov, L. Pan: Polarizationless P systems with active membranes. *Grammars* **7** (2004), 141–159.

[4] R. S. Braich, N.Chelyapov, C. Johnson, P. W. K. Rothemund, L. Adleman: Solution to a 20-variable 3-SAT problem on a DNA computer. *Science* **296** (5567) (2002), 499-502.

[5] C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Multiset Processing. *Lecture Notes in Computer Science* **2235**, Springer–Verlag, Berlin (2001).

[6] M. Cavaliere, M. Ionescu, T.-O. Ishdorj: Inhibiting/De-inhibiting Rules in P Systems. *Lecture Notes in Computer Science* **3365**, Springer (2005), 224-238.

[7] M. Cavaliere, M. Ionescu, T.-O. Ishdorj: Inhibiting/De-inhibiting P Systems with Active Membranes. *Cellular Computing (Complexity Aspects)*, ESP PESC Exploratory Workshop. Fénix Editora, Sevilla (2005), 117-130.

[8] R. Ceterchi, D. Sburlan: Simulating Boolean Circuits with P Systems. Workshop on Membrane Computing WMC-Tarragona 2003. In: A. Alhazov, C. Martín-Vide, G. Păun (Eds.): *Lecture Notes in Computer Science* **2933** (2004), 104-122.

[9] J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory.* Springer-Verlag, Berlin (1989).

[10] M.R. Garey, D.J. Johnson: *Computers and Intractability. A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco (1979).

[11] T. Head: Aqueous Simulations of Membrane Computations. *Romanian Journal of Informatics Science and Technology* **5**, 4 (2002).

[12] M. Ionescu, T.-O. Ishdorj: Replicative - Distributed Rules in P Systems with Active Membranes. Proceedings of First International Colloquium on Theoretical Aspects of Computing, Guiyang, China, September 20-24, UNU/IIST Report No. 310 (2004), 263-278, and *Lecture Notes in Computer Science* **4705** (2005), 69-84.

[13] R.J. Lipton: Using DNA to solve NP-complete problems. *Science* **268** (1995), 542-545.

[14] L. Pan, A. Alhazov, T.-O. Ishdorj: Further Remarks on P Systems with Active Membranes, Separation, Merging and Release Rules. *Soft Computing* **8** (2004), 1-5.

[15] L. Pan, T.-O. Ishdorj: P Systems with Active Membranes and Separation Rules. *Journal of Universal Computer Science* **10** (5) (2004), 630-649.

[16] L. Pan, A. Alhazov: Solving HPP and SAT by P Systems with Active Membranes and Separation Rules. *IEEE Transactions on Computers*, submitted (2005).

[17] Ch. P. Papadimitriou: *Computational Complexity*. Addison-Wesley, Reading, MA (1994).

[18] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108-143, and *TUCS Research Report* **208** (1998) (http://www.tucs.fi).

[19] Gh. Păun: P Systems with Active Membranes: Attacking NP-Complete Problems. *Journal of Automata, Languages and Combinatorics* **6** (1) (2001), 75-90.

[20] Gh. Păun: *Computing with Membranes: An Introduction*. Springer-Verlag, Berlin (2002).

[21] Gh. Păun, G. Rozenberg, A. Salomaa. *DNA Computing. New computing paradigms*. Springer-Verlag, Berlin (1998).

[22] M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini, Complexity Classes in Models of Cellular Computation with Membranes. *Natural Computing* **2**, 3 (2003), 265-285.

[23] K. Sakamoto, H. Gounzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, M. Hagiya: Molecular computation by DNA hairpin formation. *Science* **288** (2000), 1223–1226.

[24] A. Salomaa: *Formal Languages*. Academic Press, New York (1973).

[25] A. Salomaa, G. Rozenberg (Eds.): *Handbook of Formal Languages*. Springer-Verlag, Berlin (1997).

Towards a Petri Net Semantics for Membrane Systems

Jetty KLEIJN[1], **Maciej KOUTNY**[2], **Grzegorz ROZENBERG**[1,3]

[1]LIACS, Leiden University
P.O.Box 9512
NL-2300 RA Leiden, NL

[2]School of Comp. Sci.
University of Newcastle
Newcastle upon Tyne
NE1 7RU, UK

[3]Dept. of Comp. Sci.
University of Colorado at Boulder
Boulder, CO 80309-0347, USA

Abstract

We consider the modelling of the behaviour of membrane systems using Petri nets. First, a systematic, structural, link is established between a basic class of membrane systems and Petri nets. To capture the compartmentisation of membrane systems, localities are proposed as an extension of Petri nets. This leads to a locally maximal concurrency semantics for Petri nets. We indicate how processes for these nets could be defined which should be of use in order to describe what is actually going on during a computation of a membrane system.

1 Introduction

In the past 7 years *membrane systems*, also known as *P systems*, have received a lot of attention and in the process became a prominent new computational model [16, 17, 18, 24]. They are inspired by the compartmentisation of living cells and its effect on their functioning. A key structural notion is that of a *membrane* by which a system is divided into compartments where

chemical reactions can take place. These reactions transform multisets of objects present in the compartments into new objects, possibly transferring objects to neighbouring compartments, including the environment. Consequently, the behavioural aspects of membrane systems are based on sets of *reaction rules* defined for each compartment. A distinguishing feature of membrane systems is that the system is assumed to evolve in a synchronous fashion, meaning that there is a global clock common for all the compartments. Within each time unit, the system is transformed by the rules which are applied in a maximally concurrent fashion (this means that no further rules in any compartment could have been applied in the same time unit). These transformations are applied starting from an initial distribution of objects. Depending on the exact formalisation of the model, the notion of a successful (or halting) computation is defined together with its output, e.g., the number of objects sent to the environment.

The above describes the functionality of the basic membrane system model, according to [16, 18]. In addition, many different extensions and modifications of that basic model have been proposed and studied, such as priorities and catalysts. Moreover, those studies have been mostly focussed on the computational power of the models considered, including various aspects of complexity.

Given the existing body of results on the possible outcomes of computations of membrane systems, we feel that we are now in a position to also investigate and describe what is actually going on during a computation. The situation may be compared to that in the field of the semantics of programming languages based on input-output relations where the operational semantics was added to deal with the correctness of potentially non-terminating and concurrent programs. In this paper we propose to undertake this endeavour using the Petri net model (see, e.g., [20]). The reason is that they have local transformation rules and support the modelling of causality and concurrency in a direct and explicit way. In a nutshell, a Petri net is a bipartite directed graph consisting of two kinds of nodes, called *places* and *transitions*. Places together with their markings indicate the local availability of resources and thus can be used to represent objects in specific compartments, whereas transitions are actions which can occur depending on local conditions related to the availability of resources and thus can be used to represent reaction rules associated with specific compartments. When a transition occurs it consumes resources from its input places and produces items in its output places thus mimicking the effect of a reaction rule.

The basic idea of modelling a membrane system using a Petri net can be

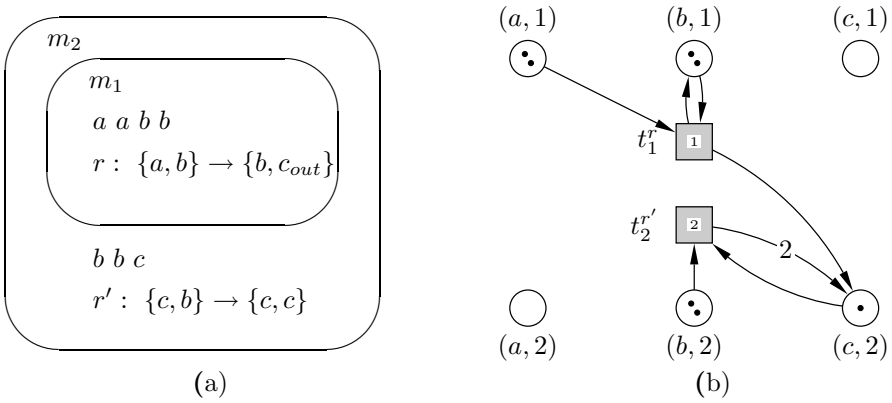


Figure 1: A membrane system (a), and the corresponding Petri net (b).

explained through an example shown in Figure 1(a). The system depicted there consists of two nested membranes (the inner membrane m_1 and the outer membrane m_2), two rules (rule r associated with the compartment c_1 inside the inner membrane, and rule r' associated with the compartment c_2 surrounded by m_2, i.e., in-between the two membranes), and three symbols denoting molecules (a, b, and c). Initially, the compartment c_1 contains two copies of both a and b, and c_2 contains two copies of b and a single copy of c. To model this membrane system using a Petri net, we introduce a separate place (x, j) for each kind of molecule x and compartment c_j. As usual, places are drawn as circles with the number of the currently associated resources represented as tokens (small black dots). For each rule r associated with a compartment c_i we introduce a separate transition t_i^r, drawn as a rectangle. Transitions are connected to places by weighted directed arcs, and if no weight is shown it is by default equal to 1. If the transformation described by a rule r of compartment c_i consumes k copies of molecule x from compartment c_j, then we introduce a k weighted arc from place (x, j) to transition t_i^r, and similarly for molecules produced by transformations. Finally, assuming that initially compartment c_j contained n copies of molecule x, we introduce n tokens into place (x, j). The resulting Petri net is depicted in Figure 1(b). As argued later on, Petri nets derived in this way can be used to describe issues related to concurrency in the behaviour of the original membrane systems.

Applying Petri nets to model membrane systems is by no means an original idea. Since multiset calculus is basic for membrane systems and also

for computing the token distribution in Petri nets [2], some connections have already been established. Some authors have in fact already proposed to interpret reaction rules of membrane systems using Petri net transitions, e.g., [4, 19]. Our aim is to demonstrate that a relationship between Petri nets and membrane systems can be established at the system level. We achieve this by defining a class of Petri nets suitable for the study of behavioural aspects of membrane systems and other systems exhibiting a mix of synchronous and asynchronous execution rules. This latter feature is motivated by the observation that the assumed strict global synchronicity of the membrane systems is not always reasonable from the biological point of view as already observed in [16]. In fact, [8] proposes to drop this assumption completely and considers fully asynchronous and sequential membrane systems; also the membrane systems of [4] are sequential, whereas [3] advocates that reactions are assigned their own execution times and uses a form of local synchronicity.

We intend to demonstrate that Petri nets obtained from membrane systems in the way described above provide a suitable model to capture and investigate the behavioural properties of membrane systems. In this sense the paper is more directed towards the computations taking place in membrane systems. After recalling the definition of membrane systems, we introduce a general class of Petri nets which can be used to define their formal concurrency semantics. This concurrency semantics will be built upon a well established technique of *unfolding* Petri nets, leading to *processes* which formalise concurrent execution histories. The paper deliberately avoids going into full technical details of the formal presentation, aiming instead at conveying the basic ideas of our proposal. Most of the formalities and proofs are delegated to the companion paper [13].

In this paper, a multiset (over a set X) is a function $\mathsf{m} : X \to \mathbb{N}$. By $\mathbb{N}^X$ we denote the set of multisets over X. For two multisets m and m' over X, we denote $\mathsf{m} \leq \mathsf{m}'$ if $\mathsf{m}(x) \leq \mathsf{m}'(x)$ for all $x \in X$. Moreover, a subset of X may be viewed through its characteristic function as a multiset over X, and for a multiset m we denote $x \in \mathsf{m}$ if $\mathsf{m}(x) \geq 1$. The sum of two multisets m and m' over X is given by $(\mathsf{m} + \mathsf{m}')(x) \stackrel{\text{df}}{=} \mathsf{m}(x) + \mathsf{m}'(x)$, the difference by $(\mathsf{m} - \mathsf{m}')(x) \stackrel{\text{df}}{=} \max\{0, \mathsf{m}(x) - \mathsf{m}'(x)\}$, as a total function extending set difference. The multiplication of m by a natural number n is given by $(n \cdot \mathsf{m})(x) \stackrel{\text{df}}{=} n \cdot \mathsf{m}(x)$. Moreover, any finite sum $\mathsf{m}_1 + \cdots + \mathsf{m}_k$ will also be denoted as $\sum_{i \in \{1,\ldots,k\}} \mathsf{m}_i$.

2 Basic Membrane Systems

For the purposes of this paper, it suffices to consider the most basic definition of membrane systems [17, 18]. Throughout the paper a *membrane system* (of degree $m \geq 1$) is a construct

$$\Pi \stackrel{\text{df}}{=} (V, \mu, w_1^0, \ldots, w_m^0, R_1, \ldots, R_m)$$

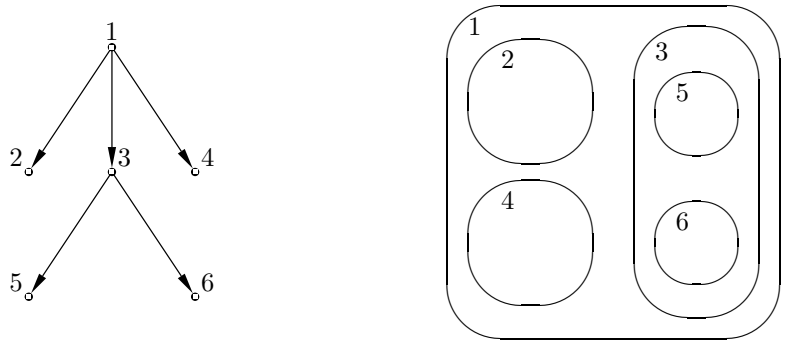
where:

- V is a finite *alphabet* consisting of (names of) objects;
- μ is a *membrane structure* given by a rooted tree with m nodes, representing the membranes, as illustrated in Figure 2 — without loss of generality, we assume that the nodes are given as the integers $1, \ldots, m$, and $(i, j) \in \mu$ will mean that there is an edge from i (parent) to j (child) in the tree of μ;
- each w_i^0 is a multiset of objects initially associated with membrane i;
- each R_i is a finite set of *evolution rules* r associated with membrane i, of the form:
$$lhs^r \rightarrow rhs^r$$
where lhs^r — the left hand side of r — is a non-empty multiset over V, and rhs^r — the right hand side of r — is a non-empty multiset over
$$V \cup \{a_{out} \mid a \in V\} \cup \{a_{in_j} \mid a \in V \text{ and } (i, j) \in \mu\} .$$

Symbols a_{in_j} represent objects a that will be sent to a child node j and a_{out} stands for an a that will be sent out to the parent node. Without loss of generality,[1] we additionally assume that no evolution rule r associated with the root of the membrane structure uses any a_{out} in rhs^r.

A membrane system Π as above evolves from configuration to configuration as a consequence of the application of (multisets of) evolution rules in each membrane. Formally, a *configuration* is a tuple $C \stackrel{\text{df}}{=} (w_1, \ldots, w_m)$ where each w_i is a multiset of object names; we define a *vector multi-rule* $\vec{R}$ as an element of $\mathbb{N}^{R_1} \times \cdots \times \mathbb{N}^{R_m}$. Given a vector multi-rule $\vec{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$, we use as additional notation $lhs_i = \sum_{r \in R_i} \widehat{R}_i(r) \cdot lhs^r$ for

[1] Since the environment can always be modelled by adding a new root to the membrane structure.



(a) (b)

Figure 2: A membrane structure (a); and the corresponding compartments (b).

the multiset of all objects in the left hand sides of the rules in $\widehat{R}_i$ and, similarly, $rhs_i = \sum_{r \in R_i} \widehat{R}_i(r) \cdot rhs^r$ is the multiset of all — possibly indexed — objects in the right hand sides.

Given two configurations, $C = (w_1, \ldots, w_m)$ and $C' = (w'_1, \ldots, w'_m)$, C can *evolve* into C' if there exists a vector multi-rule $\vec{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$ such that for every $1 \leq i \leq m$, the following hold

(i) $lhs_i \leq w_i$;

(ii) there is no rule r in R_i such that $lhs^r + lhs_i \leq w_i$; and

(iii) for each object $a \in V$,

$$w'_i(a) = w_i(a) - lhs_i(a) + rhs_i(a) + rhs_{parent(i)}(a_{in_i}) + \sum_{(i,j) \in \mu} rhs_j(a_{out}) \ ,$$

where $parent(i)$ is the father membrane of i unless i is the root in which case $parent(i)$ is undefined and $rhs_{parent(i)}(a_{in_i})$ is omitted. Note that any j in the last term must be a child membrane of i.

By (i), the configuration C has in each membrane i enough occurrences of objects for the application of the multiset of evolution rules $\widehat{R}_i$. Maximal concurrency is captured by (ii) according to which in none of the membranes an additional evolution rule can be applied. Observe that some of the $\widehat{R}_i$'s in $\vec{R}$ may be empty i.e., no evolution rules associated with the corresponding

membranes i can be used. Finally, (iii) describes the effect of the application of the rules in $\vec{R}$.

By $C \stackrel{\vec{R}}{\Longrightarrow} C'$ we denote that C evolves into C' due to the application of $\vec{R}$. Note that the evolution of C is non-deterministic in the sense that there may be different vector multi-rules applicable to C as described above. A (finite) *computation* of Π is now a (finite) sequence of evolutions starting from the initial configuration $C_0 \stackrel{\text{df}}{=} (w_1^0, \ldots, w_m^0)$.

3 Petri Nets

We first recall the key notions of the standard Petri net model. A *PT-net* is a tuple $N \stackrel{\text{df}}{=} (P, T, W, M_0)$ such that P and T are finite disjoint sets; $W : (T \times P) \cup (P \times T) \to \mathbb{N}$ is a multiset; and M_0 is a multiset of places. The elements of P and T are respectively the *places* and *transitions*, W is the *weight function* of N, and M_0 is the *initial* marking. In diagrams, places are drawn as circles, and transitions as rectangles. If $W(x, y) \geq 1$ for some $(x, y) \in (T \times P) \cup (P \times T)$, then (x, y) is an *arc* leading from x to y. As usual, arcs are annotated with their weight if this is 2 or more. We assume that, for every $t \in T$, there are places p and q such that $W(p, t) \geq 1$ and $W(t, q) \geq 1$.

Places represent local states, while markings are global states of systems represented by PT-nets. Transitions represent actions which may occur at a given marking and then lead to a new marking (the weight function specifies what resources are consumed and produced during the execution of such actions).

Figure 3 shows a PT-net model of a simple one-producer / two-consumers concurrent system, where the producer is represented by the initial token in place p and the consumers by the two tokens in place r. Using transition `a`, the producer repeatedly produces new items (tokens) and *adds* them to place q (intuitively, a buffer between the producer and the two consumers) from where they can be *taken* by one of the two consumers, and then *used* by executing transition `u`. Rather than producing a new item, the producer may at any time *cancel* the production cycle by executing transition `c`.

The *pre-* and *post-multiset* of a transition $t \in T$ are multisets of places given, for all $p \in P$, by:

$$\textsc{pre}_N(t)(p) \stackrel{\text{df}}{=} W(p, t) \quad \text{and} \quad \textsc{post}_N(t)(p) \stackrel{\text{df}}{=} W(t, p) \; .$$

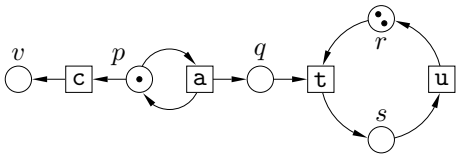


Figure 3: PT-net of the one-producer / two-consumers system.

Both notations extend to multisets of transitions U:

$$\textsc{pre}_N(U) \stackrel{\text{df}}{=} \sum_{t \in U} U(t) \cdot \textsc{pre}_N(t) \qquad \text{and} \qquad \textsc{post}_N(U) \stackrel{\text{df}}{=} \sum_{t \in U} U(t) \cdot \textsc{post}_N(t)\ .$$

A *step* is a multiset of transitions, $U : T \to \mathbb{N}$. It is *enabled* at a marking M if $M \geq \textsc{pre}_N(U)$. We denote this by $M[U\rangle$. Thus, in order for U to be enabled at M, for each place p, the number of tokens in p under M should at least be equal to the total number of tokens that are needed as an input to U, respecting the weights of the input arcs. Moreover, U is a *maximal step* at M if $M[U\rangle$ and there is no transition t such that $M[U + \{t\}\rangle$.

If U is enabled at M, then it can be *executed* leading to the marking $M' \stackrel{\text{df}}{=} M - \textsc{pre}_N(U) + \textsc{post}_N(U)$. This means that the execution of U 'consumes' from each place p exactly $W(p,t)$ tokens for each occurrence of a transition $t \in U$ that has p as an input place, and 'produces' in each place p exactly $W(t,p)$ tokens for each occurrence of a transition $t \in U$ with p as an output place. If the execution of U leads from M to M' we write $M[U\rangle M'$. Whenever U is a maximal step at M, we will also write $M[U\rangle_{max} M'$.

A finite sequence $\sigma = U_1 \ldots U_n$ of non-empty steps is a *step sequence* from the initial marking M_0 if there are markings $M_1 \ldots M_n$ of N satisfying $M_{i-1}[U_i\rangle M_i$ for every $i \leq n$. Such a σ is also called a step sequence from M_0 to M_n, and M_n itself is called a *reachable* marking.

In the same way, we can define step sequences consisting of maximal steps, and markings reachable through such step sequences. Together, they define the *maximal concurrency semantics* of the PT-net N as considered, for instance, in [5].

The example PT-net in Figure 3 admits an infinite number of step sequences. For example, $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ models the following scenario: (i) the producer produces an item which is then deposited into the buffer; (ii) the producer produces another item and, at the same time, one of the consumers takes the previously produced item from the buffer; and (iii) the consumer who retrieved the first item produced uses it and, at the same

time, the second consumer removes the second item produced from the buffer. In Figure 4 we show how this scenario changes the current marking (global state) of the PT-net. As far as the maximal concurrency semantics is concerned, $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ is not allowed: though the first two steps executed are maximal, $\{\mathtt{u},\mathtt{t}\}$ is not since, for instance, the step $\{\mathtt{a},\mathtt{u},\mathtt{t}\}$ is enabled after the execution of $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$, and $\sigma' = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{a},\mathtt{u},\mathtt{t}\}$ rather than σ is part of the maximal concurrency semantics of the PT-net in Figure 3.

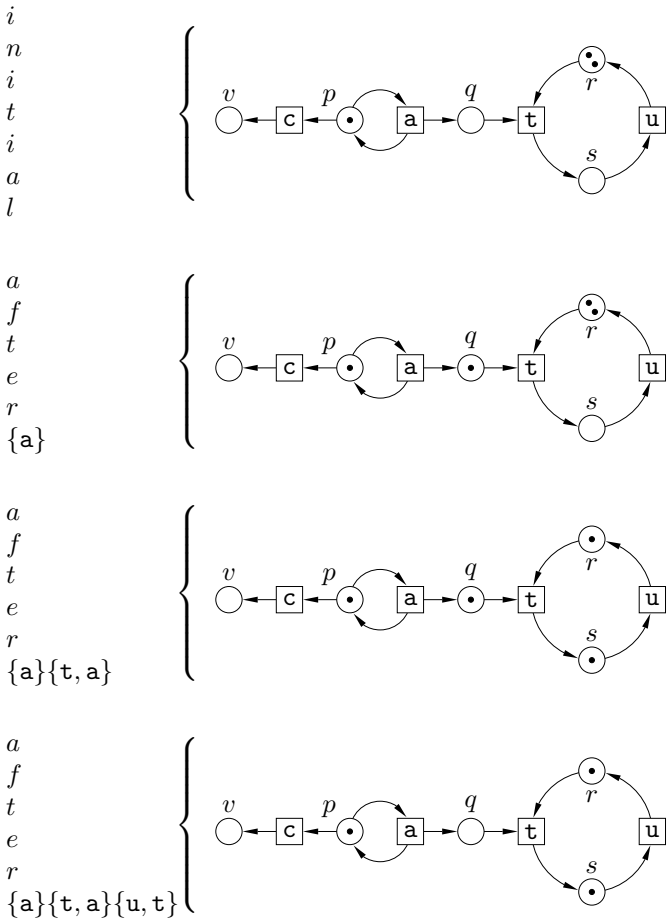


Figure 4: Executing the PT-net according to $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$.

3.1 Petri Nets with Localities

In order to represent the compartmentisation of membrane systems we now introduce a novel extension of the basic net model of PT-nets, by adding the notion of located transitions and locally maximally concurrent executions of co-located transitions. In the proposed way of specifying locality for the transitions in a PT-net, each transition belongs to a fixed unique locality. The exact mechanism for achieving this is to introduce a partition of the set of all transitions, using a locality mapping $\mathfrak{L}$. Intuitively, two transitions for which $\mathfrak{L}$ returns the same value will be co-located.

A *PT-net with localities* (or PTL-net) is a tuple $NL \stackrel{\text{df}}{=} (P, T, W, M_0, \mathfrak{L})$, where $\text{UND}(NL) \stackrel{\text{df}}{=} (P, T, W, M_0)$ is the *underlying* PT-net and $\mathfrak{L} : T \to \mathbb{N}$ is a *location mapping* for the transition set T. In the diagrams of PTL-nets, transitions are shaded rectangles with the locality being shown in the middle. Note that $\mathfrak{L}$ is merely a labelling of transitions, it is not meant as a renaming (as used later for occurrence nets).

The two execution semantics already defined for PT-nets carry over to PTL-nets, after assuming that all the notations concerning the places and transitions of a PTL-net are as in the underlying PT-net, together with the notions of marking, (maximal) step and the result of executing a step.

3.2 Membrane Systems as Petri Nets

In this section, we make our proposal on how membrane systems can be interpreted by Petri nets more precise. Given the definitions of membrane systems and Petri nets with localities, the construction sketched in the introduction can be implemented as follows.

Let $\Pi = (V, \mu, w_1^0, \ldots, w_m^0, R_1, \ldots, R_m)$ be a membrane system of degree m. Then the corresponding PTL-net is $NL_\Pi \stackrel{\text{df}}{=} (P, T, W, M_0, \mathfrak{L})$ where the various components are defined thus:

- $P \stackrel{\text{df}}{=} V \times \{1, \ldots, m\}$;
- $T \stackrel{\text{df}}{=} T_1 \cup \ldots \cup T_m$ where each T_i contains a distinct transition t_i^r for every evolution rule $r \in R_i$;

- for every place $p = (a, j) \in P$ and every transition $t = t_i^r \in T$,

$$W(p,t) \stackrel{\text{df}}{=} \begin{cases} lhs^r(a) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

$$W(t,p) \stackrel{\text{df}}{=} \begin{cases} rhs^r(a) & \text{if } i = j \\ rhs^r(a_{out}) & \text{if } (j,i) \in \mu \\ rhs^r(a_{in_j}) & \text{if } (i,j) \in \mu \\ 0 & \text{otherwise} \end{cases}$$

- for every place $p = (a, j) \in P$, its initial marking is $M_0(p) \stackrel{\text{df}}{=} w_j(a)$.
- for every transition $t = t_i^r \in T$, its locality is $\mathfrak{L}(t) \stackrel{\text{df}}{=} i$.

To capture the very tight correspondence between the membrane system Π and the PTL-net NL_Π, we introduce a straightforward bijection between configurations of Π and markings of NL_Π, based on the correspondence of object locations and places.

Let $C = (w_1, \ldots, w_m)$ be a configuration of Π. Then the corresponding marking $\phi(C)$ of NL_Π is given by $\phi(C)(a, i) \stackrel{\text{df}}{=} w_i(a)$, for every place (a, i) of NL_Π. Similarly, for any vector multi-rule $\vec{R} = (\widehat{R}_1, \ldots, \widehat{R}_m)$ of Π, we define a multiset $\psi(\vec{R})$ of transitions of NL_Π such that $\psi(\vec{R})(t_i^r) \stackrel{\text{df}}{=} \widehat{R}_i(r)$ for every $t_i^r \in T$. It is clear that ϕ is a bijection from the configurations of Π to the markings of NL_Π, and that ψ is a bijection from vector multi-rules of Π to steps of NL_Π.

It should be clear that not every PTL-net can be obtained from a membrane system using the transformation described above. For example, in any net NL_Π, two transitions sharing an input place will always have the same locality assigned by $\mathfrak{L}$.

We now can formulate a fundamental property concerning the relationship between the dynamics of the membrane system Π and that of the corresponding PTL-net:

$$C \stackrel{\vec{R}}{\Longrightarrow} C' \quad \textit{if and only if} \quad \phi(C)\ [\psi(\vec{R})\rangle_{max}\ \phi(C')\ .$$

Since the initial configuration of Π corresponds through ϕ to the initial marking of NL_Π, the above immediately implies that the computations of Π coincide with the maximal concurrency semantics of the PTL net NL_Π.

The reader might by now have observed that the membrane structure of Π is used in the definitions of the static structure of the PTL-net NL_Π (i.e., in the definitions of places, transitions and the weight function), but as

far as maximal concurrency semantics is concerned, the locality information for transitions in the form of the mapping $\mathfrak{L}$ of NL_Π is not relevant (the structure of Petri nets explicitly supports the locality aspects of the resources consumed and produced by transitions). However, it allows us to define local synchronicity presented next.

3.3 Locally Maximal Concurrency Semantics of PTL-Nets

Consider the PTL-model of the producer/consumer example as depicted in Figure 5. It conveys, in particular, the information that transitions `a` and `c` are assigned one locality, whereas transitions `t` and `u` are assigned another locality. This reflects the view that the producer operates away from the two consumers.

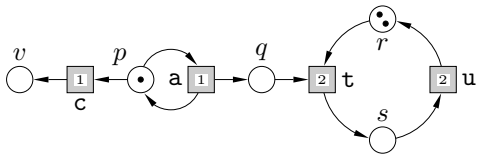


Figure 5: PTL-net of the one-producer / two-consumers system.

To define a right semantical model reflecting this distribution of computing agents, we need to change the enabling condition for steps. Now, intuitively, only those steps are allowed to occur which are maximally concurrent within the localities given by $\mathfrak{L}$.

In a PTL-net $NL = (P, T, W, M_0, \mathfrak{L})$, a step $U : T \to \mathbb{N}$ is *locally max-enabled* at a marking M if it is enabled at M in $\text{UND}(NL)$ and, in addition, there is no transition t such that $\mathfrak{L}(t) \in \mathfrak{L}(U)$ and $U+\{t\}$ is still enabled at M in $\text{UND}(NL)$. Thus a step which is locally max-enabled at a marking is not necessarily a maximal step at that marking. The induced notions of a locally maximal step sequence and marking reachability are then defined as usual using the just defined notion of enabledness.

We now can look at the impact the various definitions of enabledness have on the set of legal behaviours of a Petri net. Looking at the PT-net N in Figure 3 and PTL-net NL in Figure 5, we can observe the following. First of all, the step sequence $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$, which was possible for N, is a legal behaviour of NL under the locally maximal concurrency semantics as are many others, like $\{\mathtt{a}\}\{\mathtt{a}\}\{\mathtt{a}\}$ and $\{\mathtt{a}\}\{\mathtt{a}\}\{\mathtt{t},\mathtt{t}\}$. (Recall here that $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ was disallowed by the maximal concurrency semantics.)

However, there are also step sequences of N which are not part of the locally maximal concurrency semantics of NL; e.g., $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{t}\}$ since after executing $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$ it is possible to execute step $\{\mathtt{u},\mathtt{t}\}$ which is strictly greater than $\{\mathtt{t}\}$ and transitions $\mathtt{t}$ and $\mathtt{u}$ are co-located.

Coming back to the example shown in Figure 1(b), we have the following step sequences in the maximal concurrency semantics: the empty sequence, $\{t_1^r, t_1^r, t_2^{r'}\}$ and $\{t_1^r, t_1^r, t_2^{r'}\}\{t_2^{r'}\}$. The locally maximal concurrency semantics, on the other hand, yields several additional step sequences, like $\{t_1^r, t_1^r\}\{t_2^{r'}, t_2^{r'}\}$ and $\{t_2^{r'}\}\{t_1^r, t_1^r\}\{t_2^{r'}\}$. Note further that it does not allow $\{t_1^r, t_1^r\}\{t_2^{r'}\}$ which, in turn, is allowed by the standard step sequence semantics.

To summarise, PT-nets admit both standard and maximal concurrency semantics, while for PTL-nets we have in addition locally maximal concurrency semantics. In particular, this means that we cannot identify the exact semantical model just by looking at a net's structure; we always need to specify which execution semantics is being used.

4 Causality and Concurrency

All three variants of step sequence semantics of a Petri net considered in this paper provide important insights into the concurrency aspects of the underlying systems. They are, however, still sequential in nature in the sense that steps occur ordered thus obscuring the true causal relationships between the occurrences of transitions. On the other hand, information on causal relationship is often of high importance for system analysis and/or design. Petri nets can easily support a formal approach where this information is readily available as was recognised a long time ago, see [15] where it was proposed to unfold behaviours into structures allowing an explicit representation of causality, conflict and concurrency. A well-established way of developing such a semantics for the standard PT-nets is based on a class of acyclic Petri nets, called *occurrence nets* [21]. What one essentially tries to achieve is to trace the changes of markings due to transitions being executed along some legal behaviour of the original PT-net, and in doing so record which resources were consumed and produced.

In this section, we first explain the main ideas behind the causality semantics based on standard step sequences of PT-nets. After that, we show how this approach could be adapted to work for the locally maximal concurrency semantics of PTL-nets. Note that the maximal concurrency semantics of a PT-net coincides with the locally maximal concurrency semantics of this

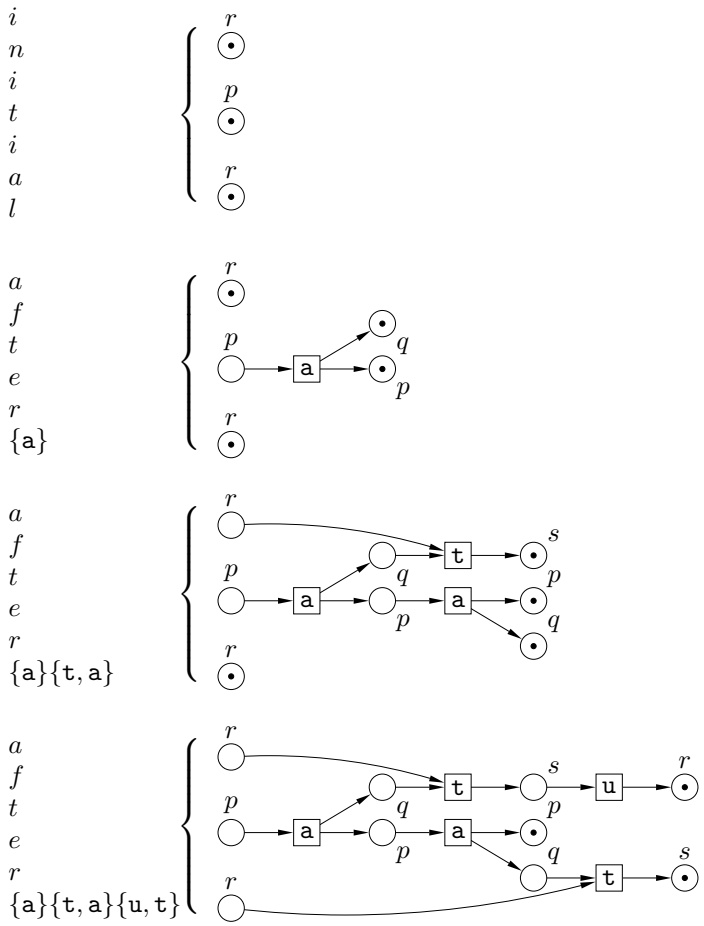


Figure 6: Constructing an occurrence net corresponding to $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$.

PT-net after extending it to a PTL-net with all transitions mapped to the same locality; hence we will only consider explicitly the locally maximal concurrency semantics.

4.1 Causal Behaviours of PT-Nets

Looking at the sequence $\sigma = \{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}\{\mathtt{u},\mathtt{t}\}$ of executions in Figure 4, it is not immediate that transition u could have occurred before the second occurrence of transition a or, in other words, that the former is not causally dependent on the latter.

Figure 6 illustrates the idea in which we *unfold* the scenario represented by σ. The initial stage shows just the initial marking which includes two separate (labelled) *conditions* (this is how places are called in occurrence nets) to represent the two initial tokens in place r. Executing step $\{\mathtt{a}\}$ consumes the p-condition, creates an a-event (this is how transitions are called in occurrence nets), as well as two new conditions: a p-condition and a q-condition. An important point is to notice that we create a fresh p-condition rather than a loop back to the initial one since we want to distinguish between different occurrences of the same token; as a result the occurrence net being constructed will be an acyclic graph. Another important point is that the environment of the generated a-event corresponds exactly to the environment of transition a; namely, it consumes a p-token and creates a p-token and a q-token. After that, executing step $\{\mathtt{t}, \mathtt{a}\}$ consists in consuming three conditions and creating two events and three fresh conditions, and similarly for the last step $\{\mathtt{u}, \mathtt{t}\}$. And, as a final result, we obtain an acyclic net labelled with places and transitions of the original PT-net; it is called a *process* of the original PT-net. The process net has a default initial marking consisting of a token in each of the conditions without an incoming arc.

It is now possible to look both at the structure of the process net and the executions which are possible from its default initial marking, making some important observations relating to:

- *Causality.* The causality relationships among the executed transitions can be read-off by following directed paths between the events; for example in Figure 6, the lower t-event is caused by both a-events, while the upper one is caused only by the leftmost a-event.

- *Concurrency.* Events for which there is no directed path from one to another can be thought of as concurrent.

- *Reachability.* Any maximal set of conditions for which there is no directed path from one condition to another corresponds to a reachable marking of the original PT-net.

- *Representation.* The step sequence on the basis of which the process was created can be executed from the initial default marking in the occurrence net. So the original behaviour has been retained. In Figure 6, there are 13 different step sequences generated by the process net defined by $\sigma = \{\mathtt{a}\}\{\mathtt{t}, \mathtt{a}\}\{\mathtt{u}, \mathtt{t}\}$, including σ itself.

- *Soundness.* Any step sequence which can be executed from the default initial marking to the default final marking (consisting of tokens placed

in each of the conditions without an outgoing arc) of the process net is also a legal step sequence of the original PT-net. Processes provide a highly compressed representation of step sequence behaviours of the original PT-net (this feature has been exploited to a significant degree in the development of efficient model checking algorithms for PT-nets).

The above advantages of the process nets of the standard PT-nets lead us to consider a similar treatment for the PTL-net model and their locally maximal concurrency semantics.

4.2 Causal Behaviours of PTL-Nets

As a first attempt, we simply adopt the unfolding strategy as in the PT-net case. We only ensure that the step sequence consists of (locally) maximal steps. Moreover, we preserve the localities of the transitions in the events created while constructing the occurrence net. Figure 7 shows the result for the PTL-net of Figure 5 and the step sequence $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$ which is allowed in the maximal and thus also in the locally maximal concurrency semantics (both the occurrence net and its default initial marking are depicted). Although this is straightforward, we still need an argument that the resulting process is what one would want to take for further analyses. In particular, one would want to retain the soundness of the previous construction. In the case of our example, we can execute the occurrence net and conclude that under the maximal rule it admits the original sequence, whereas under the locally maximal rule it admits two more step sequences, $\{\mathtt{a}\}\{\mathtt{a}\}\{\mathtt{t}\}$ and $\{\mathtt{a}\}\{\mathtt{t}\}\{\mathtt{a}\}$. And, clearly, both are legal step sequences of the original PTL-net in the locally maximal concurrency semantics.

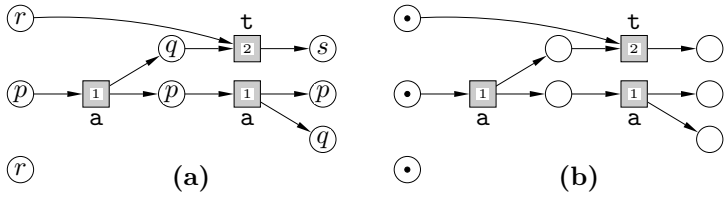


Figure 7: Process net corresponding to the step sequence $\{\mathtt{a}\}\{\mathtt{t},\mathtt{a}\}$ **(a)**; and its default initial marking **(b)**.

In general, however, it would be too hasty to accept the standard unfolding routine as satisfactory. Consider, for example, the PTL-net in Figure 8(a) and its step sequence $\{\mathtt{t},\mathtt{u},\mathtt{v}\}\{\mathtt{w},\mathtt{z}\}$ consisting of locally maximal

steps. Proceeding as in the previous case, we obtain an occurrence net shown in Figure 8(b). And the problem is that it has an execution from the default initial marking (using only locally maximal steps) which corresponds to $\{\mathtt{u},\mathtt{v}\}\{\mathtt{t},\mathtt{z}\}\{\mathtt{w}\}$. This step sequence, however, is not a locally maximal step sequence of the original PTL-net as in the second step it is possible to add transition x which is co-located with transition z.

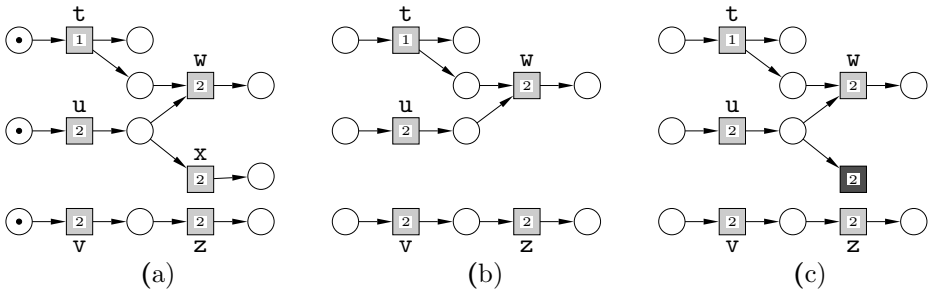


Figure 8: PTL-net **(a)**; an occurrence net constructed from step sequence $\{\mathtt{t},\mathtt{u},\mathtt{v}\}\{\mathtt{w},\mathtt{z}\}$ **(b)**; and a barbed process **(c)**.

An intuitive reason why the standard construction fails to work for the PTL-net in Figure 8(a) is that such an unfolding 'forgets' that transition x was enabled at a stage where transition w was selected. Then, delaying the execution of the w-event, creates a situation where the executed step (though locally maximal within the occurrence net since the knowledge of the enabledness of x is lost) does not correspond to a locally maximal step within the PTL-net.

Our approach to cope with this problem in [13] is to equip occurrence nets generated by PTL-nets with additional *barb-events*, represented by darkly shaded rectangles. Barb-events are not labelled with transition names and are not meant to be executed; rather, they are used in the calculation of the enabled sets of events. Such occurrence nets are called *barbed* processes. Rather than providing a full formal definition of how barb-events are added during the unfolding procedure, which we give in the companion paper [13], we only mention here that it is based on checking that transitions have not been included in the executed scenario since another co-located transition was selected. Figure 8(c) illustrates the modified construction for the net in Figure 8(a,b).

After executing $\{\mathtt{u},\mathtt{v}\}$, it is now impossible to select $\{\mathtt{t},\mathtt{z}\}$ since there is a record in the form of the barb-event that such a step would not be

maximal in the locality to which transition $\{\mathtt{z}\}$ belongs. The only way of continuing is to execute $\{\mathtt{t}\}$ and after that $\{\mathtt{z}, \mathtt{w}\}$, generating a legal step sequence $\{\mathtt{u}, \mathtt{v}\}\{\mathtt{t}\}\{\mathtt{z}, \mathtt{w}\}$.

5 Summary and Conclusions

In this paper we have proposed an approach to the modelling of the behaviour of membrane systems through a class of Petri nets with localities (PTL-nets).

We gave first a formal translation for a basic class of membrane systems, and argued that the structure of the (maximally concurrent) computations of such membrane systems is faithfully reflected by the maximal concurrency semantics of the corresponding PTL-nets. This corresponds to the situation whereby all the rules are governed by a single global clock which corresponds to the case of maximally concurrent executions, as investigated in [5]. Hence the results on the reachability of certain markings (or, equivalently, configurations in membrane systems) developed there could form the basis of an investigation, e.g., whether a particular combination of molecules in certain compartments can happen in the legal evolutions of a membrane system.

After that we moved to a less centralised view of concurrent executions, as already advocated e.g., in [8], and defined a locally maximal concurrency semantics for PTL-nets. However, in case of individual localities for all transitions, we are not exactly dealing with the asynchronous or sequential systems, proposed by [8]. Since we maintain the requirement of locally maximal concurrency executions, the resulting systems exhibit maximal *autoconcurrency*.

In the model of PTL-nets there are no additional requirements on the relationship between transitions and their localities; in particular, as already mentioned, transitions with shared input places do not have to be co-located. Moreover, the flow of resources among the localities does not necessarily follow a tree-like structure. In fact, PTL-nets with their locally maximal concurrency semantics constitute a very general framework in which membrane systems and even conglomerates of membrane systems (organisms) can be expressed and studied.

An important feature characterising the proposed basic PTL-net model is its robustness, in the sense of being easily extendable to handle salient features of more sophisticated membrane systems. Examples of such features are: (i) priorities among rules which can be dealt with using Petri nets with priorities, e.g., as in [1]; (ii) catalysts governing the enabling of

the reaction rules purely by their presence which can be dealt with using Petri nets with read arcs, e.g., as in [23]; (iii) substances forbidding certain reactions which can be dealt with using Petri nets with inhibitor arcs, e.g., as in [12]; and (iv) dissolution of membranes which can be dealt with using Petri nets with transfer arcs; e.g., as in [22, 6]. We could also consider membrane systems with rules having variable discrete durations, by suitably exploiting the locally maximal concurrency semantics of PTL-nets. Further investigation is also needed into the relationship between various P systems and a wide variety of restricted/extended Petri nets, such as [9, 10].

We finally outlined how a causality based semantics of PTL-nets could be defined and used to analyse the intricate details of concurrent computations of membrane systems. The proposed semantics is based on the unfolding of PTL-nets with the novel feature of *barb-events* needed to reflect choices in the locally maximal executions. Among the potential benefits of the proposed unfolding-based semantics is the efficient model checking approach to the verification of properties of concurrent systems modelled as Petri nets [7, 14, 11].

Summarising, we have developed a new systematic link between Petri nets and membrane systems which (hopefully) is useful for both areas. We see this formalisation only as a beginning of the research into the representation of the behaviour of membrane systems through concurrent processes.

Clearly, one could simply use the basic model of PT-nets and simulate by 'brute force' the behaviour of membrane systems. In general, however, a biologist's interest will be in how a system functions and not just in what is delivered at the end. From the modelling point of view it is therefore more convenient to include localities as a direct interpretation of 'where is what'. This also provides the possibility to introduce a notion of local synchronicity as opposed to a global clock governing the evolution of a system. The process semantics of PT-nets provides an additional formal tool to study how a system functions rather than what it computes. Whereas step sequences can be viewed as ordered by a clock, processes can be used to represent causalities. Moreover using (infinite) processes, also ongoing (potentially infinite) system behaviour can be investigated, which is also interesting from a biological point of view.

For PT-nets the notion of locality inspired by membrane systems is a new interesting feature. The process semantics for PTL-nets working under the (locally) maximal concurrency semantics still has to be developed. In this paper we have briefly indicated how the technical problems could be solved. In addition, a proper notion of causality (order relation) based on

processes (see the semantical scheme of [12]) and relevant for the biologically motivated membrane systems has to be identified as well.

5.0.1 Acknowledgments

The authors are grateful to Hendrik Jan Hoogeboom for his comments on an earlier version of this paper. We would also like to thank the anonymous referees for their constructive comments. This research was supported by the EPSRC project CASINO.

References

[1] E. Best, M. Koutny: Petri Net Semantics of Priority Systems. *Theoretical Computer Science* **96** (1992), 175–215.

[2] C. S. Calude, Gh. Păun, G. Rozenberg, A.Salomaa (Eds.): Multiset Processing. Mathematical, Computer Science, and Molecular Computing Points of View. *Lecture Notes in Computer Science* **2235**, Springer-Verlag (2001).

[3] M. Cavaliere, D. Sburlan: Time-Independent P Systems. In: G. Mauri et al. (Eds.): Proceedings of *WMC 2004 Lecture Notes in Computer Science* **3365**, Springer-Verlag (2005), 239–258.

[4] S. Dal Zilio, E. Formenti: On the Dynamics of PB Systems: a Petri Net View. In: C. Martín-Vide et al. (Eds.): Proceedings of *WMC 2003 Lecture Notes in Computer Science* **2933**, Springer-Verlag (2004), 153–167.

[5] R. Devillers, R. Janicki, M. Koutny, P. E. Lauer: Concurrent and Maximally Concurrent Evolution of Nonsequential Systems. *Theoretical Computer Science* **43** (1986), 213–238.

[6] C. Dufourd: Réseaux de Petri avec Reset/Transfert: Décidabilité et Indécidabilité. ENS Cachan (1998).

[7] J. Esparza, S. Römer, W.Vogler: An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design* **20 (3)** (2002), 285–310.

[8] R. Freund: Asynchronous P Systems and P Systems Working in the Sequential Mode. In: G. Mauri et al. (Eds.): Proceedings of *WMC*

2004 Lecture Notes in Computer Science **3365**, Springer-Verlag (2005), 36–62.

[9] O. H. Ibarra, Z. Dang, O. Egecioglu: Catalytic P Systems, Semilinear Sets, and Vector Addition Systems. *Theoretical Computer Science* **312 (2–3)** (2004), 379–399.

[10] O. H. Ibarra, H. -C. Ye, Z. Dang: The Power of Maximal Parallelism in P Systems. In: C. Calude et al. (Eds.): Proceedings of *DLT 2004 Lecture Notes in Computer Science* **3340**, Springer-Verlag (2004), 212–224.

[11] V. Khomenko, M. Koutny, W. Vogler: Canonical Prefixes of Petri Net Unfoldings. *Acta Informatica* **40** (2003), 95–118.

[12] H. C. M. Kleijn, M. Koutny: Process Semantics of General Inhibitor Nets. *Information and Computation* **190** (2004), 18–69.

[13] H. C. M. Kleijn, M. Koutny, G. Rozenberg (2005). *Process Semantics of Petri Nets with Localities*. In preparation.

[14] K. L. McMillan: Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. In: G. von Bochmann, D. K. Probst (Eds.): Proceedings of *CAV'1992 Lecture Notes in Computer Science* **663**, Springer-Verlag (1992), 164–174.

[15] M. Nielsen, G. Plotkin, G. Winskel: Petri Nets, Event Structures and Domains, Part I. *Theoretical Computer Science* **13** (1980), 85–108.

[16] Gh. Păun: Membrane Computing, An Introduction. Springer-Verlag (2002).

[17] Gh. Păun: Computing with Membranes. An Introduction. *Bulletin of the EATCS* **67** (1999), 139–152.

[18] Gh. Păun, G. Rozenberg: A Guide to Membrane Computing. *Theoretical Computer Science* **287** (2002), 73–100.

[19] Z. Qi, J. You, H. Mao: P Systems and Petri Nets. In: C. Martín-Vide et al. (Eds.): Proceedings of *WMC 2003 Lecture Notes in Computer Science* **2933**, Springer-Verlag (2004), 286–303.

[20] W. Reisig, G. Rozenberg (Eds.): Lectures on Petri Nets. *Lecture Notes in Computer Science* **1491, 1492**, Springer-Verlag (1998).

[21] G. Rozenberg, J. Engelfriet (1998): Elementary Net Systems. In: *Lectures on Petri Nets I: Basic Models. Lecture Notes in Computer Science* **1491**, Springer-Verlag , 12–121.

[22] R. Valk: Self-Modifying Nets, a Natural Extension of Petri Nets. In: G. Ausiello, C. Böhm (Eds.): Proceedings of *ICALP 1978 Lecture Notes in Computer Science* **62**, Springer-Verlag (1978), 464–476.

[23] W. Vogler: Partial Order Semantics and Read Arcs. *Theoretical Computer Science* **286** (2002), 33–63.

[24] The P Systems Web Page `http://psystems.disco.unimib.it/`

Păun's Systems and Accounting

Waldemar KORCZYNSKI

Kielce University of Technology
ul. Tysiaclecia Panstwa Polskiego 7 Pl-25314 Kielce, Poland
E-mail: korwald@wsu.kielce.pl

Abstract

In this paper an interpretation of Păun's systems in the area of accounting is described. It is shown that some special P-systems may be models for well known fundamental notions of accounting theory.

1 Introduction

Păun's systems may be seen as (sets of) trees with vertices in a given set V labelled by pairs of the form (m, R) where $m : A \rightarrow \mathbf{Nat}$ is a multiset over a given set A and R is a set of *rules*. These rules describe the way of transformation of multisets assigned to the vertices of the tree and the **tree itself**. Interpreting the tree as the infrastructure of a dynamic system, the multisets as the resources of it and the rules as a description of the dynamics of the system one can say that the rules may change the resources **and** the infrastructure of the system. The set $Rules(A, V)$ of rules over A and V consists of *structure preserving rules* describing the *production* and the *transfer* of some objects (the resources) and two kinds of rules (*dissolving-* and *splitting-rules*) changing the structure (the tree). The evolution of such a system may be seen as a partially ordered multiset[1] of occurrences of the rules. Details can be found e.g. in [2].

[1] In the original definition of P-systems a region may contain not a multiset, but a **set** of rules. However in the real world of existing economic systems the economic entities often have at their disposal many identical procedures (e.g. many identical machines) and in this paper we can't ignore this fact.

2 A Model of Accounting System

Economic entities are set apart from the environment by some abstract conditions; the so called "balance equation" being expressions of the form

$$\sum_{p \in assetsParts(A)} value(p) = \sum_{s \in FinSources(A)} value(s)$$

saying that "the (sum of) assets equals the (sum of) liabilities". Balance equations may be added and multiplied by numbers (e.g. by any denomination of a currency). By an *economic entity* we mean any pair of the form (E, R) with E being a balance equation and R being a nonvoid set of (parametrized) procedures. It is a counterpart of the notion of a region in a Păun's system. The balance equation corresponds to the contents of the region (the left side of the equation) and membrane (the equality itself). We also have an additional information about "the origin" of the contents of the region (the right side of the equation). The set R of procedures corresponds exactly to the set of rules of the region. The tree-ordering[2] of the regions corresponds to the "to be a part of" relation and is determined by the standard condition

$$(E_1, R_1) \preceq (E_2, R_2) \Leftrightarrow \exists_{E_3} E_2 = E_1 + E_3$$

The economic processes compared e.g. with computations in a computer are very "slow" and one can define a kind of "simultaneousness". This is not a "true" equivalence but in every day life one can treat as simultaneous events which occur on the same day, week or even year and assign to every event a moment in time. Such a function called a *registration* has to be an injection Accounting systems describe all events always by some numbers understood as a common measure (money) for all goods transferred in the system. This measure assigns to any multiset $\mathfrak{v} : sorts\_of\_goods \rightarrow numbers$ a number $value(\mathfrak{v})$. By an *account* one can understand any named pair of the form $acc = (w_1, w_2)$ with w_1 and w_2 being lists called *debt* and *credit* over an "alphabet" $\mathfrak{A}$ consisting of sequences of the form $\mathfrak{a}$=*(description(*$\mathfrak{a}$*), value(*$\mathfrak{a}$*))*. The elements of the set $\mathfrak{A}$ are interpreted as multisets of goods labelled by their "prices". The form of the first part of such a pair, i.e. the *description*-part is fixed by the appropriate legal regulations of a state, the second part of such a "letter" is defined by the market. Now by a *transaction* we mean any sequence $\mathfrak{t} = (r_1, ..., r_m; r^1, ..., r^n)$ of pairs of the form ($\mathfrak{a}$, acc)

[2]This order is a tree because the balance equations may be added iff they concern disjoint sets of goods.

with $\mathfrak{a} \in \mathfrak{A}$ and *acc* being the name of an account. We identify the economic events with their registration and assume that any economic event has to be recorded on at least two accounts on opposite sides of them. If for $i \leqq m, j \leqq m$ we have $r_i = (\mathfrak{a}_i, acc_i)$ and $r^j = (\mathfrak{a}^j, acc^j)$ then we record the transaction $\mathfrak{t} = (r_1, ...r_m; r^1, ..., r^n)$ on the debt-sides of accounts acc_1, .., acc_m and on the credit-sides of the accounts acc^1, .., acc^n. Every registration of a letter $\mathfrak{a}_i$ ($\mathfrak{a}^j$) on the debt (credit) side of the account acc_i (acc^j) determines a registration-time of this transaction on the account $acc_i(acc^j)$. To every transaction we associate a sequence $t(r_1), ..., t(r_m), t(r^1), ..., t(r^n)$ of the moments of registrations of the transaction $\mathfrak{t} = (r_1, .., r_m; r^1, ..., r^n)$ on the accounts $acc_1, ..., acc_m$, $acc^1, ..., acc^n$. Such transactions correspond to the so called *antiport transport rules* which are the only rules in a Păun's system being a model of an accounting system.

By an *accounting system* one can understand any description of a homomorphism of production, delivering and receiving goods (we call all such processes *performance processes*) processes into economic ones. That only means that some, "noneconomic" aspects of performance have to be forgotten. The economic processes are concurrent that means some events may be incomparable. Unfortunately the fundamental assumption of any accounting system is an assigning to every economic event the unique period of time in which this event has occurred. These periods are linearly ordered isomorphic to the natural numbers. The equivalence classes of the kernel of such an assigning are called *simultaneous* events. The strict linear ordering of these events on both (debit and credit) sides of an account is connected with the linear way of registration and has nothing in common with order of appearance of those "simultaneous" events. This linear extension may be carried out in many ways, and in general there exists no widely accepted method of doing it. In some situations the existence of such a method may be of great importance[3]. The whole description of *performance processes* is a composition of the forgetting-homomorphism mentioned above and the extension of the identity on the set of simultaneous events to a linear ordering. The first part of this composition is precisely described by the corresponding law-regulation, the second one creates sometimes possibilities of corrupt practices[4].

[3]For example the operation of remittance into an account and the operation of withdrawing money from this account may be done on the same day. So in the sense of the "same-day-equivalence" they are simultaneous. However if the account was empty the remittance should be done as the first operation.

[4]Let us note that the same two events α and β may occur on an account, say A, in the order "first α next β and on another account, say B, as "first β then α". This property

3 Concluding Remarks

The classic, time-free membrane systems are good tools for the modelling of causality and conflict relations in various hierarchical systems. Hierarchical systems in which a time registration is needed can be seen as aggregates containing Păun's systems, a kind of (at least partially ordered) time and some rules of event-registration. Accounting systems satisfy these requirements because the registration of economic events is relatively precise described by various law regulations. One can therefore anticipate that this fact serves a motive for a developing of the timed membrane systems and, on the other hand, timed membrane systems become an inspiration for a formal and relatively easy description of many accounting issues .

References

[1] R. Mattesich: *Die wissenschaftlichen Grundlagen des Rechnungswesens*, Duesseldorf (1970).

[2] Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlin (2002).

is very important and the frequency of the appearance of such "inconsistences" may be used as a criterion of "the merits" of a given accounting system. One can say that such a system is good if such different registrations of the same event are always "simultaneous".

Quantum Sequential P Systems with Unit Rules and Energy Assigned to Membranes

Alberto LEPORATI, Giancarlo Mauri, Claudio ZANDRON

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy
E-mail: {leporati,mauri,zandron}@disco.unimib.it

Abstract

We propose a quantum version of P systems with unit rules and energy assigned to membranes. Differently from the classical version, the new quantum P systems do not need to use priorities over rules to be computationally complete.

We also propose a quantum version of register machines as a tool to study the computational power of quantum models of computation.

1 Introduction

P systems (also called *membrane systems*) have been introduced in [21] as a new class of distributed and parallel computing devices, inspired by the structure and functioning of living cells. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin*. Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be applied. Usually, the result of a computation is the multiset of objects contained in an *output membrane* or emitted from the skin of the system.

Recently, the P systems community has shown some interest in the search for *quantum* P systems. In this paper, starting from the ideas exposed

in [17], we propose a quantum version of P systems with unit rules and energy assigned to membranes which have recently appeared in the literature [10]. The proposed quantum P systems are sequential; moreover, at every computation step only one rule can be applied (and hence they are in some sense deterministic). Differently from the classical version, the amount of energy assigned to a membrane is not a property of the membrane itself, but is instead represented by the energy level of a quantum harmonic oscillator which is contained in the region enclosed by the membrane. Another notable difference between the classical and the quantum version of our P systems is that in the quantum version only one rule is assigned to each membrane. As a consequence, we obtain computational completeness without the need to assign priorities to rules, as it is done in the classical case.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For a systematic introduction, we refer the reader to [22]. The latest information about P systems can be found in [25].

This is by no means the first time that energy is considered in P systems. We recall in particular [1, 9, 24, 14, 18, 19, 17].

The paper is organized as follows. In section 2 some preliminaries are given: in particular, we recall register machines (section 2.1), classical P systems with unit rules and energy assigned to membranes [10], together with their computational capabilities (section 2.2), and some notions of quantum computing (section 2.3). In section 3 we define the quantum version of such P systems, and in section 4 we establish their computational completeness. In section 5 we introduce a quantum version of register machines, as a tool to study present and future quantum computational models. Conclusions are given in section 6.

2 Preliminaries

2.1 Register Machines

A *deterministic n–register machine* is a construct $M = (n, P, l_0, l_h)$, where n is the number of registers, P is a finite set of instructions injectively labelled with a given set $lab(M)$, l_0 is the label of the first instruction to be executed, and l_h is the label of the last instruction of P. Registers contain non-negative integer values. Without loss of generality, we can assume $lab(M) = \{1, 2, \ldots, m\}$, $l_0 = 1$ and $l_h = m$. The instructions of P have the following forms:

- $j : (INC(r), k)$, with $j, k \in lab(M)$

  This instruction increments the value contained in register r, and then jumps to instruction k.

- $j : (DEC(r), k, l)$, with $j, k, l \in lab(M)$

  If the value contained in register r is positive then decrement it and jump to instruction k. If the value of r is zero then jump to instruction l (without altering the contents of the register).

- $m : Halt$

  Stop the machine. Note that this instruction can only be assigned to the final label m.

Register machines provide a simple universal computational model. Indeed, the results proved in [11] (based on the results established in [20]) as well as in [12] and [13] immediately lead to the following proposition.

Proposition 1. *For any partial recursive function $f : \mathbf{N}^\alpha \to \mathbf{N}^\beta$ there exists a deterministic $(\max\{\alpha, \beta\} + 2)$–register machine M computing f in such a way that, when starting with $(n_1, \ldots, n_\alpha) \in \mathbf{N}^\alpha$ in registers 1 to α, M has computed $f(n_1, \ldots, n_\alpha) = (r_1, \ldots, r_\beta)$ if it halts in the final label l_h with registers 1 to β containing r_1 to r_β, and all other registers being empty; if the final label cannot be reached, then $f(n_1, \ldots, n_\alpha)$ remains undefined.*

2.2 P Systems with Unit Rules and Energy Assigned to Membranes

A P system with unit rules and energy assigned to membranes [10] of degree $d + 1$ is a construct Π of the form:

$$\Pi = (A, \mu, e_0, \ldots, e_d, w_0, \ldots, w_d, R_0, \ldots, R_d)$$

where:

- A is an alphabet of *objects*;

- μ is a *membrane structure*, with the membranes labelled by numbers $0, \ldots, d$ in a one-to-one manner;

- $e_0, \ldots, e_d$ are the initial energy values assigned to the membranes $0, \ldots, d$. In what follows we assume that $e_0, \ldots, e_d$ are non-negative integers;

- $w_0, \ldots, w_d$ are multisets over A associated with the regions $0, \ldots, d$ of μ;
- $R_0, \ldots, R_d$ are finite sets of *unit rules* associated with the membranes $0, \ldots, d$. Each rule has the form $(\alpha : a, \Delta e, b)$, where $\alpha \in \{in, out\}$, $a, b \in A$, and $|\Delta e|$ is the amount of energy that — for $\Delta e \geq 0$ — is added to or — for $\Delta e < 0$ — is subtracted from e_i (the energy assigned to membrane i) by the application of the rule.

By writing $(\alpha_i : a, \Delta e, b)$ instead of $(\alpha : a, \Delta e, b) \in R_i$, we can specify only one set of rules R with

$$R = \{(\alpha_i : a, \Delta e, b) \ : \ (\alpha : a, \Delta e, b) \in R_i, 0 \leq i \leq d\}$$

The *initial configuration* of Π consists of $e_0, \ldots, e_d$ and $w_0, \ldots, w_d$. The transition from a configuration to another one is performed by non-deterministically choosing one rule from some R_i and applying it (observe that here we consider a sequential model of applying the rules instead of choosing rules in a maximally parallel way, as it is often required in P systems). Applying $(in_i : a, \Delta e, b)$ means that an object a (being in the membrane immediately outside of i) is changed into b while entering membrane i, thereby changing the energy value e_i of membrane i by Δe. On the other hand, the application of a rule $(out_i : a, \Delta e, b)$ changes object a into b while leaving membrane i, and changes the energy value e_i by Δe. The rules can be applied only if the amount e_i of energy assigned to membrane i fulfills the requirement $e_i + \Delta e \geq 0$. Moreover, we use some sort of local priorities: if there are two or more applicable rules in membrane i, then one of the rules with $\max |\Delta e|$ has to be used.

A sequence of transitions is called a *computation*; it is *successful* if and only if it halts. The *result* of a successful computation is considered to be the distribution of energies among the membranes (a non-halting computation does not produce a result). If we consider the energy distribution of the membrane structure as the input to be analysed, we obtain a model for accepting sets of (vectors of) non-negative integers.

The following result, proved in [10], establishes computational completeness for this model of P systems.

Proposition 2. *Every partial recursive function $f : \mathbf{N}^\alpha \to \mathbf{N}^\beta$ can be computed by a P system with unit rules and energy assigned to membranes with (at most)* $\max\{\alpha, \beta\} + 3$ *membranes.*

It is interesting to note that the proof of this proposition is obtained by simulating register machines. In the simulation, a P system is defined which contains one subsystem for each register of the simulated machine. The contents of the register is expressed as the energy value e_i assigned to the i-th subsystem. A single object is present in the system at every computation step, which stores the label of the instruction of P currently simulated. Increment instructions are simulated in two steps by using the rules $(in_i : p_j, 1, \widetilde{p_j})$ and $(out_i : \widetilde{p_j}, 0, p_k)$. Decrement instructions are also simulated in two steps, by using the rules $(in_i : p_j, 0, \widetilde{p_j})$ and $(out_i : \widetilde{p_j}, -1, p_k)$ or $(out_i : \widetilde{p_j}, 0, p_l)$. The use of priorities associated to these last rules is crucial to correctly simulate a decrement instruction. For the details of the proof we refer the reader to [10].

On the other hand, by omitting the priority feature we do not get systems with universal computational power. Precisely, in [10] it is proved that P systems with unit rules and energy assigned to membranes without priorities and with an arbitrary number of membranes characterize the family $PsMAT^{\lambda}$ of Parikh sets generated by context–free matrix grammars (with λ-rules).

2.3 Quantum Computers

From an abstract point of view, a quantum computer can be considered as made up of interacting parts. The elementary units (memory cells) that compose these parts are two–levels quantum systems called *qubits*. A qubit is typically implemented using the energy levels of a two–levels atom, or the two spin states of a spin–$\frac{1}{2}$ atomic nucleus, or a polarization photon. The mathematical description — independent of the practical realization — of a single qubit is based on the two–dimensional complex Hilbert space $\mathbb{C}^2$. The boolean truth values 0 and 1 are represented in this framework by the unit vectors of the canonical orthonormal basis, called the *computational basis* of $\mathbb{C}^2$:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad\qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Qubits are thus the quantum extension of the classical notion of bit, but whereas bits can only take two different values, 0 and 1, qubits are not confined to their two basis (also *pure*) states, $|0\rangle$ and $|1\rangle$, but can also exist in states which are coherent superpositions such as $\psi = c_0 |0\rangle + c_1 |1\rangle$, where c_0 and c_1 are complex numbers satisfying the condition $|c_0|^2 + |c_1|^2 = 1$. Performing a measurement of the state alters it. Indeed, performing a measurement on a qubit in the above superposition will return 0 with

probability $|c_0|^2$ and 1 with probability $|c_1|^2$; the state of the qubit after the measurement (*post–measurement state*) will be $|0\rangle$ or $|1\rangle$, depending on the outcome.

A *quantum register* of size n (also called an n–*register*) is mathematically described by the Hilbert space $\otimes^n\mathbb{C}^2 = \underbrace{\mathbb{C}^2 \otimes \ldots \otimes \mathbb{C}^2}_{n \text{ times}}$, representing a set of n qubits labelled by the index $i \in \{1, \ldots, n\}$. An n–*configuration* (also *pattern*) is a vector $|x_1\rangle \otimes \ldots \otimes |x_n\rangle \in \otimes^n\mathbb{C}^2$, usually written as $|x_1, \ldots, x_n\rangle$, considered as a quantum realization of the boolean tuple $(x_1, \ldots, x_n)$. Let us recall that the dimension of $\otimes^n\mathbb{C}^2$ is 2^n and that $\{|x_1, \ldots, x_n\rangle : x_i \in \{0, 1\}\}$ is an orthonormal basis of this space called the n–register *computational basis*.

Computations are performed as follows. Each qubit of a given n–register is prepared in some particular pure state ($|0\rangle$ or $|1\rangle$) in order to realize the required n–configuration $|x_1, \ldots, x_n\rangle$, quantum realization of an input boolean tuple of length n. Then, a linear operator $G : \otimes^n\mathbb{C}^2 \to \otimes^n\mathbb{C}^2$ is applied to the n–register. The application of G has the effect of transforming the n–configuration $|x_1, \ldots, x_n\rangle$ into a new n–configuration $G(|x_1, \ldots, x_n\rangle) = |y_1, \ldots, y_n\rangle$, which is the quantum realization of the output tuple of the computer. We interpret such modification as a computation step performed by the quantum computer. The action of the operator G on a superposition $\Phi = \sum c^{i_1 \ldots i_n} |x_{i_1}, \ldots, x_{i_n}\rangle$, expressed as a linear combination of the elements of the n–register basis, is obtained by linearity: $G(\Phi) = \sum c^{i_1 \ldots i_n} G(|x_{i_1}, \ldots, x_{i_n}\rangle)$. We recall that linear operators which act on n–registers can be represented as order 2^n square matrices of complex entries. Usually (but not in this paper) such operators, as well as the corresponding matrices, are required to be unitary. In particular, this implies that the implemented operations are logically reversible (an operation is *logically reversible* if its inputs can always be deduced from its outputs).

All these notions can be easily extended to quantum systems which have $d > 2$ pure states. In this setting, the d–valued versions of qubits are usually called *qudits* [15]. As it happens with qubits, a qudit is typically implemented using the energy levels of an atom or a nuclear spin. The mathematical description — independent of the practical realization — of a single qudit is based on the d–dimensional complex Hilbert space $\mathbb{C}^d$. In particular, the pure states $|0\rangle, \left|\frac{1}{d-1}\right\rangle, \left|\frac{2}{d-1}\right\rangle, \ldots, \left|\frac{d-2}{d-1}\right\rangle, |1\rangle$ are represented by the unit vectors of the canonical orthonormal basis, called the *computational*

basis of $\mathbb{C}^d$:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \left|\frac{1}{d-1}\right\rangle = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad \cdots, \quad \left|\frac{d-2}{d-1}\right\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

As before, a *quantum register* of size n can be defined as a collection of n qudits. It is mathematically described by the Hilbert space $\otimes^n\mathbb{C}^d$. An *n–configuration* is now a vector $|x_1\rangle \otimes \ldots \otimes |x_n\rangle \in \otimes^n\mathbb{C}^d$, simply written as $|x_1, \ldots, x_n\rangle$, for x_i running on $L_d = \left\{0, \frac{1}{d-1}, \frac{2}{d-1}, \ldots, \frac{d-2}{d-1}, 1\right\}$. An n–configuration can be viewed as the quantum realization of the "classical" tuple $(x_1, \ldots, x_n) \in L_d^n$. The dimension of $\otimes^n\mathbb{C}^d$ is d^n and the set $\{|x_1, \ldots, x_n\rangle : x_i \in L_d\}$ of all n–configurations is an orthonormal basis of this space, called the *n–register computational basis*. Notice that the set L_d can also be interpreted as a set of truth values, where 0 denotes falsity, 1 denotes truth and the other elements indicate different degrees of indefiniteness.

Let us now consider the set $\mathcal{E}_d = \left\{\varepsilon_0, \varepsilon_{\frac{1}{d-1}}, \varepsilon_{\frac{2}{d-1}}, \ldots, \varepsilon_{\frac{d-2}{d-1}}, \varepsilon_1\right\} \subseteq \mathbb{R}$ of real values; we can think to such quantities as energy values. To each element $v \in L_d$ (and hence to each object $|v\rangle \in A$) we associate the energy level ε_v; moreover, let us assume that the values of $\mathcal{E}_d$ are all positive, equispaced, and ordered according to the corresponding objects: $0 < \varepsilon_0 < \varepsilon_{\frac{1}{d-1}} < \cdots < \varepsilon_{\frac{d-2}{d-1}} < \varepsilon_1$. If we denote by $\Delta\varepsilon$ the gap between two adjacent energy levels then the following linear relation holds:

$$\varepsilon_k = \varepsilon_0 + \Delta\varepsilon\,(d-1)\,k \qquad\qquad \forall\, k \in L_d \tag{1}$$

Notice that it is not required that $\varepsilon_0 = \Delta\varepsilon$. As explained in [17], the values ε_k can be thought of as the energy eigenvalues of the infinite dimensional quantum harmonic oscillator truncated at the $(d-1)$-th excited level (see Figure 1),

whose Hamiltonian on $\mathbb{C}^d$ is:

$$H = \begin{bmatrix} \varepsilon_0 & 0 & \ldots & 0 \\ 0 & \varepsilon_0 + \Delta\varepsilon & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \varepsilon_0 + (d-1)\Delta\varepsilon \end{bmatrix} \tag{2}$$

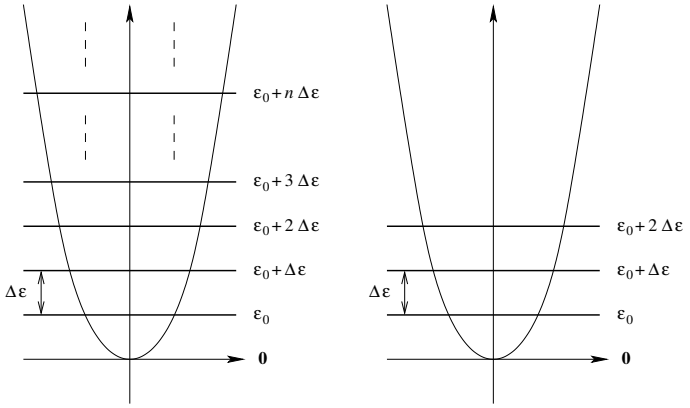


Figure 1: Energy levels of the infinite dimensional (on the left) and of the truncated (on the right) quantum harmonic oscillator

The unit vector $|H = \varepsilon_k\rangle = \left|\frac{k}{d-1}\right\rangle$, for $k \in \{0, 1, \ldots, d-1\}$, is the eigenvector of the state of energy $\varepsilon_0 + k\Delta\varepsilon$. To modify the state of a qudit we can use creation and annihilation operators on the Hilbert space $\mathbb{C}^d$, which are defined respectively as:

$$a^\dagger = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & \sqrt{2} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \sqrt{d-1} & 0 \end{bmatrix} \qquad a = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & \sqrt{2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \sqrt{d-1} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

It is easily verified that the action of $a^\dagger$ on the vectors of the canonical orthonormal basis of $\mathbb{C}^d$ is the following:

$$a^\dagger \left|\frac{k}{d-1}\right\rangle = \sqrt{k+1}\left|\frac{k+1}{d-1}\right\rangle \qquad \text{for } k \in \{0, 1, \ldots, d-2\}$$
$$a^\dagger |1\rangle = \mathbf{0}$$

whereas the action of a is:

$$a \left|\frac{k}{d-1}\right\rangle = \sqrt{k}\left|\frac{k-1}{d-1}\right\rangle \qquad \text{for } k \in \{1, 2, \ldots, d-1\}$$
$$a |0\rangle = \mathbf{0}$$

Using $a^\dagger$ and a we can also introduce the following operators:

$$N = a^\dagger a = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & d-1 \end{bmatrix} \qquad aa^\dagger = \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & d-1 & 0 \\ 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

The eigenvalues of the self–adjoint operator N are $0, 1, 2, \ldots, d-1$, and the eigenvector corresponding to the generic eigenvalue k is $|N = k\rangle = \left|\frac{k}{d-1}\right\rangle$. This corresponds to the notation adopted in [15], where the qudit base states are denoted by $|0\rangle, |1\rangle, \ldots, |d-1\rangle$, and it is assumed that a qudit can be in a superposition of the d base states:

$$c_0 |0\rangle + c_1 |1\rangle + \ldots + c_{d-1} |d-1\rangle$$

with $c_i \in \mathbb{C}$ for $i \in \{0, 1, \ldots, d-1\}$, and $|c_0|^2 + |c_1|^2 + \ldots + |c_{d-1}|^2 = 1$.

One possible physical interpretation of N is that it describes the *number of particles* of physical systems consisting of a maximum number of $d-1$ particles. In order to add a particle to the k particles state $|N = k\rangle$ (thus making it switch to the "next" state $|N = k+1\rangle$) we apply the creation operator $a^\dagger$, while to remove a particle from this system (thus making it switch to the "previous" state $|N = k-1\rangle$) we apply the annihilation operator a. Since the maximum number of particles that can be simultaneously in the system is $d-1$, the application of the creation operator to a full $d-1$ particles system does not have any effect on the system, and returns as a result the null vector. Analogously, the application of the annihilation operator to an empty particle system does not affect the system and returns the null vector as a result.

Another physical interpretation of operators $a^\dagger$ and a, by operator N, follows from the possibility of expressing the Hamiltonian (2) as follows:

$$H = \varepsilon_0 \,\mathrm{Id} + \Delta\varepsilon \,\mathrm{N} = \varepsilon_0 \,\mathrm{Id} + \Delta\varepsilon \, a^\dagger a$$

In this case $a^\dagger$ (resp., a) realizes the transition from the eigenstate of energy $\varepsilon_k = \varepsilon_0 + k\,\Delta\varepsilon$ to the "next" (resp., "previous") eigenstate of energy $\varepsilon_{k+1} = \varepsilon_0 + (k+1)\,\Delta\varepsilon$ (resp., $\varepsilon_{k-1} = \varepsilon_0 + (k-1)\,\Delta\varepsilon$) for any $0 \leq k < d-1$ (resp., $0 < k \leq d-1$), while it collapses the last excited (resp., ground) state of energy $\varepsilon_0 + (d-1)\,\Delta\varepsilon$ (resp., ε_0) to the null vector.

The collection of all linear operators on $\mathbb{C}^d$ is a d^2–dimensional linear space whose canonical basis is:

$$\{E_{x,y} = |y\rangle \langle x| \ : \ x, y \in L_d\}$$

Since $E_{x,y}|x\rangle = |y\rangle$ and $E_{x,y}|z\rangle = \mathbf{0}$ for every $z \in L_d$ such that $z \neq x$, this operator transforms the unit vector $|x\rangle$ into the unit vector $|y\rangle$, collapsing all the other vectors of the canonical orthonormal basis of $\mathbb{C}^d$ to the null vector. Each of the operators $E_{x,y}$ can be expressed, using the whole algebraic structure of the associative algebra of operators, as a suitable composition of creation and annihilation operators, as explained in [17].

3 Quantum P Systems with Unit Rules and Energy Assigned to Membranes

Let us now define a quantum version of P systems with unit rules and energy assigned to membranes. All the elements of the model (multisets, the membrane hierarchy, configurations, and computations) are defined just like the corresponding elements of the classical P system, but for objects and rules.

The objects of A are represented as pure states of a quantum system. If the alphabet contains $d \geq 2$ elements, then without loss of generality we can put $A = \left\{|0\rangle, \left|\frac{1}{d-1}\right\rangle, \left|\frac{2}{d-1}\right\rangle, \ldots, \left|\frac{d-2}{d-1}\right\rangle, |1\rangle\right\}$, that is, $A = \{|a\rangle \ : \ a \in L_d\}$. As stated above, the quantum system will also be able to assume as a state any superposition of the kind:

$$c_0|0\rangle + c_{\frac{1}{d-1}}\left|\frac{1}{d-1}\right\rangle + \ldots + c_{\frac{d-2}{d-1}}\left|\frac{d-2}{d-1}\right\rangle + c_1|1\rangle$$

with $c_0, c_{\frac{1}{d-1}}, \ldots, c_{\frac{d-2}{d-1}}, c_1 \in \mathbb{C}$ such that $\sum_{i=0}^{d-1}\left|c_{\frac{i}{d-1}}\right|^2 = 1$. A multiset is simply a collection of quantum systems, each in its own state.

The membrane structure is defined just like in the classical case. In order to represent the energy values assigned to membranes we must use quantum systems which can exist in an infinite (countable) number of states. Hence we assume that every membrane of the quantum P system has an associated infinite dimensional quantum harmonic oscillator whose state represents the energy value assigned to the membrane. To modify the state of such harmonic oscillator we can use the infinite dimensional version of creation ($a^\dagger$) and annihilation (a) operators described above, which are commonly used in quantum mechanics. The actions of $a^\dagger$ and a on the state of an infinite dimensional harmonic oscillator are analogous to the actions on the states of truncated harmonic oscillators; the only difference is that in the former case there is no state with maximum energy, and hence the creation operator never produces the null vector. Also in this case it is possible to express operators $E_{x,y} = |y\rangle\langle x|$ as appropriate compositions of $a^\dagger$ and a.

The initial configuration of a quantum P system with unit rules and energy assigned to membranes of degree $d+1$ consists of $e_0, \ldots, e_d$, the initial energy values assigned to the membranes, and $w_0, \ldots, w_d$, the multisets of objects initially present in the regions $0, \ldots, d$ determined by the membrane structure.

Rules are defined as (n, d)–functions, that is, functions of the kind $f : A^n \to A^n$. Such functions are not necessarily bijections on A^n: they can be arbitrary mappings. This means that the linear operators which realize such functions are not necessarily unitary. To write these linear operators we use an extension of the *Conditional Quantum Control* technique introduced in [2]. Such operators are sums of "local" operators, each being a tensor product of suitable compositions of the operators $a^\dagger$ and a. An equivalent formulation is possible, using spin–rising (J_+) and spin–lowering (J_-) operators, following the lines illustrated in [17].

The quantum realization of a "controlled behavior" can be obtained by making use of the operators $E_{X,X} = |X\rangle \langle X|$, for $X \in L_d$. For simplicity, let us first consider the case of a $(2, 2)$–function, that is, a two–input/two–output boolean function. For a reason that will be clear in a moment, we call *control qubit* and *target qubit* the first and the second input, respectively. If we want to realize a linear operator performing the condition: "if the control qubit is $|1\rangle$ then the operator O_1 is applied to the target qubit (and the control qubit is left unchanged)", then we can build the operator $E_{1,1} \otimes O_1$, where $E_{1,1} = |1\rangle \langle 1|$ checks for the condition "the control qubit is $|1\rangle$" and O_1 is the operator which acts on the target qubit $|x_2\rangle$. Note that if the control qubit is $|0\rangle$ then the operator $E_{1,1} \otimes O_1$ produces the null vector of $\mathbb{C}^2 \otimes \mathbb{C}^2$. Similarly, $E_{0,0} \otimes O_0$, with $E_{0,0} = |0\rangle \langle 0|$, realizes the condition "if the control qubit is $|0\rangle$ then the operator O_0 is applied to the target qubit $|x_2\rangle$ (and the control qubit is left unchanged)".

The same applies to (n, d)–functions, where the first k qudits are used as *control qudits* and the remaining $n - k$ are used as *target qudits*. We can thus realize any controlled behavior of the kind: "if the control qudits are in the (basis) states $X = |x_1, x_2, \ldots, x_k\rangle$, then apply the operator O_X to target qudits" (and leave the control qudits unaltered). The global operator that describes the behavior of the (n, d)–function has thus the form:

$$|0\rangle \langle 0| \otimes O_0 + |1\rangle \langle 1| \otimes O_1 + \ldots + |d^k - 1\rangle \langle d^k - 1| \otimes O_{d^k-1} = \sum_{X=0}^{d^k-1} |X\rangle \langle X| \otimes O_X$$

where $E_{X,X} = |X\rangle \langle X|$ is the orthogonal projection of the Hilbert space $\otimes^k \mathbb{C}^d$ which selects the X-th control configuration, and collapses to the null

vector all the other configurations.

We can now precisely describe how rules are defined in our model of quantum P systems. As in the classical case, rules are associated to membranes rather than to the regions enclosed by them. Each rule of R_i is an operator of the form

$$|y\rangle \langle x| \otimes O, \qquad \text{with } x, y \in L_d \tag{3}$$

where O is a linear operator which can be expressed by an appropriate composition of operators $a^\dagger$ and a. The part $|y\rangle \langle x|$ is the *guard* of the rule: it makes the rule "active" (that is, the rule produces an effect) if and only if a quantum system in the basis state $|x\rangle$ is present. The semantics of rule (3) is the following: If an object in state $|x\rangle$ is present in the region immediately outside membrane i, then the state of the object is changed to $|y\rangle$ and the operator O is applied to the state of the infinite dimensional harmonic oscillator associated with the membrane. Notice that the application of O can result in the null vector, so that the rule has no effect even if its guard is satisfied; this fact is equivalent to the condition $e_i + \Delta e \geq 0$ on the energy of membrane i required in the classical case. Differently from the classical case, no local priorities are assigned to the rules. If two or more rules are associated to membrane i, then they are summed. This means that, indeed, we can think to each membrane as having only one rule with many guards. When an object is present, the inactive parts of the rule (those for which the guard is not satisfied) produce the null operator as a result. If the region in which the object occurs contains two or more membranes, then all their rules are applied to the object. Observe that the object which activates the rules never crosses the membranes. This means that the objects specified in the initial configuration can change their state but never move to a different region. Notwithstanding, transmission of information between different membranes is possible, since different objects may modify in different ways the energy state of the harmonic oscillators associated with the membranes.

The application of one or more rules determines a *transition* between two configurations. A *halting configuration* is a configuration in which no rule can be applied. A sequence of transitions is a *computation*. A computation is *successful* if and only if it *halts*, that is, reaches a halting configuration. The *result* of a successful computation is considered to be the distribution of energies among the membranes in the halting configuration. A non-halting computation does not produce a result. Just like in the classical case, if we consider the energy distribution of the membrane structure as the input

to be analyzed, we obtain a model for accepting sets of (vectors of) non-negative integers.

4 Computational Completeness

In this section we prove that quantum P systems with unit rules and energy assigned to membranes are computationally complete, that is, they are able to compute any partial recursive function $f : \mathbf{N}^\alpha \to \mathbf{N}^\beta$. As in the classical case, the proof is obtained by simulating register machines.

Theorem 1. *Every partial recursive function $f : \mathbf{N}^\alpha \to \mathbf{N}^\beta$ can be computed by a quantum P system with unit rules and energy assigned to membranes with (at most)* $\max\{\alpha, \beta\} + 3$ *membranes.*

Proof. Let $M = (n, P, 1, m)$ be a deterministic n–register machine that computes f. Let m be the number of instructions of P. The initial instruction of P has the label 1, and the halting instruction has the label m. Observe that, according to Proposition 1, $n = \max\{\alpha, \beta\} + 2$ is enough.

The input values $x_1, \ldots, x_\alpha$ are expected to be in the first α registers, and the output values are expected to be in registers 1 to β at the end of a successful computation. Moreover, without loss of generality, we may assume that at the beginning of a computation all the registers except (eventually) the registers 1 to α contain zero.

We construct the quantum P system

$$\Pi = (A, \mu, e_0, \ldots, e_n, w_0, \ldots, w_n, R_0, \ldots, R_n)$$

where:

- $A = \{|j\rangle \, : \, j \in L_m\}$
- $\mu = [_0[_1]_1 \cdots [_\alpha]_\alpha \cdots [_n]_n]_0$
- $e_i = \begin{cases} |\varepsilon_{x_i}\rangle & \text{for } 1 \leq i \leq \alpha \\ |\varepsilon_0\rangle & \text{for } \alpha + 1 \leq i \leq n \\ \mathbf{0} & \text{(the null vector) for } i = 0 \end{cases}$
- $w_0 = |0\rangle$
- $w_i = \emptyset \qquad \text{for } 1 \leq i \leq n$
- $R_0 = \emptyset$

- $R_i = \sum_{j=1}^{m} O_{i_j}$ for $1 \leq i \leq n$

  where the O_{i_j}'s are local operators which simulate instructions of the kind $j : (INC(i), k)$ and $j : (DEC(i), k, l)$ (one local operator for each increment or decrement operation which affects register i). The details on how the O_{i_j}'s are defined are given below.

The value contained in register i, $1 \leq i \leq n$, is represented by the energy value $e_i = |\varepsilon_{x_i}\rangle$ of the infinite dimensional quantum harmonic oscillator associated with membrane i. Figure 2 depicts a typical configuration of Π.

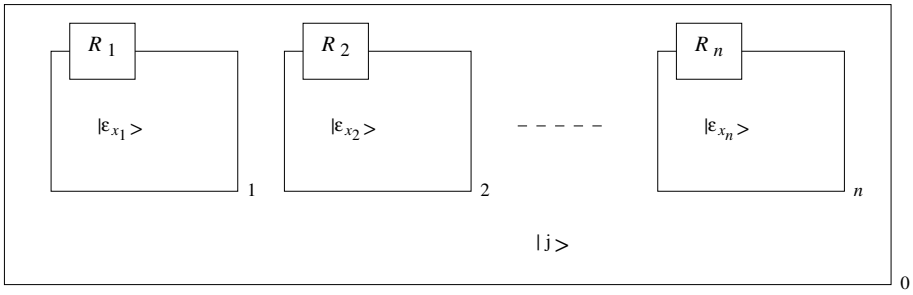


Figure 2: A configuration of the simulating P system

The skin contains one object of the kind $|j\rangle$, $j \in L_m$, which mimics the program counter of machine M. Precisely, if the program counter of M has the value $k \in \{1, 2, \ldots, m\}$ then the object present in region 0 is $\left|\frac{k-1}{m-1}\right\rangle$. In order to avoid cumbersome notation, in what follows we denote by $|p_k\rangle$ the state $\left|\frac{k-1}{m-1}\right\rangle$ of the quantum system which mimics the program counter.

The sets of rules R_i depend upon the instructions of P. Precisely, the simulation works as follows.

1. Increment instructions $j : (INC(i), k)$ are simulated by a guarded rule of the kind $|p_k\rangle \langle p_j| \otimes a^\dagger \in R_i$.

   If the object $|p_j\rangle$ is present in region 0, then the rule transforms it into object $|p_k\rangle$ and increments the energy level of the harmonic oscillator contained in membrane i.

2. Decrement instructions $j : (DEC(i), k, l)$ are simulated by a guarded rule of the kind:

$$|p_l\rangle \langle p_j| \otimes |\varepsilon_0\rangle \langle \varepsilon_0| + |p_k\rangle \langle p_j| \otimes a \in R_i$$

In fact, let us assume that the object $|p_j\rangle$ is present in region 0 (if $|p_j\rangle$ is not present then the above rule produces the null operator), and let us denote by O the above rule. The harmonic oscillator may be in the base state $|\varepsilon_0\rangle$ or in a base state $|\varepsilon_x\rangle$ with x a positive integer.

If the state of the harmonic oscillator is $|\varepsilon_0\rangle$ then the rule produces:

$$\begin{aligned} &O(\,|p_j\rangle \otimes |\varepsilon_0\rangle) = \\ &\quad = (|p_l\rangle\,\langle p_j| \otimes |\varepsilon_0\rangle\,\langle\varepsilon_0|)(|p_j\rangle \otimes |\varepsilon_0\rangle) + (|p_k\rangle\,\langle p_j| \otimes a)(|p_j\rangle \otimes |\varepsilon_0\rangle) = \\ &\quad = |p_l\rangle \otimes |\varepsilon_0\rangle + |p_k\rangle \otimes \mathbf{0} = |p_l\rangle \otimes |\varepsilon_0\rangle \end{aligned}$$

that is, the state of the oscillator is unaltered and the program counter is set to $|p_l\rangle$.

If the state of the harmonic oscillator is $|\varepsilon_x\rangle$, for a positive integer x, then the rule produces:

$$\begin{aligned} &O(\,|p_j\rangle \otimes |\varepsilon_x\rangle) = \\ &\quad = (|p_l\rangle\,\langle p_j| \otimes |\varepsilon_0\rangle\,\langle\varepsilon_0|)(|p_j\rangle \otimes |\varepsilon_x\rangle) + (|p_k\rangle\,\langle p_j| \otimes a)(|p_j\rangle \otimes |\varepsilon_x\rangle) = \\ &\quad = |p_l\rangle \otimes \mathbf{0} + |p_k\rangle \otimes a\,|\varepsilon_x\rangle = |p_k\rangle \otimes |\varepsilon_{x-1}\rangle \end{aligned}$$

that is, the energy level of the harmonic oscillator is decremented and the program counter is set to $|p_k\rangle$.

The set R_i of rules is obtained by summing all the operators which affect (increment or decrement) register i. The Halt instruction is simply simulated by doing nothing with the object $|p_m\rangle$ when it appears in region 0.

It is apparent from the description given above that after the simulation of each instruction each energy value e_i equals the value contained in register i, with $1 \leq i \leq m$. Hence, when the halting symbol $|p_m\rangle$ appears in region 0, the energy values $e_1, \ldots, e_\beta$ equal the output of the program P. □

Let us conclude this section by observing that, in order to obtain computational completeness, it is not necessary that the objects cross the membranes. This fact avoids one of the problems raised in [17]: the existence of a "magic" quantum transportation mechanism which is able to move objects according to the target contained in the rule (and working against the so called "tunnel effect"). In quantum P systems with unit rules and energy assigned to membranes, the only problem is to keep the object $|p_j\rangle$ localised in region 0, so that it never enters into the other regions. In other words, the major problem of this kind of quantum P systems is to oppose the tunnel effect.

5 Quantum Register Machines

The P system illustrated in Figure 2, which has been used to prove Theorem 1, suggests to define also a quantum version of register machines.

A *quantum n–register machine* is defined exactly as in the classical case, as a four–tuple $M = (n, P, l_0, l_h)$. For simplicity, also the instructions of P are denoted in the usual way:

$$j : (INC(i), k) \qquad \text{and} \qquad j : (DEC(i), k, l)$$

This time, however, these instructions are appropriate linear operators acting on the Hilbert space whose vectors describe the (global) state of M.

The structure of the machine resembles the P system which has been used to prove Theorem 1. Each register of the machine is an infinite dimensional quantum harmonic oscillator, capable to assume the base states $|\varepsilon_0\rangle, |\varepsilon_1\rangle, |\varepsilon_2\rangle, \ldots$, corresponding to its energy levels. The program counter of the machine is instead realized through a quantum system capable to assume m different base states, from the set $\{|x\rangle : x \in L_m\}$.

A *configuration* of M is given by the value of the program counter and the values contained in the registers. From a mathematical point of view, a configuration of M is a (base) vector of the Hilbert space $\mathbb{C}^m \otimes (\otimes^n \mathcal{H})$, where $\mathcal{H}$ is the Hilbert space associated with every quantum harmonic oscillator. Notice that here we are just interested in simulating a classical machine behavior, and hence we do not care about superpositions of states. A transition between two configurations is obtained by executing one instruction of P (the one pointed at by the program counter).

The instruction $j : (INC(r), k)$ is defined as operator

$$O^{INC}_{j,r,k} = |p_k\rangle \langle p_j| \otimes \left(\otimes^{r-1}\mathbb{I}\right) \otimes a^\dagger \otimes \left(\otimes^{n-r}\mathbb{I}\right)$$

with $\mathbb{I}$ the identity operator on $\mathcal{H}$, whereas the instruction $j : (DEC(r), k, l)$ is defined as the operator

$$\begin{aligned} O^{DEC}_{j,r,k,l} = |p_l\rangle \langle p_j| \otimes \left(\otimes^{r-1}\mathbb{I}\right) \otimes |\varepsilon_0\rangle \langle \varepsilon_0| \otimes \left(\otimes^{n-r}\mathbb{I}\right) + \\ |p_k\rangle \langle p_j| \otimes \left(\otimes^{r-1}\mathbb{I}\right) \otimes a \otimes \left(\otimes^{n-r}\mathbb{I}\right) \end{aligned}$$

Hence the program P can be formally defined as the sum O_P of all these operators:

$$O_P = \sum_{j,r,k} O^{INC}_{j,r,k} + \sum_{j,r,k,l} O^{DEC}_{j,r,k,l}$$

Thus O_P is the global operator which describes a computation step of M. The Halt instruction is simply executed by doing nothing when the program counter assumes the value $|p_m\rangle$. For such value, O_P becomes the null operator. Hence M halts after T execution steps if $T = \min\{t \in \mathbf{N} \,:\, O_P^t = \mathbf{0}\}$.

From the definition of the system, it is apparent that any classical deterministic n–register machine can be simulated by a corresponding quantum n–register machine: the simulation proceeds exactly as described in the proof of Theorem 1. As a consequence, also quantum n–register machines are computationally complete.

It should also be evident that the proof of Theorem 1 can be modified to show that quantum P systems are able to simulate quantum register machines. Indeed, the notable difference between the quantum P systems described above and quantum register machines is that in the latter model we modify the values contained in registers using *global* operators (if a given register need not be modified then the identity operator is applied to its state) whereas in the former model we operate locally, on a smaller Hilbert space. Hence, as it happens in classical P systems, membranes are used to divide the site where the computation occurs into independent local areas. The effect of each rule is local, in the sense that the rule affects only the state of one subsystem. Due to the simulations mentioned above, we can order these computational models with respect to their computational power, as follows:

deterministic register machines	$\leq$	quantum register machines	$\leq$	quantum P systems with unit rules and energy assigned to membranes

Quantum register machines can thus be used as a tool to study the computational power of other quantum models of computation, just like it happens in the classical case.

6 Conclusions and Directions for Future Research

In this paper we have introduced a quantum version of P systems with unit rules and energy assigned to membranes. Objects are represented as pure states in a finite Hilbert space, whereas rules are defined as generic functions which map the alphabet into itself. Such functions are implemented using a generalization of the Conditional Quantum Control technique, and may yield non-unitary operators. Energy values are associated to membranes by incorporating an infinite dimensional quantum harmonic oscillator in every

membrane. For the application of rules leading from one configuration of the system to the next configuration we consider a sequential model, instead of the usual model of maximal parallelism. The input of a computation is given by the distribution of energy values carried by the membranes. Analogously, the result of a successful computation is the distribution of energy values at the end of the computation.

In this paper we have proved that such quantum model of computation is computationally complete, that is, it is able to compute any partial recursive function $f : \mathbf{N}^\alpha \to \mathbf{N}^\beta$. This result has been obtained by simulating classical deterministic register machines. We have also proposed quantum register machines as a tool to study the computational power of present and future quantum computational models.

It is currently an open problem, as well as an interesting direction for future research, to precisely assess the computational power of quantum P systems and quantum register machines. Concerning the power of quantum P systems we note that, in analogy with other models of quantum computers, there is the possibility to initialize the system with a multiset of objects whose state is a superposition of classical (pure) states. As a result, the computation will transform such input multiset to an output multiset which is obtained by linearity as a superposition of the results of the computation on every single classical state. As usual, when we measure the state of the systems which occur into the output multiset we will obtain a pure state as a result, according to the probability distribution which is induced by the coefficients of the superposition. An interesting question, not afforded in this paper, is whether the measurement of the state of an object into a region should have only *local* effects, or instead make the global configuration of the P system collapse to a classical state. Another interesting aspect of quantum P systems to be investigated is their behavior when some quantum systems in the initial configuration are in an entangled state.

References

[1] G. Alford: Membrane systems with heat control. In *Pre–Proceedings of the Workshop on Membrane Computing*, Curtea de Argeş, Romania (2002).

[2] A. Barenco, D. Deutsch, A. Ekert, R. Jozsa: Conditional Quantum Control and Logic Gates: *Physical Review Letters* **74** (1995), 4083–4086.

[3] P. Benioff: Quantum Mechanical Hamiltonian Models of Discrete Processes. *Journal of Mathematical Physics* **22** (1981), 495–507.

[4] P. Benioff: Quantum Mechanical Hamiltonian Models of Computers. *Annals of the New York Academy of Science* **480** (1986), 475–486.

[5] D. Deutsch: Quantum Theory, the Church–Turing Principle, and the Universal Quantum Computer. *Proceedings of the Royal Society of London* **A 400** (1985), 97–117.

[6] R. P. Feynman: Simulating Physics with Computers. *International Journal of Theoretical Physics* **21** (6–7) (1982), 467–488.

[7] R. P. Feynman: Quantum Mechanical Computers. *Optics News* **11** (1985), 11–20.

[8] R. Freund: Sequential P-systems. *Romanian Journal of Information Science and Technology* **4** (1–2) (2001), 77–88.

[9] R. Freund: Energy-controlled P systems. In: Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing. International Workshop, WMC-CdeA 2002, Curtea de Argeş, Romania, August (2002), *Lecture Notes in Computer Science* **2597**, Springer–Verlag, Berlin (2003), 247–260.

[10] R. Freund, A. Leporati, M. Oswald, C. Zandron: Sequential P Systems with Unit Rules and Energy Assigned to Membranes. In: Proceedings of Machines, Computations and Universality, MCU 2004, Saint–Petersburg, Russia, September 21–24 (2004), *Lecture Notes in Computer Science* **3354**, Springer-Verlag, Berlin (2005), 200–210.

[11] R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae* **49** (1–3) (2002), 81–102.

[12] R. Freund, Gh. Păun: On the Number of Non-terminals in Graph-controlled, Programmed, and Matrix Grammars. In: Margenstern, M., Rogozhin, Y. (Eds.): Proc. Conf. Universal Machines and Computations, Chişinău (2001). *Lectured Notes in Computer Science* **2055**, Springer–Verlag (2001), 214–225.

[13] R. Freund, Gh. Păun: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *Theoretical Computer Science* **312** (2004), 143–188.

[14] P. Frisco: The conformon–P system: a molecular and cell biology–inspired computability model. *Theoretical Computer Science* **312** (2004), 295–319.

[15] D. Gottesman: Fault–tolerant quantum computation with higher–dimensional systems. *Chaos, Solitons, and Fractals* **10** (1999), 1749–1758.

[16] J. Gruska: *Quantum Computing*. McGraw–Hill (1999).

[17] A. Leporati, D. Pescini, C. Zandron: Quantum Energy–based P Systems. In *Proceedings of the First Brainstorming Workshop on Uncertainty in Membrane Computing*, Palma de Mallorca, Spain, November 8–10 (2004), 145–167.

[18] A. Leporati, C. Zandron, G. Mauri: Simulating the Fredkin Gate with Energy–Based P Systems. *Journal of Universal Computer Science* **10** (5) (2004), 600–619. A preliminary version is contained in [23], 292–308.

[19] A. Leporati, C. Zandron, G. Mauri: Universal Families of Reversible P Systems. In: *Proceedings of Machines, Computations and Universality, MCU 2004*, Saint–Petersburg, Russia, September 21–24 (2004), *Lecture Notes in Computer Science* **3354**, Springer-Verlag, Berlin (2005), 257–268.

[20] M. L. Minsky: *Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey (1967).

[21] Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, **1** (61) (2000), 108–143. Also in: *Turku Centre for Computer Science – TUCS* Report No. 208 (1998).

[22] Gh. Păun: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin (2002).

[23] Gh. Păun, A. Riscos Nuñez, A. Romero Jiménez, F. Sancho Caparrini (Eds.): *Second Brainstorming Week on Membrane Computing*, Seville, Spain, February 2–7 (2004). Department of Computer Sciences and Artificial Intelligence, University of Seville TR (01/2004).

[24] Gh. Păun, Y. Suzuki, H. Tanaka: P Systems with energy accounting. *International Journal Computer Math.* **78** (3) (2001), 343–364.

[25] The P systems Web page: `http://psystems.disco.unimib.it/`

Editing Distances Between Membrane Structures*

Damián LÓPEZ, José M. SEMPERE

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
46071 Valencia (SPAIN)
E-mail: {dlopez,jsempere}@dsic.upv.es

Abstract

In this work we propose an efficient solution to calculate the minimum editing distance between membrane structures of arbitrary P systems. We use a new model of tree automata based on multisets of states and symbols linked to the finite control. This new model accepts a set of trees with symmetries between their internal nodes (*mirrored trees*). Once we have calculated the editing distance between an arbitrary tree and an arbitrary multiset tree automaton, we can translate the classical operations of insertion, deletion and substitution into rule applications of membrane dissolving and membrane creation.

1 Introduction

One of the main components of P systems is the membrane structure. This structure evolves during the computation time due to the application of rules associated to the membranes. The membrane structure can be represented by a tree in which the internal nodes denote regions which have inner regions inside. The root of the tree is always associated to the skin membrane of the P system.

The relation between regions and trees has been recently strengthened by Freund *et al.* [7]. These authors have established that any recursively enumerable set of trees can be generated by a P system with active membranes and string objects. So, P systems can be viewed as tree generators.

*Work supported by the Spanish CICYT under contract TIC2003-09319-C03-02.

In this work we use multiset tree automata to accept and handle the tree structures defined by P systems [16]. This model is an extension of classical tree automata [8] in which the states and symbols of the finite control form *multisets*. Multiset theory has been linked to parallel processing as showed in [2].

The main aspect we will solve in this work is the one related to *editing structural configurations* of P systems. Recently, Csuhaj-Varjú *et al.* [4] have proposed editing distances between configurations of P systems. Here, we restrict our solution only to the structural configuration of P systems, that is the membrane structure underlying any P system configuration. The multiset tree automata model that we propose in this work will be useful to calculate so. Here we can take advantage of a previous work on editing distances between trees and tree automata [10].

The structure of this work is as follows: First we introduce basic definitions and notation about multisets, tree languages and automata and P systems. In section 3, we introduce the model of multiset tree automata, we define the relation of *mirroring* between trees and we establish some results between tree automata, multiset tree automata and mirroring trees. In section 4, we use previous results about editing distances between trees and tree automata in order to solve the minimum editing distance between membrane structures. Finally, we establish some conclusions and give some guidelines for future works.

2 Notation and Definitions

In the sequel we will provide some concepts from formal language theory, membrane systems and multiset processing. We suggest the following books to the reader [15], [12] and [2].

Multisets

First, we will provide some definitions from multiset theory as exposed in [17].

Definition 2.1 *Let D be a set. A multiset over D is a pair $\langle D, f\rangle$ where $f : D \longrightarrow \mathbb{N}$ is a function.*

Definition 2.2 *Suppose that $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ are two multisets. The removal of multiset B from A, denoted by $A \ominus B$, is the multiset $C = \langle D, h\rangle$ where for all $a \in D$ $h(a) = max(f(a) - g(a), 0)$.*

Definition 2.3 *Let $A = \langle D, f\rangle$ be a multiset; we will say that A is* empty *if for all $a \in D$, $f(a) = 0$.*

Definition 2.4 *Let $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ be two multisets. Their sum, denoted by $A \oplus B$, is the multiset $C = \langle D, h\rangle$, where for all $a \in D$ $h(a) = f(a) + g(a)$.*

Definition 2.5 *Let $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ be two multisets. We will say that $A = B$ if for all $a \in D$ $f(a) = g(a)$.*

The number of elements that a multiset contains can be finite. In such case, the multiset will be finite too. The size of any multiset M, denoted by $|M|$ will be the number of elements that it contains. We are specially interested in the class of multisets that we call *bounded multisets*. They are multisets that hold the property that the sum of all the elements is bounded by a constant n. Formally, we will denote by $\mathcal{M}_n(D)$ the set of all multisets $\langle D, f\rangle$ such that $\sum_{a\in D} f(a) = n$.

A concept that is quite useful to work with sets and multisets is the *Parikh mapping*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \to \mathbb{N}^n$ where $D = \{d_1, d_2, \cdots, d_n\}$ and D^* is the set of strings defined by D. Given an element $x \in D^*$ we define $\Psi(x) = (\#_{d_1}(x), \cdots, \#_{d_n}(x))$ where $\#_{d_j}(x)$ denotes the number of occurrences of d_j in x.

Later, we will use tuples of symbols and states as strings and we will apply the Parikh mapping as defined above.

Tree Automata and Tree Languages

Now, we will introduce some concepts from tree languages and automata as exposed in [3, 8]. First, a *ranked alphabet* is the pair (V, r) where V is an alphabet and r is a finite relation in $V \times \mathbb{N}$. We denote by V_n the subset $\{\sigma \in V : (\sigma, n) \in r\}$. Given (V, r) we define $maxarity(V)$ as the maximum integer n such that$(\sigma, n) \in r$.

For every ranked alphabet (V, r), the set of trees over V, is denoted by V^T and defined inductively as follows:

$a \in V^T$ for every $a \in V_0$
$\sigma(t_1, ..., t_n) \in V^T$ whenever $\sigma \in V_n$ and $t_1, ..., t_n \in V^T$, $(n > 0)$

and let a *tree language* over V be defined as a subset of V^T.

Given the tuple $l =< 1, 2, ..., k >$ we will denote the set of permutations of l by $perm(l)$. Let $t = \sigma(t_1, ..., t_n)$ be a tree over V^T, we will denote the

set of permutations of t at first level by $perm_1(t)$. Formally, $perm_1(t) = \{\sigma(t_{i_1}, ..., t_{i_n}) :< i_1, i_2, ..., i_n >\in perm(< 1, 2, ..., n >)\}$.

Let $\mathbb{N}^*$ be the set of finite strings of natural numbers, separated by dots, formed using the product as the composition rule and the empty word λ as the identity. Let the prefix relation $\leq$ in $\mathbb{N}^*$ be defined by the condition that $u \leq v$ if and only if $u \cdot w = v$ for some $w \in \mathbb{N}^*$ $(u, v \in \mathbb{N}^*)$. A finite subset D of $\mathbb{N}^*$ is called a *tree domain* if:

$$u \leq v \text{ where } v \in D \text{ implies } u \in D, \text{ and}$$
$$u \cdot i \in D \text{ whenever } u \cdot j \in D \ (1 \leq i \leq j)$$

Each tree domain D could be seen as an unlabelled tree whose nodes correspond to the elements of D where the hierarchy relation is the prefix order. Thus, each tree t over V can be seen as an application $t : D \to V$. The set D is called the *domain of the tree* t, and denoted by $dom(t)$. The elements of the tree domain $dom(t)$ are called *positions* or *nodes* of the tree t. We denote by $t(x)$ the label of a given node x in $dom(t)$.

Let the level of $x \in dom(t)$ be denoted by $level(x)$. Intuitively, the level of a node measures its distance from the root of the tree. Then, we can define the depth of a tree t as $depth(t) = max\{level(x) : x \in dom(t)\}$. In the same way, for any tree t, we denote the size of the tree by $|t|$ and the set of subtrees of t (denoted with $Sub(t)$) as follows:

$$Sub(a) = \{a\} \text{ for all } a \in V_0$$
$$Sub(t) = \{t\} \cup \bigcup_{i=1,...,n} Sub(t_i) \text{ for } t = \sigma(t_1, ..., t_n) \ (n > 0)$$

For any set of trees T, $Sub(T) = \bigcup_{t \in T} Sub(t)$. Given a tree $t = \sigma(t_1, \ldots, t_n)$, the root of t will be denoted as $root(t)$ and defined as $root(t) = \sigma$. If $t = a$ then $root(t) = a$. The successors of a tree $t = \sigma(t_1, \ldots, t_n)$ will be defined as $H^t =< root(t_1), \ldots, root(t_n) >$.

Definition 2.6 *A* finite deterministic tree automaton *is defined by the tuple* $A = (Q, V, \delta, F)$*: where* Q *is a finite set of states;* V *is a ranked alphabet,* $Q \cap V = \emptyset$*;* $F \subseteq Q$ *is a set of final states and* $\delta = \bigcup_{i:V_i \neq \emptyset} \delta_i$ *is a set of transition functions defined as follows:*

$$\delta_n : (V_n \times (Q \cup V_0)^n) \to Q \qquad n > 0$$
$$\delta_0(a) = a \qquad \forall a \in V_0$$

Given the state $q \in Q$, we define the *ancestors* of the state q, denoted by $Anc(q)$, as the set of strings

$$Anc(q) = \{p_1 \cdots p_n : p_i \in Q \cup V_0 \wedge \delta_n(\sigma, p_1, ..., p_n) = q \in \delta\}$$

From now on, we will refer to finite deterministic tree automata simply as *tree automata*. We suggest [3, 8] for other definitions on tree automata.

The transition function δ is extended to a function $\delta : V^T \to Q \cup V_0$ on trees as follows:

$$\begin{aligned}
&\delta(a) = a \text{ for any } a \in V_0 \\
&\delta(t) = \delta_n(\sigma, \delta(t_1), \ldots, \delta(t_n)) \text{ for } t = \sigma(t_1, \ldots, t_n) \ (n > 0)
\end{aligned}$$

Note that the symbol δ denotes both the set of transition functions of the automaton and the extension of these functions to operate on trees. In addition, you can observe that the tree automaton A cannot accept any tree of depth zero.

Given a finite set of trees T, let the *subtree automaton* for T be defined as $AB_T = (Q, V, \delta, F)$, where:

$$\begin{aligned}
&Q = Sub(T) \\
&F = T \\
&\delta_n(\sigma, u_1, \ldots, u_n) = \sigma(u_1, \ldots, u_n) && \sigma(u_1, \ldots, u_n) \in Q \\
&\delta_0(a) = a && a \in V_0
\end{aligned}$$

P Systems

Finally, we will introduce basic concepts from the theory of membrane systems taken from [12]. A general P system of degree m is a construct

$$\Pi = (V, T, C, \mu, w_1, \cdots, w_m, (R_1, \rho_1), \cdots, (R_m, \rho_m), i_0),$$

where:

- V is an alphabet (the *objects*)
- $T \subseteq V$ (the *output alphabet*)
- $C \subseteq V$, $C \cap T = \emptyset$ (the *catalysts*)
- μ is a membrane structure consisting of m membranes

- w_i, $1 \leq i \leq m$ is a string representing a multiset over V associated with the region i
- R_i, $1 \leq i \leq m$ is a finite set of *evolution rules* over V associated with the ith region and ρ_i is a partial order relation over R_i specifying a *priority*.

  An evolution rule is a pair (u, v) (or $u \rightarrow v$) where u is a string over V and $v = v'$ or $v = v'\delta$ where v' is a string over

  $$\{a_{here}, a_{out}, a_{in_j} \mid a \in V, 1 \leq j \leq m\}$$

  and δ is an special symbol not in V (it defines the *membrane dissolving action*).
- i_0 is a number between 1 and m and it specifies the *output* membrane of Π (in the case that it equals to ∞ the output is read outside the system).

The language generated by Π in external mode ($i_0 = \infty$) is denoted by $L(\Pi)$ and it is defined as the set of strings that can be defined by collecting the objects that leave the system by arranging the leaving order (if several objects leave the system at the same time then permutations are allowed). The set of numbers that represent the objects in the output membrane i_0 will be denote by $N(\Pi)$. Obviously, both sets $L(\Pi)$ and $N(\Pi)$ are defined only for *halting computations*.

Some kinds of P systems which have been proposed focus on the creation, division and modification of membrane structures. There have been several works in which these operations have been proposed (see, for example, [1, 11, 12, 13]).

In the following, we enumerate some kinds of rules which are able to modify the membrane structure:

1. 2-division: $[_h a]_h \rightarrow [_{h'} b]_{h'} [_{h''} c]_{h''}$
2. Creation: $a \rightarrow [_h b]_h$
3. Dissolving: $[_h a]_h \rightarrow b$

The power of P systems with the previous operations and other ones (e.g. *exocytosis, endocytosis*, etc.) has been widely studied in the previously related works and other ones.

3 Multiset Tree Automata and Mirrored Trees

We will extend some definitions of tree automata and tree languages over multisets. We will introduce the concept of multiset tree automata and then we will characterize the set of trees that they accept, as exposed in [16]. Observe that our approach is different from Csuhaj-Varjú *et al.* [5] and from Kudlek *et al.* [9] where the authors consider the case that *bags of objects* are analyzed by an abstract machine. Here, we do not consider *bags of (sub)trees* but we introduce bags of states and symbols in the finite control of the automata.

Given any tree automaton $A = (Q, V, \delta, F)$ and $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, we can associate to δ_n the multiset $\langle Q \cup V_0, f\rangle \in \mathcal{M}_n(Q \cup V_0)$ where f is defined by $\Psi(p_1 p_2 \ldots p_n)$. The multiset defined in such way will be denoted by $M_\Psi(\delta_n)$. Alternatively, we can define $M_\Psi(\delta_n)$ as $M_\Psi(p_1) \oplus M_\Psi(p_2) \oplus \cdots \oplus M_\Psi(p_n)$ where $\forall 1 \leq i \leq n\ M_\Psi(p_i) \in \mathcal{M}_1(Q \cup V_0)$. Observe that if $\delta_n(\sigma, p_1, p_2, \ldots, p_n) \in \delta$, $\delta'_n(\sigma, p'_1, p'_2, \ldots, p'_n) \in \delta$ and $M_\Psi(\delta_n) = M_\Psi(\delta'_n)$ then δ_n and δ'_n are defined over the same set of states and symbols but in different order (that is the multiset induced by $\langle p_1 p_2 \cdots p_n\rangle$ equals to the one induced by $\langle p'_1 p'_2 \ldots p'_n\rangle$).

Now, we can define a *multiset tree automaton* that performs a bottom-up parsing as in the tree automaton case.

Definition 3.1 *A* multiset tree automaton *is defined by the tuple* $MA = (Q, V, \delta, F)$*, where* Q *is a finite set of states,* V *is a ranked alphabet with* $maxarity(V) = n$*,* $Q \cap V = \emptyset$*,* $F \subseteq Q$ *is a set of final states and* δ *is a set of transition functions defined as follows:*

$$\delta = \bigcup_{\substack{1 \leq i \leq n \\ i\,:\,V_i \neq \emptyset}} \delta_i$$

$$\begin{aligned} \delta_i : (V_i \times \mathcal{M}_i(Q \cup V_0)) &\to \mathcal{P}(\mathcal{M}_1(Q)) \qquad i = 1, \ldots, n \\ \delta_0(a) = M_\Psi(a) &\in \mathcal{M}_1(Q \cup V_0) \qquad \forall a \in V_0 \end{aligned}$$

We can take notice that every tree automaton A defines a multiset tree automaton MA as follows

Definition 3.2 *Let* $A = (Q, V, \delta, F)$ *be a tree automaton. The multiset tree automaton induced by* A *is defined by the tuple* $MA = (Q, V, \delta', F)$ *where each* δ' *is defined as follows:* $M_\Psi(r) \in \delta'_n(\sigma, M)$ *if* $\delta_n(\sigma, p_1, p_2, ..., p_n) = r$ *and* $M_\Psi(\delta_n) = M$*.*

Observe that, in the general case, the multiset tree automaton induced by A is non deterministic.

As in the case of tree automata, δ' could also be extended to operate on trees. Here, the automaton carries out a bottom-up parsing where the tuples of states and/or symbols are transformed by using the Parikh mapping Ψ to obtain the multisets in $\mathcal{M}_n(Q \cup V_0)$. If the analysis is completed and δ' returns a multiset with at least one final state, the input tree is accepted. So, δ' can be extended as follows

$\delta'(a) = M_\Psi(a)$ for any $a \in V_0$

$\delta'(t) = \{M \in \delta'_n(\sigma, M_1 \oplus \cdots \oplus M_n) : M_i \in \delta'(t_i)\ 1 \leq i \leq n\}$ for $t = \sigma(t_1, \ldots, t_n)$ $(n > 0)$

Formally, every multiset tree automaton MA accepts the following language

$$L(MA) = \{t \in V^T : M_\Psi(q) \in \delta'(t), q \in F\}$$

Another extension which will be useful is the one related to the ancestors of every state. So, we define $Anc_\Psi(q) = \{M : M_\Psi(q) \in \delta_n(\sigma, M)\}$. The following two results characterize the relation between the languages accepted by tree automata and the multiset tree automata induced by them.

Theorem 3.1 *(Sempere and López, [16]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A and $t = \sigma(t_1, \ldots, t_n) \in V^T$. If $\delta(t) = q$ then $M_\Psi(q) \in \delta'(t)$.*

Corollary 3.1 *(Sempere and López, [16]) Let $A = (Q, V, \delta, F)$ be a tree automaton and $MA = (Q, V, \delta',\ F)$ be the multiset tree automaton induced by A. If $t \in L(A)$ then $t \in L(MA)$.*

Mirrored Equivalent Trees

We will introduce the concept of *mirroring* in tree structures as was exposed in [16]. Informally speaking, two trees will be related by mirroring if some permutations at the structural level are hold. For example, the trees of Figure 1 have identical subtrees except that some internal nodes have changed their order.

We propose a definition that relates all the trees with this mirroring property. For any other concepts used in this section, we refer to the previous section 2 on tree automata.

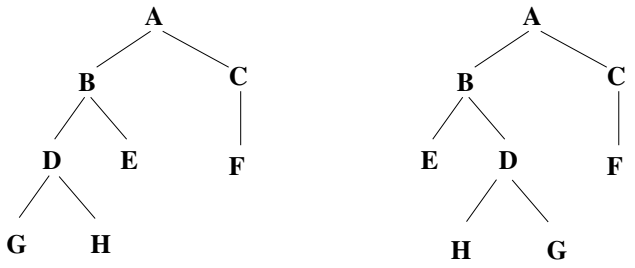


Figure 1: Two mirrored trees

Definition 3.3 *Let (V, r) be a ranked alphabet and t and s be two trees from V^T. We will say that t and s are* mirror equivalent*, denoted by $t \bowtie s$, if one of the following conditions holds:*

1. $t = s = a \in V_0$
2. $t \in perm_1(s)$
3. $t = \sigma(t_1, \ldots, t_n)$, $s = \sigma(s_1, \ldots, s_n)$ *and there exists* $< s^1, s^2, \ldots, s^k > \in perm(< s_1, s_2, ..., s_n >)$ *such that* $t_i \bowtie s^i \ \forall 1 \leq i \leq n$

The following results characterize the set of trees accepted by a multiset tree automaton induced by a tree automaton.

Theorem 3.2 *(Sempere and López, [16]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $t = \sigma(t_1, \ldots, t_n) \in V^T$ and $s = \sigma(s_1, \ldots, s_n) \in V^T$. Let $MA = (Q, V, \delta', F)$ be the multiset tree automaton induced by A. If $t \bowtie s$ then $\delta'(t) = \delta'(s)$.*

Note that the converse result of last theorem is not generally true. For instance, consider the trees: $t = \sigma(a)$ and $s = \sigma(a, \sigma(a))$ and the tree automaton with the following transition function:

$$\delta_1(\sigma, a) = q_1 \in F \qquad \delta_2(\sigma, a, q_1) = q_1 \in F$$

it is easy to see that $\delta'(t) = \delta'(s)$ but t is not mirror equivalent to s.

Corollary 3.2 *(Sempere and López, [16]) Let $A = (Q, V, \delta, F)$ be a tree automaton, $MA = (Q, V, \delta', F)$ the multiset tree automaton induced by A and $t \in V^T$. If $t \in L(MA)$ then, for any $s \in V^T$ such that $t \bowtie s$, $s \in L(MA)$.*

The last results were useful to propose an algorithm to determine whether two trees are mirror equivalent or not [16]. So, given two trees s and t, we can establish in time $\mathcal{O}((min\{|t|, |s|\})^2)$ if $t \bowtie s$.

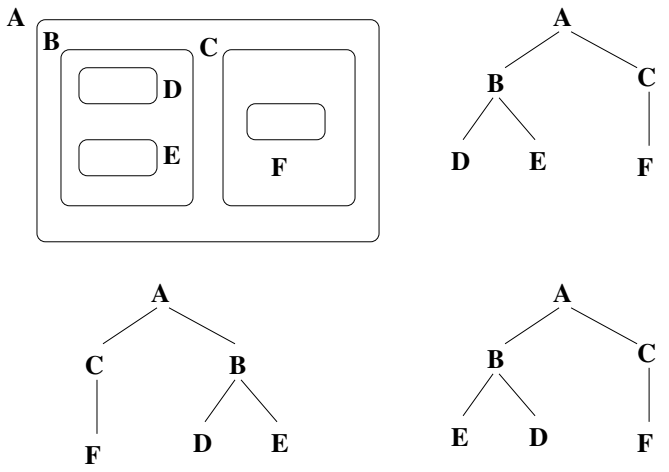


Figure 2: A membrane structure together with different representations by trees

4 Solving the Membrane Structure Recognition Problem

Recently, in [7], a way to generate trees by membrane systems has been proposed. Initially, any membrane structure can be represented by a tree taking the membrane structure as a hierarchical order between regions. Freund et al. [7] have taken advantage of a variant of P systems with active membranes and string objects. Active membranes have an electrical charge (*polarization*) together with a set of rules that allow the membrane to change polarizations, move objects (strings), dissolving the membrane, 2-dividing the membrane, etc. They have proved that any recursively enumerable tree language can be generated by a P system.

A way to recognize two identical membrane structures by taking advantage of tree representations was proposed in [16]. For example, let us see Figure 2, in which we represent a membrane structure with different trees.

Obviously, the initial order of a membrane structure can be fixed. Anyway, whenever the system evolves (membrane dissolving, division, creation, etc.) this order can be at least somehow ambiguous. Furthermore, the initial order of a P system is only a naming convention given that the membrane structure of any P system can be renamed without changing its behavior due to the parallelism ingredient (observe that if this mechanism was sequential then the ordering could be important for the final output).

The representation by trees could be essential for the analysis of the dynamic behavior of P systems. Whenever we work with trees to represent the membrane structure of a given P system, we can find a *mirroring effect*. Again, look at Figure 2: the three different trees proposed for the membrane structure have a mirroring property, that is, some subtrees at a given level of the tree have been permuted.

The method that we propose to establish if two membrane structures μ and $\mu\prime$ are identical is based on the algorithm proposed in [16]. First, we represent μ and $\mu\prime$ by t and s respectively. Then, we apply the proposed algorithm and, if $t \bowtie s$ we can affirm that μ and $\mu\prime$ are identical.

5 Editing Distances Between Membrane Structures

The study of relations between membrane structures is proposed in the sequel. Here, the main problem can be established as follows

"*Let μ and $\mu\prime$ be two membrane structures corresponding to arbitrary P systems. What is the minimum set of membrane rule applications needed to transform one into the other ?*"

The solution of the last problem can be approached by using multiset tree automata and editing distances between trees and tree automata. A previous work [10], considered the case of tree automata. Here, we will extend the previous results to multiset tree automata as described in previous sections.

First, we will describe the method employed in [10], in order to give the main components of the editing distance calculation.

Given a tree automata $A = (Q, V, \delta, F)$ and a tree t, the distance between t and A can be established as the minimum in the set $\{D(t, q) : q \in F\}$, where $D(t, q)$ is the minimum distance of the tree t to the state q. The distance $D(t, q)$ evaluates the number of operations needed to reduce the tree t to the state q according to the function δ in automata A. Some operations involved in the distance refer to operations for trees as *Insertion, Deletion* and *Substitution*. We consider the costs for these operations as exposed in [10]. Observe that these costs are usually defined by taking into account the sizes of the trees. So, the bigger tree involved in the operation, the bigger cost to handle it:

- Insertion

  $\forall a \in V \; I(a) = 1$

  $I(\sigma(t_1, t_2, \ldots, t_k)) = 1 + \sum_{\forall j} I(t_j)$

- Deletion

  $\forall a \in V \ B(a) = 1$

  $B(\sigma(t_1, t_2, \ldots, t_k)) = 1 + \sum_{\forall j} B(t_j)$

- Substitution

  $\forall a \in V \ S(a, a) = 0$

  $\forall a, b \in V \ S(a, b) = 1$

  $S(\sigma(t_1, t_2, \ldots, t_k), a) = B(\sigma(t_1, t_2, \ldots, t_k)) + I(a)$

  $S(a, \sigma(t_1, t_2, \ldots, t_k)) = B(a) + I(\sigma(t_1, t_2, \ldots, t_k))$

So, the distance of every (sub)tree to a tree automaton will involve every ancestor of each state of the automata together with the substructures of the tree. If we have to reduce the structure $\sigma(s_1, s_2, \cdots, s_n)$ to the state q such that $Anc(q)$ contains $< p_1, \ldots, p_m >$, we will have to modify substructures s_i or we will have to insert states p_j at the minimum cost.

The edition cost of every tree to every state of the automaton can be calculated by considering the set of ancestors of the state and the set of successors of the tree. Then we can apply a dynamic programming scheme that takes into account previous calculations which can be stored in a distance matrix. For additional details of this method we refer the reader to [10].

The main components used to calculate the distance of a tree t to a multiset tree automaton MA are the same as in the tree automata case with the following remarks:

1. The successors of any node in the tree are considered as a multiset instead of a sequence.

2. The ancestors of every state in the automaton form a multiset.

3. The editing costs for trees and states are the same as in the tree automata.

4. The calculation of the edit distance is performed by using a edition matrix which can be obtained by using a dynamic programming strategy with some differences which will be explained later.

We propose **Algorithm 1** which obtains the distance from a tree t to a multiset tree automaton MA. Note that the target of the algorithm is to force the automaton to accept the tree. Therefore the set of edit operations

is not fully needed. The algorithm use edit operations for substitution (reduction) of a tree to a state of the automaton, deletion of a (sub)tree and insertion of a state. Intuitively, the substitution of a tree by a state of the automaton could be seen as the substitution of the tree by the nearest tree that could be reduced to the state.

Algorithm 1 Algorithm to obtain the minimum distance from a tree t to the nearest tree in $L(MA)$.

Input:
A multiset tree automaton $A = (Q, V, \delta, F)$.
A tree t.
Output:
Edit distance from t to the automaton A.
Method:
/* initialization */
$\forall t' \in Sub(t) \quad B[t'] = |t'| \quad end\forall$
$\forall a \in V_0 \quad I[a] = 1 \quad end\forall$
$\forall q \in Q$
$I[q] = min\{|t'| \ : \ \delta(t') = q\}$
$\forall t' \in Sub(t)$
$D[t', q] = \infty$
$end\forall$
$end\forall$
$\forall a, b \in V_0$
$$D[a, b] = \begin{cases} 1 & \text{if } a \neq b \\ 0 & \text{otherwise} \end{cases}$$
$D[a, q] = 1 + I[q] \ : \ q \in Q$
$end\forall$
/* iteration */
$\forall t' = \sigma(t'_1, \ldots, t'_n) \in Sub(t)$ /* postorder traverse */
$\forall \delta(\sigma, M) = M_\Psi(p)$
$D[t', p] = min(D[t', p], MMC(t', \delta(\sigma, M)))$
$end\forall$
$end\forall$
Return($min\{D[t', q] \ : \ q \in F\}$)
EndMethod:

The error-correcting analysis method is shown in **Algorithm 1**. First the cost of the basic operations are obtained (i.e. insertion cost of a state

and deletion of a subtree). Each of the calculations carried out are stored in a distance matrix indexed by the set of subtrees and the set of states of the automaton. This matrix is first initialized and the basic distances are stored. Distances between symbols in V_0 and between any symbol and any state of the automaton are also considered.

Note that the key problem of the algorithm is to find, for any subtree $t' = \sigma(t_1, \ldots, t_n)$ of t and any transition $\delta(\sigma, M) = M_\Psi(p)$, with $M \in Anc_\Psi(p))$, the matching of minimum cost between each t_i and the states and symbols in M. This problem can be reduced to the *minimum cost maximum matching* or *maximum bipartite matching* problem [14]. It is known that this problem can be solved in polynomial time by reducing it to the *minimum cost maximum flow* (MCMF) problem (see also [14]). This scheme is similar to the one proposed in [18] where the author considers distances between unordered trees.

Briefly, MCMF looks for obtaining, for a given graph $G = (V, E)$ in which functions *capacity* and *cost* are defined among the edges, the best way (with lower cost) to send the maximum flow between two nodes of the graph. The flow has to take into account the capacity constraint. The cost function measures the penalization of each unit of flow. Several solutions have been implemented to solve this problem and their complexities depend on the number of nodes n and the number of edges of the graph m. A proper algorithm for our purposes could be the one by Edmons and Karp [6] because its complexity depends only on the number of nodes of the graph ($\mathcal{O}(n^3)$).

Given a tree $t = \sigma(t_1, \ldots, t_n)$ and a transition $M_\Psi(p) \in \delta(\sigma, M)$, the minimum cost matching between t_i and the states in M can be obtained by the subroutine MMC. First, this subroutine builds the directed graph from the parameters and set the proper capacities and costs functions among the edges. Then, a general solution could be run in order to solve the matching. The subroutine is shown in **Algorithm 2**.

Intuitively, each successor tree and each state (namely nodes t_i and q_j respectively) have their own nodes in the graph. Each node in one set is connected with all the nodes in the other. These connections model the reduction (substitution) of each tree to each state. Therefore, the capacity of these edges is set to 1 (these edges can be used only once) and the cost is set to the distance between each tree and each state. Note that this distance is always available due to the postorder traverse of the tree.

The set of edit operations we consider also takes into account the insertion of a state. The node iq and the connections between this node and the nodes q_j model the insertion operation. Thus, the cost of these edges is set

to the insertion cost of the state. Note that the number of insertions of each state is bounded by the number of occurrences of the state in M, therefore, the capacities of these edges is set to this value.

In the same way, in order to model the deletion of trees, the node dt and the connections with the successor trees are considered in the graph. Each tree can be deleted only once, therefore the capacity of these edges is set to 1. Obviously, the cost of these edges is set to the cost of deleting the corresponding tree.

The construction of the graph also considers a source node s. This node is connected to the tree nodes, with connectivity 1 and cost 0 (these edges must be selected without cost). The node s is also connected to the node iq and the cost of this edge is set to 0. Note that the number of state insertions is bounded by the number of states, therefore, the capacity of this edge is set to $|M|$. The cost of this connection is set to 0.

Finally, the graph construction considers a sink node ss. This node is connected with the state nodes q_j with cost 0. Note that the edition process aims to fit the set of successors with the multiset of ancestors, thus, the capacity of the edges must be set to the number of occurrences of each state. The node dt is also connected with the node ss with cost 0. This edge models the tree deletions, therefore, the capacity of the connection must be set to the number of trees that can be deleted.

Example 5.1 *Let us consider the tree*

$$t = \sigma(\sigma(b, \sigma(a, \sigma(a, b), a)), \sigma(a, \sigma(a, a)))$$

and the automaton defined by the following transition functions with $q_3 \in F$:

$\delta(\sigma, aq_1a) = q_1 \quad \delta(\sigma, bq_2) = q_2 \quad \delta(\sigma, aa) = q_1$
$\delta(\sigma, b) = q_2 \quad \delta(\sigma, q_1q_2) = q_3$

First, the insertion and deletion costs are obtained.

Deletion costs:

t_1	t_2	t_3	t_4	t_5	t_6
3	*6*	*8*	*3*	*5*	*14*

Insertion costs:

q_1	q_2	q_3
3	*2*	*6*

Algorithm 2 *MMC Subroutine to obtain the Maximum Matching of Minimum Cost.*

Input:

A multiset tree automaton transition $\delta(\sigma, M) = M_\Psi(p)$.
A tree $t = \sigma(t_1, \ldots, t_n)$.

Output:

Minimum cost of the maximum match between $\{t_1, \ldots, t_n\}$ *and* M.

Method:

/* *construction of the graph* */
Let $G = (V, E)$ *where:*
$V = \{t_1, \ldots, t_n\} \cup M \cup \{s, ss, iq, dt\}$
$(t_i, q_j) \in E,\ \forall q_j \in M;\ i : 1..n$
$(t_i, dt) \in E,\ i : 1..n$
$(iq, q_j) \in E,\ \forall q_j \in M$
$(s, t_i) \in E,\ i : 1..n$
$(q_j, ss) \in E,\ \forall q_j \in M$
$(s, iq), (dt, ss) \in E$
/* *set capacities of each edge* */
$c(t_i, q_j) = 1,\ \forall q_j \in M;\ i : 1..n$
$c(t_i, dt) = 1,\ i : 1..n$
$c(iq, q_j) = \#_{q_j}(M),\ i : 1..n$
$c(s, t_i) = 1,\ i : 1..n$
$c(q_j, ss) = \#_{q_j}(M),\ \forall q_j \in M$
$c(s, iq) = |M|,\ c(dt, ss) = n$
/* *set cost of each edge* */
$d(t_i, q_j) = D[t_i, q_j],\ \forall q_j \in M;\ i : 1..n$
$d(t_i, dt) = B[t_i],\ i : 1..n$
$d(iq, q_j) = I[q_j],\ \forall q_j \in M$
$d(s, t_i) = 0,\ i : 1..n$
$d(q_j, ss) = 0,\ \forall q_j \in M$
$d(s, iq) = 0$
$d(dt, ss) = 0$
Return$(MinCostMaxFlow(G))$
EndMethod:

Then, the editing process considers the first postorder subtree $\sigma(a, b)$ *and the first transition* $\delta(\sigma, aq_1a) = q_1$. *The process starts with the construction of the graph shown in Figure 3.*

Solid lines in Figure 3 show the minimum cost matching. The distance

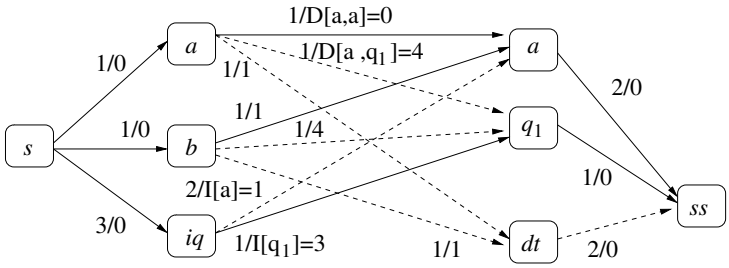


Figure 3: Underlying graph to obtain the distance of the first postorder subtree to the first transition of the automaton. Edge labels show the capacity/cost. Solid lines show the best matching.

is stored in the matrix of distances. Note that this cost is improved when the third transition is considered. The following table shows an intermediate state of the matrix.

D_A	t_1	t_2	t_3	t_4	t_5	t_6
q_1	*1*	*1*	*3*			
q_2	*1*	*3*	*3*			
q_3	*7*	*5*				

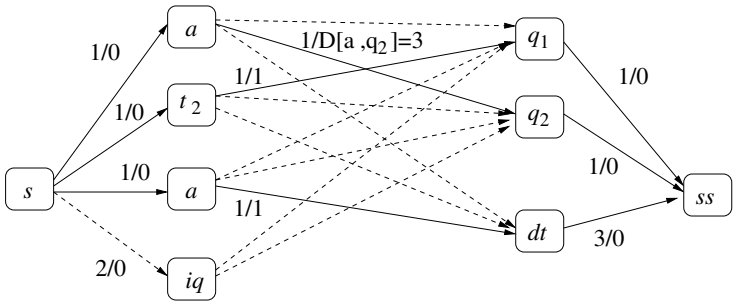


Figure 4: Underlying graph to obtain the distance of the first postorder subtree to the first transition of the automaton. Solid lines show the best matching.

We now show the distance of the third postorder subtree to the state q_3.

The underlying graph is shown in Figure 4. The best matching is shown in solid lines. □

Observe that the minimum editing distance that we have calculated can be established in terms of operations which have a translation into membrane rules. Let us consider that μ is the membrane structure which is accepted by the multiset tree automaton MA and μ' is represented by a tree t. We have the following correspondences between edition operations and membrane rules:

1. Insertion of state q

   Let us suppose that the insertion is produced to match the ancestors of a state p. The minimum tree that can be reduced to q is t_j. The operations needed to achieve this goal in the membrane structure are membrane creation at region p in order to obtain membrane structure t_j.

2. Reduction of tree t_i to state q

   Let us suppose that the tree which can be reduced to q with a minimal cost is t_j, according with the δ function of the automaton. The operations needed to make this reduction are the ones involved to transform t_i to t_j at a region k. These operations consider again membrane creation and dissolving depending on the operations involved in the minimum distance from t_i to t_j.

3. Substitution of a by b

   The region a is dissolved and created with a new label.

4. Deletion of tree t_i

   Let us suppose that t_i is a membrane structure at region k. The deletion consists of several membrane dissolving of structure t_i.

6 Conclusions and Future Work

We have proposed a method to calculate the minimum number of membrane rules needed to transform a membrane structure into a different one. The number of rules needed, if so, establishes an editing distance between P systems by taking into account only membrane modifications. This measure

can provide new definitions about *structural confluence* in P systems, that is structural agreement during evolution.

Observe that we have worked with simplified version of P systems. That is, the objects inside any region do not influence the editing distance. A future research will consider how the objects can be taken into account to calculate the editing distance.

Acknowledgements

The authors are grateful to the anonymous referees which have made sharp remarks to this work. Special thanks are given to Gheorghe Păun and Mario Pérez-Jiménez for useful discussions during the *3rd Brainstorming Week on Membrane Computing* which was held in Sevilla from 31st January to 4th February 2005.

References

[1] A. Alhazov, T.O. Ishdorj: Membrane operations in P systems with active membranes. In: Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho Caparrini (Eds.): *Proceedings of the Second Brainstorming Week on Membrane Computing*. TR 01/04 of RGNC, Sevilla University (2004), 37–44.

[2] C. Calude, Gh. Păun, G. Rozenberg and A. Salomaa: Multiset Processing. *Lecture Notes in Computer Science* **2235**, Springer–Verlag, Berlin (2001).

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, M. Tommasi: Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata` (1997).

[4] E. Csuhaj-Varjú, A. Di Nola, Gh. Păun, M. Pérez-Jiménez, G. Vaszil: Editing Configurations of P systems. Submitted (2005).

[5] E. Csuhaj-Varjú, C. Martín-Vide, V. Mitrana: Multiset Automata. In [2], 69–83.

[6] J. Edmonds and R.M. Karp: Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM* **19** (2) (1972), 248–264.

[7] R. Freund, M. Oswald, A. Păun. P Systems Generating Trees. In: G. Mauri, Gh. Păun, C. Zandron (Eds.): *Pre-proceedings of Fifth Workshop on Membrane Computing* (WMC5) MolCoNet project IST-2001-32008 (2004), 221–232.

[8] F. Gécseg and M. Steinby: *Handbook of Formal Languages,* Volume 3. Springer–Verlag, Berlin (1997), 1–69.

[9] M. Kudlek, C. Martín-Vide, Gh. Păun: Towards a Formal Macroset Theory. In [2], 123–133.

[10] D. López, J. M. Sempere, P. García: Error correcting analysis for tree languages. *International Journal of Pattern Recognition and Artificial Intelligence* **12** (2000), 357–368.

[11] A. Păun: On P systems with active membranes. In: *Proceedings of the First Conference on Unconventionals Models of Computation* (2000), 187–201.

[12] Gh. Păun: *Membrane Computing: An Introduction.* Springer–Verlag, Berlin (2002).

[13] Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori: On the power of membrane division in P systems. In: *Proc. Conf. on Words, Languages and Combinatorics.* Kyoto (2000).

[14] R.L. Rivest T.H. Cormen, C.E. Leiserson, C. Stein: *Introduction to Algorithms.* MIT Press and McGraw-Hill, second edition (2001).

[15] G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages.* Volume 1, Springer–Verlag, Berlin (1997).

[16] J. M. Sempere, D. López: Recognizing membrane structures with tree automata. In M.A. Gutiérrez Naranjo, A. Riscos-Núñez, F.J. Romero-Campero, D. Sburlan (Eds.): *Proceedings of the 3rd Brainstorming Week on Membrane Computing 2005. RGNC Report* 1(2005) Research Group on Natural Computing, Sevilla University, Fénix Editora (2005), 305–316.

[17] A. Syropoulos: Mathematics of Multisets. In [2], 347–358.

[18] K. Zhang: A constrained edit distance between unordered labelled trees. *Algorithmica* **15** (1996), 205–222.

Relational Membrane Systems

Adam OBTUŁOWICZ

Institute of Mathematics
Polish Academy of Sciences
E-mail: adamo@impan.gov.pl

We continue the discussion, initiated in [2], of the interconnections between membrane computing, cf. [3], and Gandy's mechanisms, cf. [1].

Both membrane computing approach and Gandy's mechanism approach to computing devices concern hierarchically organized systems, where the hierarchical organization of a given system is determined by the nesting relation of the less complex parts of the system in the more complex parts of the system.

In membrane computing approach the nesting relation which determines the hierarchical organization of a given membrane system is modeled by the tree whose nodes are membranes of the system, where membranes correspond to the parts of the system and for all two different membranes of the system there is no common membrane which is nested simultaneously in both of them.

In Gandy's mechanism approach the nesting relation which determines hierarchical organization of a given system is modeled by the restriction of the membership relation $\in$ to the union $\mathrm{WTC}(x) \cup L$ for that hereditary finite set x which is a model of the whole system, where $\mathrm{WTC}(x)$ is the weak transitive closure of x and L is a set of urelements, see Appendix. In this case urelements are elementary (indecomposable) parts and the elements of $\mathrm{WTC}(x)$ are composite parts, where for two different composite parts there may exist a common composite part which is nested in both of them.

Thus the considered approaches are different because of the shape of models used in them.

We introduce and discuss a notion of a relational membrane system which is a generalization of models used in the considered approaches and which includes the case of a fuzzy nesting relation appearing in practice.

Let $\mathcal{D} = \{N, [0,1]\}$, where N is the set of natural numbers with 0 and $[0,1]$ is the closed unit interval.

For a set $D \in \mathcal{D}$ we define a [finite] *D-relational membrane system* S to be given by a function $\mathcal{E}_S : \mathbb{U}_S \times \mathbb{U}_S \to D$, a distinguished proper subset $\mathbb{O}_S$ of [a finite] $\mathbb{U}_S$ and a distinguished element $r_S \in \mathbb{U}_S - \mathbb{O}_S$ such that the following conditions hold:

i) $\mathcal{E}_S(m, a) = 0$ for all $m \in \mathbb{U}_S$ and $a \in \mathbb{O}_S$,

ii) the *underlying graph* G_S of S with the set V_S of vertices given by

$$V_S = \mathbb{U}_S - \mathbb{O}_S$$

and the set E_S of edges given by

$$E_S = \{(m, m') \in V_S \times V_S \,|\, \mathcal{E}_S(m', m) > 0\}$$

is a *rooted graph with the root* r_S, i.e., G_S is an acyclic graph and for every $m \in V_S$ there exist a natural number $n > 0$ and a route $m_1 \dots m_n$ in G_S whose first element m_1 is r_S and the last element m_n is m, where a *route* in a directed graph G is meant to be a finite string $v_1 \dots v_n$ with $n > 0$ of vertices of G such that if $n > 1$, then (v_i, v_{i+1}) is an edge of G for all i with $1 \leq i < n$.

The sets $\mathbb{U}_S, \mathbb{O}_S, \mathbb{U}_S - \mathbb{O}_S$ are called the *universe* of S, the *set of objects* of S, the *set of membranes* of S, respectively, the function $\mathcal{E}_S$ is called the *immediate nesting relation* of S, and r_S is called the *root* or the *skin* of S.

The immediate nesting relation $\mathcal{E}_S$ of a D-relational membrane system S is interpreted in the following way:

— for $D = N$ the value $\mathcal{E}_S(x, y)$ means that exactly $\mathcal{E}_S(x, y)$ copies of part x are immediately nested in part y,

— for $D = [0, 1]$ the value $\mathcal{E}_S(x, y)$ means that with certainty degree $\mathcal{E}_S(x, y)$ part x is immediately nested in part y.

Thus $[0, 1]$-relational membrane systems are fuzzy relational membrane systems.

If S is a finite N-relational membrane system whose underlying graph G_S is a tree, i.e., for every $m \in V_S$ there exist a unique route in G_S whose first element is r_S and the last element is m, and the following condition holds:

iii) $$\mathcal{E}_S(m, m') \leq 1 \quad \text{for all } m, m' \in \mathbb{U}_S - \mathbb{O}_S,$$

then S is a usual membrane system whose set of membranes is $\mathbb{U}_S - \mathbb{O}_S$, the set of objects is $\mathbb{O}_S$, G_S corresponds to the membrane structure of S in Păun's sense, and for all $m \in \mathbb{U}_S - \mathbb{O}_S$ and $a \in \mathbb{O}_S$ the value $\mathcal{E}_S(a, m)$ means that exactly $\mathcal{E}_S(a, m)$ copies of a are contained in the region of m.

For every hereditary finite set x with urelements in L the characteristic function of the restriction of the membership relation $\in$ to the set $\mathrm{WTC}(x) \cup L$ is the immediate nesting relation of that N-relational membrane system whose universe is $\mathrm{WTC}(x) \cup L$ and the set of objects is L.

For every finite N-relational membrane system S satisfying the condition

iv) $$\mathcal{E}_S(x, y) \leq 1 \quad \text{for all } x, y \in \mathbb{U}_S,$$

one constructs a hereditary finite set $\mathrm{hf}(S)$ over $\mathbb{U}_S$ such that S is isomorphic to that N-relational membrane system S' whose universe is $\mathrm{WTC}(\mathrm{hf}(S)) \cup \mathbb{O}_S$ and the immediate nesting relation $\mathcal{E}_{S'}$ of S' is the characteristic function of the restriction of the membership relation $\in$ to $\mathrm{WTC}(\mathrm{hf}(S)) \cup \mathbb{O}_S$.

The set $\mathrm{hf}(S)$ is defined inductively by

$$\begin{aligned}\mathrm{hf}(S) = \{a \in \mathbb{O}_S \,|\, \mathcal{E}_S(a, r_S) > 0\} \cup \{r_S\} \\ \cup \{\mathrm{hf}(S(m)) \,|\, \mathcal{E}_S(m, r_S) > 0 \text{ and } m \in \mathbb{U}_S - \mathbb{O}_S\},\end{aligned}$$

where $S(m)$ is that subsystem of S whose immediate nesting relation $\mathcal{E}_{S(m)}$ is the restriction of $\mathcal{E}_S$ to the set

$$\begin{aligned}\mathbb{U}_{S(m)} = \{m' \in \mathbb{U}_S - \mathbb{O}_S \,|\, &\text{there exists a route in } G_S \\ &\text{with the first element } m \text{ and the last element } m'\} \cup \mathbb{O}_S\end{aligned}$$

and m is the root of $S(m)$.

The construction of $\mathrm{hf}(S)$ appears useful for a discussion of certain evolutive transformations of N-relational membrane systems which are determined by simultaneous applications of different evolution rules. Namely, these evolutive transformations can be described in terms of hereditary finite sets $\mathrm{hf}(S)$ by using set theoretical operations of the union, the intersection, the difference of sets, and that unary operation $\{?\}$ whose value is $\{x\}$ for a hereditary finite set x. We begin with a description of evolutive transformations of hereditary finite sets themselves.

We consider those evolutive transformations of hereditary finite set into hereditary finite sets which are determined by evolution rules written in Păun's manner as the parenthese expressions:

$R_1)$ $[a] \to b$ (*dissolution rule*),

R_2) $[a][b] \to [b[a]]$ (*in-rule*),

R_3) $[[a]] \to [a][\,]$ (*out-rule*),

where a, b are urelements.

The single applications from the top of the above rules to hereditary finite sets are described in the following way:

— if $a \in y \in x \in \mathrm{HF}$, then the dissolution rule $[a] \to b$ *can be applied to* x and the result of its application is a new hereditary finite set of the form

$$(x - \{y\}) \cup (y - \{a\}) \cup \{b\},$$

— if $a \in y \in x \in \mathrm{HF}$, $b \in z \in x$, and $z \neq y$, then the in-rule $[a][b] \to [b[a]]$ *can be applied to* x and the result of its application is a new hereditary finite set of the form

$$(x - \{y, z\}) \cup \{z \cup \{y\}\},$$

— if $a \in z \in y \in x \in \mathrm{HF}$ and $y - \{z\} \neq \emptyset$, then the out-rule $[[a]] \to [a][\,]$ *can be applied to* x and the result of its application is a new hereditary finite set of the form

$$(x - \{y\}) \cup \{y - \{z\}, z\}.$$

The above described single applications of evolution rules R_1), R_2), R_3) from the top determine evolutive transformations of hereditary finite sets into the new hereditary finite sets from the top. One can describe by using $\cup$, $-$, and $\{?\}$ a more complicated case of evolutive transformations of hereditary finite sets, where these transformations are determined by simultaneous applications of many different rules to many different elements of $\mathrm{WTC}(x)$ for a hereditary finite set x to be transformed and those elements are not necessarily the elements of x itself.

The evolutive transformations of hereditary finite sets considered above can be "transferred" to N-relational membrane systems by using the construction of $\mathrm{hf}(S)$ to define evolutive transformations of N-relational membrane systems themselves. These evolutive transformations of N-relational membrane system can be then applied to verify correctness of massively parallel computations realized by evolution processes of N-relational membrane systems.

Appendix

For a potentially infinite set L of labels or names which are *urelements*, i.e., they are not (treated as) sets themselves, we define inductively a family of sets HF_i for natural numbers $i \geq 0$ such that

$$\begin{aligned} \mathrm{HF}_0 &= \emptyset, \\ \mathrm{HF}_{i+1} &= \text{the set of nonempty finite subsets of } L \cup \mathrm{HF}_i. \end{aligned}$$

The elements of the union $\mathrm{HF} = \bigcup\{\mathrm{HF}_i \,|\, i \geq 0\} \cup \{\emptyset\}$ are called *hereditary finite sets over* L or *hereditary finite sets with urelements* in L, or simply *hereditary finite sets* if there is no risk of confusion.

For $x \in \mathrm{HF}$ we define its weak transitive closure $\mathrm{WTC}(x)$ by

$$\mathrm{WTC}(x) = \bigcup\{\mathrm{WTC}(y) \,|\, y \in x \text{ and } y \in \mathrm{HF}\} \cup \{x\}.$$

References

[1] R. Gandy: Church's thesis and principles for mechanisms. In: J. Barwise et al. (Eds.): *The Kleene Symposium.* North-Holland, Amsterdam (1980), 123–148.

[2] A. Obtułowicz: Gandy's principles for mechanisms and membrane computing. In: Gh. Păun and M. J. Pérez-Jiménez Eds.: *Proc. of 3rd Brainstorming Week on Membrane Computing*. Sevilla (2005).

[3] Gh. Păun: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin (2002).

On the Rule Complexity of Universal Tissue P Systems

Yurii ROGOZHIN[1], **Sergey VERLAN**[2]

[1]Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
E-mail: rogozhin@math.md

[2]LITA, Université de Metz, France
E-mail: verlan@univ-metz.fr

Abstract

In the last time several attempts to decrease different complexity parameters (number of membranes, size of rules, number of objects etc.) of universal P systems were done. In this article we consider another parameter which was not investigated yet: the number of rules of the system. We show that 8 rules suffice to recognise any recursively enumerable language if splicing tissue P systems are considered.

1 Introduction

P systems were introduced by Gh. Păun in [6] as distributed parallel computing devices of biochemical inspiration, specifically, starting from the structure and the functioning of a living cell. The cell is considered as a set of compartments (membranes) nested one in another and which contain objects and evolution rules. The base model does not specify neither the nature of these objects, nor the nature of rules. Numerous variants specify these two parameters by obtaining a lot of different models of computing, see [11] for a comprehensive bibliography.

The inspiration for *tissue P systems* comes from two sides. On one hand, P systems previously introduced may be viewed as transformations of labels associated to nodes of a tree. Therefore, it is natural to consider same transformations on a graph. On the other hand, they may be obtained by following the same reflections as for P systems, but starting from a tissue of cells and not from a single cell.

Tissue P systems were first considered by Gh. Păun and T. Yokomori in [8] and [9]. They have richer possibilities and the advantages of new topology have to be investigated.

There are many results dealing with the descriptional complexity of (tissue) P systems. In most of the cases, the main complexity parameter of such systems – the number of membranes/cells is investigated. Recently, other parameters such as the size of rules or the number of objects started to be investigated. For example, in [5, 1] systems having a minimal number of objects are investigated.

In this article we consider another complexity parameter, *the number of rules*, which has not been investigated yet. We take a particular class of tissue P systems, *splicing tissue P systems*, which is a mixture of tissue P systems and splicing Head systems and which were introduced by Gh. Păun in [6]. In this case, we show that systems having 8 rules are universal and that they can recognise any recursively enumerable language modulo a suitable codification.

2 Definitions

We do not present here definitions concerning concepts of the theory of formal languages. We refer to [3] and [10] for more details. We only remark that we denote the empty word by ε.

A *tag system* of degree $m > 0$, see [2] and [4], is a triplet $T = (m, V, P)$, where $V = \{a_1, \ldots, a_{n+1}\}$ is an alphabet and where P is a set of productions of form $a_i \to P_i, 1 \leq i \leq n$, $P_i \in V^*$. The symbol a_{n+1} is called a halting symbol. A configuration of the system T is a word w. We pass from the configuration $w = a_{i_1} \ldots a_{i_m} w'$ to the next configuration z by erasing the first m symbols of w and by adding P_{i_1} to the end of the word: $w \Rightarrow z$, if $z = w' P_{i_1}$.

The computation of T over the word $x \in V^*$ is a sequence of configurations $x \Rightarrow \ldots \Rightarrow y$, where either $y = a_{n+1} a_{i_1} \ldots a_{i_{m-1}} y'$, or $y' = y$ and $|y'| < m$, where $|w|$ is the length of word w. In this case we say that T halts on x and that y' is the result of the computation of T over x. We say that T recognises the language L if for all $x \in L$, T halts on x, and T halts only on words from L.

We note that tag systems of degree 2 are able to recognise the family of recursively enumerable languages, see [2] and [4]. Moreover, systems constructed in [2] have non-empty productions and halt only by reaching the symbol a_{n+1} in first position.

2.1 Splicing Operation

By an (abstract) *molecule* we understand a word over an alphabet.

A *splicing rule* (over an alphabet V) is a 4-tuple (u_1, u_2, u_3, u_4) where $u_1, u_2, u_3, u_4 \in V^*$. It is frequently written as $u_1\#u_2\$u_3\#u_4$, $\{\$, \#\} \notin V$, or in two dimensions as $\begin{array}{c|c} u_1 & u_2 \\ \hline u_3 & u_4 \end{array}$. Strings u_1u_2 and u_3u_4 are called splicing *sites*. The *diameter* of splicing rule $u_1\#u_2\$u_3\#u_4$ is the following vector $(|w_1|, |w_2|, |w_3|, |w_4|)$.

We say that a word x *matches* rule r if x contains an occurrence of one of the two sites of r. We also say that x and y are *complementary* with respect to a rule r if x contains one site of r and y contains the other one. In this case we also say that x or y may *enter* rule r. When x and y can enter a rule $r = u_1\#u_2\$u_3\#u_4$, *i.e.*, we have $x = x_1u_1u_2x_2$ and $y = y_1u_3u_4y_2$, it is possible to define the application of r to couple x, y. The result of this application is w and z where $w = x_1u_1u_4y_2$ and $z = y_1u_3u_2x_2$. We also say that x and y are spliced and w and z are the result of this splicing. We write this as follows: $(x, y) \vdash_r (w, z)$ or

$$\begin{array}{c|c} x_1u_1 & u_2x_2 \\ \hline y_1u_3 & u_4y_2 \end{array} \vdash_r \begin{array}{c} x_1u_1u_4y_2 \\ \hline y_1u_3u_2x_2 \end{array}.$$

The pair $\sigma = (V, R)$ where V is an alphabet and R is a set of splicing rules is called a *splicing scheme* or an H-scheme.

For a splicing scheme $\sigma = (V, R)$ and for a language $L \subseteq V^*$ we define:
$\sigma(L) \stackrel{\text{def}}{=} \{w, z \in V^* \mid \exists x, y \in L, \exists r \in R : (x, y) \vdash_r (w, z)\}$.
Now we can introduce the iteration of the splicing operation.
$\sigma^0(L) = L$,
$\sigma^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)),\ i \geq 0$,
$\sigma^*(L) = \cup_{i \geq 0}\sigma^i(L)$.
The iterated splicing preserves the regularity of a language:

Theorem 1. *[7] Let $L \subseteq T^*$ be a regular language and let $\sigma = (T, R)$ be a splicing scheme. Then language $\sigma^*(L)$ is regular.*

2.2 Splicing Tissue P Systems

A *splicing tissue P system* of degree $m \geq 1$ is a construct

$$\Pi = (V, T, G, A_1, \ldots, A_m, R_1, \ldots, R_m),$$

where V is a finite alphabet, $T \subseteq V$ is the terminal alphabet and G is the underlying directed labeled graph of the system. The graph G has m nodes

(cells) numbered from 1 to m. Each node has a label that contains a set of strings (a language) over V. The symbols $A_1, \ldots, A_m$ are finite sets of strings over V that give initial labels of nodes of G. Symbols R_i, $1 \leq i \leq m$ are finite sets of rules (associated to regions) of the form $(r; tar_1, tar_2)$, where r is a splicing rule: $r = u_1\#u_2\$u_3\#u_4$ and $tar_1, tar_2 \in \{here, go_j, out\}$, $1 \leq j \leq m$ are target indicators.

A *configuration* of Π is the m-tuple $(N_1, \ldots, N_m)$, where $N_i \subseteq V^*$. A *transition* between two configurations $(N_1, \ldots, N_m) \Rightarrow (N'_1, \ldots, N'_m)$ is defined as follows. In order to pass from one configuration to another, splicing rules of each node are applied in parallel to all possible words that belong to the label of that node. After that, the result of each splicing is distributed according to target indicators. More exactly, if there are x, y in N_i and $r = (u_1\#u_2\$u_3\#u_4; tar_1; tar_2)$ in R_i, such that $(x, y) \vdash_r (w, z)$, then words w and z are sent to nodes indicated by tar_1, respectively tar_2. We write this as follows $(x, y) \vdash_r (w, z)(tar_1, tar_2)$. If $tar_k = here$, $k = 1, 2$ then the word remains in node i; if $tar_k = go_j$, then the word is sent to node j (it is clear that there must be an edge (i, j) in G); if $tar_k = out$, the word is sent outside of the system.

Since the words are present in an arbitrary number of copies, after the application of rule r in node i, words x and y are still present in the same node.

A *computation* in a splicing tissue P system Π is a sequence of transitions between configurations of Π which starts from the initial configuration $(A_1, \ldots, A_m)$. The result of the computation consists of all words over terminal alphabet T which are sent outside the system at some moment of the computation. We denote by $L(\Pi)$ the language generated by system Π.

We also define the notion of an *input* for the system above. An input word for a system Π is simply a word w over the non-terminal alphabet of Π. The computation of Π on input w is obtained by adding w to the axioms of A_1 and after that by evolving Π as usual.

We denote by $ELStP_m(spl, go)$ the family of languages generated by tissue splicing P systems having a degree at most m.

We shall consider a restriction of splicing tissue P systems. A *restricted splicing tissue P system* is a special class of splicing tissue P systems which has the property that for any rule $(r; tar_1, tar_2)$ either $tar_1 = tar_2 = go_j$, or $tar_1 = tar_2 = out$. This means that both resulting strings are moved over the same connection. In this case, we may associate splicing rules to corresponding edges. If both targets are *out*, then we associate the splicing rule with an edge going to a special node called *out*. A restricted splicing tissue P system will be denoted as $(V, T, G, A_1, \ldots, A_m, R)$, where V, T, G,

and A_i, $1 \leq i \leq m$, have the same meaning as before and R is a set of splicing rules associated to edges.

3 Main Results

Let $V = \{a_1, \ldots, a_n\}$ be an alphabet. Consider coding functions c and $\bar{c}$ defined as follows: $c(a_i) = \alpha^i\beta$ and $\bar{c}(a_i) = \beta\alpha^i$. We extend these functions to words and put $c(w) = c(b_1)\ldots c(b_m)$ if $w = b_1 \ldots b_m$.

Theorem 2. *Let $TS = (2, V, P)$ be a tag system and $w \in V^*$. Then, there is a restricted splicing tissue P system $\Pi = (V', T, G, A_1, \ldots, A_m, R)$, having 8 rules, which given the word $Xc(w)Y$ as input simulates TS on input w,* i.e. *such that:*

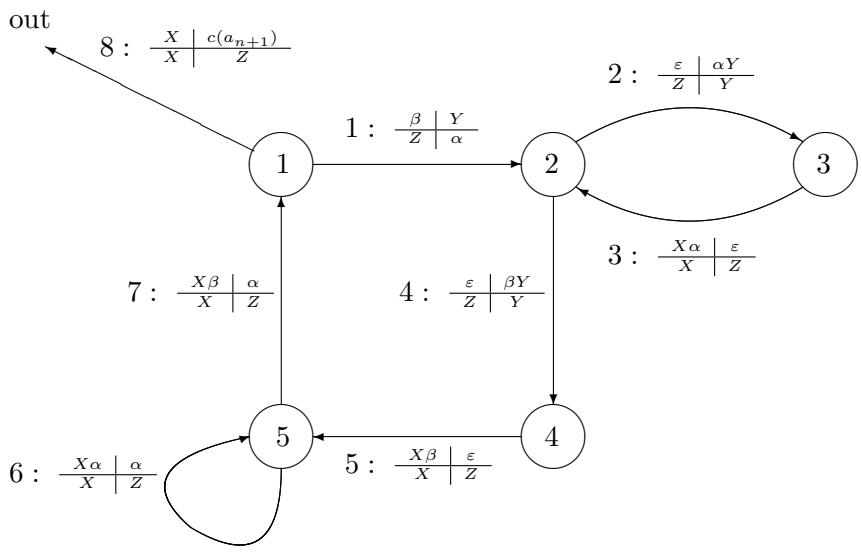
1. *for any word w on which TS halts producing the result w', the system Π produce an unique result $Xc(w')Y$.*
2. *for any word w on which TS does not halt, the system Π computes infinitely without producing a result.*

Proof. We construct the system Π as follows.

Let $|V| = n$.

$V' = \{\alpha, \beta, X, Y, Z\}$, $T = \{X, Y, \alpha, \beta\}$.

The graph G and rules from R are given below:



The initial languages A_j are given as follows.

$A_1 = \{Zc(P_i)\bar{c}(a_i)Y \mid a_i \to P_i \in P\} \cup \{XZ\}$, $A_2 = \{ZY\}$, $A_3 = A_4 = A_5 = \{XZ\}$.

The main idea of the construction is the following. Using rule 1 one attaches a production P_i and symbol a_i, $1 \leq i \leq n$ at the end of the word (in this way a guess is made about the first symbol of the word). After that, indices of the first and the last symbol are decreased simultaneously by taking off one α (since all symbols are coded in unary alphabet this decreases the index of the symbol). This work is done by rules 2 and 3. When the same number of α is present at both ends of a string, *i.e.*, both indices coincide, the system removes one more symbol and the string returns to node 1 where it may be processed again. The check for equality is made by rules 4 and 5, while the elimination of the second symbol is done by rules 6 and 7. When a symbol a_{n+1} begins the string, rule 8 is used and the resulting string is sent outside of the system. It is quite obvious that the system simulates in this way productions of the tag system. It is also clear that a successful computation in TS may be reconstructed from a successful computation in Π. For this it is enough to look at strings of form XwY in node 1. □

Remark 1. *It is clear that the alphabet V' may be reduced to two elements by reencoding letters X, Y, Z, α, β in binary alphabet.*

Hence the system constructed above needs 8 rules, 2 symbols and $n + 5$ initial axioms. The diameter is given in the following table:

Rule(s)	Diameter
1	$(1, 1, 1, 1)$
2, 4	$(0, 2, 1, 1)$
3, 5	$(2, 0, 1, 1)$
6, 7	$(2, 1, 1, 1)$
8	$(1, n + 2, 1, 1)$

It is easy to observe that if we put $c(a_{n+1}) = \beta\beta$, then the last diameter becomes $(1, 2, 1, 1)$, hence the diameter of the whole system is $(2, 2, 1, 1)$.

4 Conclusions

In this work we investigated a different complexity parameter of (tissue) P systems – the number of rules – which was not investigated before. We showed that it is possible to construct a universal system having only 8 rules. In order to achieve this, we used a particular class of splicing tissue

P systems. An open problem raised by this result is if the above number is minimal. Other open problems concern the minimal number of rules in the case of ordinary P systems or P systems with symbol-objects.

References

[1] A. Alhazov, R. Freund, M. Oswald: Tissue P systems with antiport rules and small numbers of symbols and cells. In: *Proceedings of the ESF Exploratory Workshop on Cellular Computing (Complexity Aspects),* Sevilla (Spain), January 31st - February 2nd (2005), 7–22.

[2] J. Cocke, M. Minsky: Universality of tag systems with p=2. *Journal of the ACM*, **11** (1) (1964), 15–20.

[3] J. Hopcroft, R. Motwani, J. Ullman: *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Mass., 2nd edition (2001).

[4] M. Minsky: *Computations: Finite and Infinite Machines.* Prentice Hall, Englewood Cliffts, NJ (1967).

[5] Gh. Păun, J. Pazos, M. J. Pérez-Jiménez, A. Rodriguez-Patón: Symport/antiport P Systems with three objects are universal. *Fundamenta Informaticae*, **64** (1–4) (2005), 345–358.

[6] Gh. Păun: Computing with membranes.*Journal of Computer and System Sciences* **1** (61) (2000), 108–143. Also TUCS Report No. 208 (1998).

[7] Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing: New Computing Paradigms.* Springer-Verlag, Berlin (1998).

[8] Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen* **60** (2002), 635–660.

[9] Gh. Păun, T. Yokomori: Membrane computing based on splicing. In: E. Winfree, D. K. Gifford (Eds.): *DNA Based Computers.* American Mathematical Society (1999), 217–232.

[10] G. Rozenberg, A. Salomaa:*Handbook of Formal Languages.* Springer-Verlag, Berlin (1997).

[11] The P Systems Web Page `http://psystems.disco.unimib.it/`

Modeling the Dynamical Parallelism of Bio-Systems

Dragoş SBURLAN

Department of Computer Science and A.I.
University of Seville, Reina Mercedes Ave., Seville, Spain

Faculty of Mathematics and Informatics
Ovidius University of Constantza
124 Mamaia Ave., Constantza, Romania
E-mail: dsburlan@univ-ovidius.ro

Abstract

Among the many events that occur in the life of biological organisms are a multitude of specific chemical transformations, which provide the cell with usable energy and the molecules needed to form its structure and coordinate its activities. These biochemical reactions as well as all other cellular processes are governed by basic principles of chemistry and physics. A significant factor that determines whether or not reactions could take place is the entropy and it measures the randomness of the system. This measure depends on various factors like degrees of freedom (movement, vibration) for molecules, order in the solution, number of molecules, etc. In an abstract framework, all these factors that describe the way molecules interact can be expressed by means of a multivalued function that depends on the current state of the system. Inspired by these facts, here we introduce and study several bio-mimetic computational systems that use rewriting rules, working in a degree of parallelism (specified by a multivalued function) that depends on the current state of the system. Moreover, we are interested by systems that produce the same output, independently of the multivalued mappings considered.

1 Introduction

In nature, we often find biological systems (but not only) that are not necessarily homogeneous, consisting of many discrete, interacting entities that

have a certain physical spatial distribution. This fact suggests that even if these entities interact in a parallel manner, they obey to some local conditions (concentration for instance) and therefore the interaction parallelism cannot be considered as maximal or fixed, but as a variable that depends on the state of the system.

For example, at the cell level, bimolecular mechanisms are the result of many different chemical reactions that take place with a certain degree of mutual independence but such that they finally (and amazingly) exhibit an overall co-ordination. Traditionally these behaviors were modeled using the theory of partial derivative equations and nonlinear dynamical systems. However, this approach usually gave the general evolution and the dynamics of the system but not always giving the exact solution. From the discrete point of view, the interest was mainly in inferring the properties of the languages generated by such bio-inspired models. Although discrete models of complex phenomena may generate errors, the magnitude of the errors can be arbitrarily reduced by considering a better granularity of the phenomenon. This approach leads in general to a high computational effort, so a more efficient way of studying properties of bio-systems might be to consider discrete formal systems that have embedded into their formal description a certain degree of randomness that describes the way systems evolve.

One such property regards the measure of parallelism. From this point of view, using biochemical reasoning, one might predict for instance that given a particular state of a bio-system and the rules that make it evolves, an approximate next state is reached after a particular time. Basically, even if one does not know the exact number of times the rules are applied, one knows that after a particular time the reactions that had the potential to be applied, were actually accomplished in an approximate rate with respect to the state of the system.

Moreover, from the computer science point of view, in case we are trying to make use of bio-systems as computational devices we should be able to control their behavior no matter the rate of parallelism occurring within them. Therefore, we are interested in systems that one might call "parallel fault tolerant" meaning that they produce the same output no matter which is the "evolution" of the parallelism. This assumption might also have a biological counterpart, namely natural sub-systems are able to regulate themselves and replace, in case is needed, the functions of other sub-systems such that the overall system can realize the same task. From this point of view one can assume that a complex bio-system (like a cell or whatever organism) has the ability to reach a "desired" state, no matter how local decisions were made.

Here we will consider two bio-mimetic models, namely Lindenmayer systems, inspired by the development of multi-cellular organisms and P system with promoters/inhibitors, inspired by the enzyme activation/deactivation process, occurring in the cells.

We assume the reader to be familiar with basic notions of Lindenmayer systems and P systems.

2 On the Dynamical Parallelism of L Systems

In this section we extend the classical definitions of Lindenmayer systems in order to fit a more general perspective. Such systems model biological developments in which parts of organism change simultaneously but not in the total parallel manner as in the classical Lindenmayer theory, but with respect to the current state of the organism. Several results regarding the computational power of these systems are also presented.

Given a set B, the set of all subsets of B is denoted by 2^B. Given another set A, a map $F : A \to 2^B$ is called a multivalued map, from A into B. The special case where $F(x)$ is the singleton $\{F(x)\}$ for every $x \in A$ will be emphasized by using the term single-valued (or simply a map) for F.

Definition 2.1 *An M-rate $0L$ system, denoted by $M0L$, is a triplet $H = (V, P, \omega)$, where $V = \{a_1, \ldots, a_m\}$ is a finite alphabet, P is a set of rules of the form $i : (a \to \alpha,\ F_i)$, $a \in V$, $\alpha \in V^*$, $F_i : V^* \to 2^{I\!N}$, such that $F_i(z) \in \mathcal{P}(\{0, 1, 2, \ldots, |z|_a\})$, for all $z \in V^*$, is a partially defined multivalued map, and $\omega \in V^*$ is the axiom. The set of rules P has to be complete, i.e., for each symbol $a \in V$ there must exist at least one rule $i : (a \to \alpha,\ F_i) \in P$ with this letter a on the left side.*

Consider the partition $P = P_1 \cup P_2 \cup \ldots \cup P_m$ where $P_k = \{(a_k \to \alpha, F) \mid (a_k \to \alpha, F) \in P\}$. Denote $t_k = |P_k|$ and let $P_k = \{(a_k \to \alpha_1, F_{(k,1)}), \ldots, (a_k \to \alpha_{t_k}, F_{(k,t_k)})\}$.

$M0L$ systems use M-rate parallel derivations, i.e., x directly derives y in a $M0L$ system $H = (V, P, \omega)$, with $x, y \in V^*$, written as $x \overset{MOL}{\Longrightarrow}_H y$, providing that a rule $(a_k \to \alpha_i,\ F_{(k,i)}) \in P_k$, $1 \leq k \leq m$, $1 \leq i \leq t_k$, is applied (nondeterministically choosing the positions of symbols a_k to be rewritten in x):

- j times, $j \in F_{(k,i)}(x)$, if

$$|x|_{a_k} \geq \sum_{h=1}^{t_k} l_h,$$

where $(l_1, \ldots, l_{t_k}) \in F_{(k,1)}(x) \times F_{(k,2)}(x) \times \ldots \times F_{(k,t_k)}(x)$ or at most j times, $j \in F_{(k,i)}(x)$ (with competition on symbols if $|P_k| \geq 2$, nondeterministically choosing the rules) otherwise. This type of applying the rules is called the *weak* mode.

- $max(1, j)$ times, $j \in F_{(k,i)}(x)$, if

$$|x|_{a_k} \geq \sum_{h=1}^{t_k} l_h,$$

where $(l_1, \ldots, l_{t_k}) \in F_{(k,1)}(x) \times F_{(k,2)}(x) \times \ldots \times F_{(k,t_k)}(x)$ or at most j times, $j \in F_{(k,i)}(x)$ (with competition on symbols if $|P_k| \geq 2$, nondeterministically choosing the rules) but at least one time if possible, otherwise. This type of applying the rules is called the *strong* mode.

Remark 2.1 *Here we have defined the most general case by considering that multivalued functions, depending on the current state of the system, control the applications of rules. This assertion can be more intuitively understood if we express this mathematical formalism by means of percentages. For a given state of a bio-system (specified by a multiset w), one can predict that a certain rule, say $a \to \alpha$, is to be applied on the multiset w in a rate specified by a value in the interval $(x, y) \subseteq (0, 1)$. Therefore, in that computational step, the rule $a \to \alpha$ is applied a number of times $\lceil x \cdot |w|_a \rceil \leq i \leq \lceil y \cdot |w|_a \rceil$. However, when generalizing this concept, we might assume that there are more than one interval that control the applications of rules, and this bring us to the formalism above introduced.*

The strong mode of derivation imposes the restriction that a rule, if it has all required objects, will be executed at least one time (of course, respecting the competition with other rules).

Definition 2.2 *An M-rate $T0L$ system, denoted by $MTOL$, is a triplet $H = (V, T, \omega)$, where V is a finite alphabet, $T = \{T_1, \cdots, T_k\}$ is a finite set of tables over V, where each table T_i, $1 \leq i \leq k$, is a complete set of CF rules over V, and $\omega \in V^*$ is the axiom. We say that x directly derives y in a $MT0L$ system $H = (V, T, \omega)$, with $x, y \in V^*$, written as $x \overset{MTOL}{\Longrightarrow}_H y$, if $x \overset{OL}{\Longrightarrow}_{H_i} y$ for some i, $1 \leq i \leq k$, with the $0L$ system $H_i = (V, T_i, \omega)$.*

Definition 2.3 *An M-rate $ET0L$ system, denoted by $MET0L$, is a quadruple $H = (V, T, \omega, \Delta)$, where $\overline{H} = (V, T, \omega)$ is an $MT0L$ system, and $\Delta \subseteq V$, $\Delta \neq \emptyset$, is the terminal alphabet. In an $MET0L$ system $H = (V, T, \omega, \Delta)$, x*

directly derives y, *with* $x, y \in V^*$, *written as* $x \overset{METOL}{\Longrightarrow}_H y$, *if* $x \overset{MTOL}{\Longrightarrow}_{\overline{H}} y$. *The transitive and reflexive closure of* $\overset{METOL}{\Longrightarrow}_H$ *is denoted by* $\overset{METOL}{\Longrightarrow}^*_H$. *The generated language of the* $MET0L$ *system* H *(denoted by* $L(H)$*) is* $L(H) = \{w \in \Delta^* \mid \omega \overset{METOL}{\Longrightarrow}^*_H w\}$

Definition 2.4 *An* $M0L$ *(or* $MT0L$, $MET0L$*) system generating the same language independently of the multivalued mappings associated with the rules is called* parallel-free $M0L$ *(or* $MT0L$, $MET0L$ *respectively) system.*

The families of languages generated by $M0L$ (or $MT0L$, $MET0L$) systems working in the strong mode are denoted by $M0L^s$ ($MT0L^s$, $MET0L^s$ respectively).

The families of languages generated by $M0L$ ($MT0L$, $MET0L$ respectively) systems working in the weak mode are denoted by $M0L^w$ ($MT0L^w$, $MET0L^w$ respectively).

When we speak about above mentioned families of languages, we will denote the parallel-free property by adding the superscript pf.

We denote by $U = \{L \mid |L| = 1\}$ the family of all singleton languages.

We have the following results.

Theorem 1 $MET0L^{w,pf} = ME0L^{w,pf} = U \cup \{\emptyset\}$.

Proof. The reason is trivial, because a system working in parallel-free mode means that whatever multivalued mappings associated with rules we choose, the system produces the same output; in addition, because the system works in a weak mode then it means that certain rules might not be applied at all even if there are objects (but not enough) that are within the scope of them. So, one can choose the multivalued mappings in such a manner that the rules cannot be applied at all. Therefore, such systems generate languages containing at most the systems axiom. □

Using a similar argument one can prove that:

Theorem 2 $MT0L^{w,pf} = M0L^{w,pf} = U$.

Example 2.1 *Let* $H = (V, P, \omega)$ *be a* $M0L^s$ *system such that:*

$$\begin{aligned} V &= \{a, b, c\}, \\ P &= \{1 : (a \to aa, F_1(w)) \text{ with } F_1(w) = 0 \text{ for all } w \in V^*, \\ &\quad\ 2 : (b \to bb, F_2(w)) \text{ with } F_2(w) = 0 \text{ for all } w \in V^*, \\ &\quad\ 3 : (c \to cc, F_3(w)) \text{ with } F_3(w) = 0 \text{ for all } w \in V^*\}, \\ \omega &= abc \end{aligned}$$

The system H generates the language $L_H = \{a^n b^n c^n \mid n \geq 1\}$.

Theorem 3 $ME0L^w = MET0L^w \supset RE$.

Proof. The proof is based on the argument that there is no constraint on how the multivalued mappings are chosen. In particular one can choose the mappings such that there exists a system that simulates the moves of a deterministic Turing machine (the computation of the Turing machine is "stored" into the mappings). Moreover, there exist uncomputable multivalued mappings associated to rules such that the system generates a language which is not in RE.

Here is an example of such a system. Let us consider the function $f : M \longrightarrow \{0,1\}$ where $M \subset I\!N$ is an uncomputable set, such that $f(n) = \begin{cases} 1, & n \in M, \\ 0, & n \notin M. \end{cases}$ Obviously, f is an uncomputable mapping.

We define the a $ME0L^w$ system $H = (V, P, \omega, \Delta)$ where $V = \{a, b\}$, $\omega = a$, $\Delta = \{b\}$ and the set P contains the following rules:

$(a \to ab, F_1)$, such that $F_1(w) = 1$ for all $w \in V^*$;

$(a \to b, F_2)$, such that $F_2(w) = f(|w|)$ for all $w \in V^*$.

If the first rule is applied, then the system generates ab^n, $n \in I\!N$. At any moment, nondeterministically, rule $(a \to b, F_2)$ can be selected to be applied. A terminal string is obtained iff the rule is applied and the symbol a is transformed into symbol b. In this way the system H generates the language $\{b^{n+1} \mid n+1 \in M\}$. Consequently, $L(H) \cap b^+ = \{b^n \mid n \in M\} \notin RE$, hence $L(H) \notin RE$. □

Example 2.2 *The language $L = \{a, a^3\}$ is not $MOL^{s,pf}$ (or $MT0L^{s,pf}$) language. This is proved by contradiction as follows. If there exists a $MOL^{s,pf}$ system $H = (V, P, \omega)$ such that $L(H) = \{a, a^3\}$ then, since obviously $V = \{a\}$, we have two cases: (i) $\omega = a$ and $a \Rightarrow a^3$, hence $a^3 \Rightarrow a^k$, $k \neq 1, 3$, therefore a contradiction; (ii) $w = a^3$, hence $a \Rightarrow a$ and $a \Rightarrow \lambda$. Thus $a^3 \Rightarrow a^2$ and so $a^2 \in L(G)$, therefore a contradiction.*

Obviously, the following results stand:

Proposition 1 $MT0L^{s,pf} \subset RE$.

Proposition 2 $M0L^{s,pf} \subset RE$.

Now, we will prove that there exists a class of ET0L systems that are independent of the multivalued mappings associated to the rules and which are able to generate the whole class of ET0L languages.

Theorem 4 $MET0L^{s,pf} = ET0L$.

Proof. We prove this result by double inclusion.

(1) $MET0L^{s,pf} \supseteq ET0L$.
Consider an ET0L system $\widetilde{H} = (\widetilde{V}, \widetilde{T}, \widetilde{\omega}, \widetilde{\Delta})$ such that $\widetilde{T} = \{\widetilde{T_1}, \widetilde{T_2}\}$.

Let $h_1 : \widetilde{V}^* \to \overline{V}^*$ be a morphism such that $h_1(a) = \overline{a}$, $a \in \widetilde{V}$. Also, let $h_2 : \widetilde{V}^* \to \overline{\overline{V}}^*$ be a morphism such that $h_2(a) = \overline{\overline{a}}$, $a \in \widetilde{V}$.

We will simulate the computation of the system $\widetilde{H}$ with an MET0L system $H = (V, T, \omega, \Delta)$ defined as follows.
• $V = \widetilde{V} \cup \{h_1(A), h_2(A) \mid A \in \widetilde{V}\} \cup \{t_1, t_2, t_3, t_4\} \cup \{\#\}$;
• $T = \{T_1, T_2, T_3, T_4\}$, where

$$
\begin{aligned}
T_1 &= \{(A \to h_1(\alpha)t_1, F) \text{ for all } A \to \alpha \in \widetilde{T_1}\} \\
&\cup \{(h_1(A) \to h_1(A), F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(h_2(A) \to \#, F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(\# \to \#, F), (t_1 \to \lambda, F), (t_2 \to \#, F), (t_3 \to \#, F), (t_4 \to \#, F)\}, \\
T_2 &= \{(A \to A, F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(h_1(A) \to At_2, F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(h_2(A) \to \#, F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(\# \to \#, F), (t_1 \to \#, F), (t_2 \to \lambda, F), (t_3 \to \#, F), (t_4 \to \#, F)\}, \\
T_3 &= \{(A \to h_2(\alpha)t_3, F) \text{ for all } A \to \alpha \in \widetilde{T_2}\} \\
&\cup \{(h_1(A) \to \#, F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(h_2(A) \to h_2(A), F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(\# \to \#, F), (t_1 \to \#, F), (t_2 \to \#, F), (t_3 \to \lambda, F), (t_4 \to \#, F)\}, \\
T_4 &= \{(A \to A, F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(h_2(A) \to At_4, F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(h_1(A) \to \#, F) \text{ for all } A \in \widetilde{V}\} \\
&\cup \{(\# \to \#, F), (t_1 \to \#, F), (t_2 \to \#, F), (t_3 \to \#, F), (t_4 \to \lambda, F)\};
\end{aligned}
$$

• $\omega = \widetilde{\omega}$;
• $\Delta = \widetilde{\Delta}$.

Here is how the construction is done. We want to simulate the applications of $\widetilde{H}$ tables; to this aim let us assume, without loosing the generality, that first $\widetilde{T_1}$ is simulated. So, in H, at the beginning, one table is chosen nondeterministically; in case table T_2 or T_4 is chosen, then the computation

will not stop because at least one rule of type $(A \to \#, F)$ will be applied at least one time (recall that our constructed system is working in the strong sense) and therefore the symbol # will never be removed. If table table T_1 (or T_3) is chosen then rules of type $(A \to h_1(\alpha)t_1, F)$ will be executed. However because the parallelism is not necessarily maximal but depends on the multivalued mappings associated to rules, then not necessarily all symbols A from the current sentential form will be rewritten. Now, assume that there are still symbols A not rewritten despite the existence of rules handling symbol A. In addition, remark that there exists at least one symbol from $\overline{V}$. If this is the case, observe that if any other table is chosen for a next application then symbol # is produced (because, for sure there will be applied a rule of type $(A \to \#, F) \in T_2$, $(h_1(A) \to \#, F) \in T_3$, $(t_1 \to \#, F)$ or $(h_1(A) \to \#, F) \in T_4$). Therefore, the only table that can be applied and that does not produce the symbol # is again T_1. Finally, all symbols $A \in V$ will be rewritten by table T_1 and all symbols t_1 will be deleted, and in the sentential form we will have only overlined symbols. At that moment, if we choose to apply any other table except T_2 the symbol # is again produced. In case table T_2 is applied, then with a similar mechanism as before, the system checks whether or not all overlined symbols are rewritten into regular ones. Again, during this checking procedure if we choose to apply another table other than T_2, symbol # is produced. The simulation of the application of table $\widetilde{T_2}$ follows a similar pattern.

Observe that the strong working mode feature is essential because if at a certain moment a wrong table is chosen for application, then we have to be sure that at least one symbol # is produced, hence a rule has to be applied at least once if it can be applied.

It is easy to see that the constructed system generates the same language as the arbitrarily considered ET0L. Consequently, we have that

$$MET0L^{s,pf} \supseteq ET0L$$

(2) $ET0L \supseteq MET0L^{s,pf}$.
In order to prove this inclusion, we will simulate the computation of an arbitrary MET0L^{s,pf} system $\overline{H} = (\overline{V}, \overline{T}, \overline{\omega}, \overline{\Delta})$ with an ETOL system $H = (V, T, \omega, \Delta)$ we construct. First, remark that because the system $\overline{H}$ is parallel-free, then, no matter how the multivalued mappings associated with rules are chosen, the result of the computation is the same. In particular one can associate with all rules of the system the multivalued mappings such that, during the computation, the rules are applied in a total parallel manner (as for L systems).

Therefore H is defined as follows:
- $V = \overline{V}$;
- $T = \{T_1, T_2, \ldots, T_k\}$ providing that $\overline{T} = \{\overline{T_1}, \overline{T_2}, \ldots, \overline{T_k}\}$;
- $\omega = \overline{\omega}$;
- $T_i = \{A \to \alpha \mid (A \to \alpha, F) \in \overline{T_i}, 1 \leq i \leq k\}$;
- $\Delta = \overline{\Delta}$.

Observe that in an ET0L system, when a certain table is applied, if a rule can be applied then it will be applied (of course respecting the nondeterminism if exists). This corresponds to the strong mode of derivation for MET0L systems.

Consequently, we have that $ET0L \supseteq MET0L^{s,pf}$.

□

3 On the Dynamical Parallelism of P Systems

In this section we will extend the definition of another formal bio-inspired model – P Systems. However, we will focus only on a particular model motivated by the cell enzyme activation/deactivation mechanism, namely P systems with promoters/inhibitors.

Definition 3.1 *A P system with* M-rate derivations *(in short, a* P^M *system) of degree* $m \geq 1$*, with catalysts and promoters is a construct*

$$\Pi = (V, C, P, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where:

- V *is an alphabet; its elements are called objects;*

- $C \subseteq V$ *is a distinguished subset of the alphabet, called the set of catalysts;*

- $P \subseteq V$ *is a distinguished subset of the alphabet, called the set of promoters;*

- μ *is a membrane structure consisting of* m *membranes labeled* $1, \ldots, m$*;*

- w_i, $1 \leq i \leq m$, *specify the multiset of objects present in the corresponding regions at the beginning of the computation;*

• R_i, $1 \leq i \leq m$, *are finite sets of evolution rules over* V *associated with regions* $1, 2, \ldots, m$ *of* μ*; we have non-cooperative rules, of the form* $(a \to v,\ F)$*, where* a *is an object from* $V \setminus C$*,* v *is a string over* $\{a_{here}, a_{out} \mid a \in V \setminus C\} \cup \{a_{in_j} \mid a \in V \setminus C, 1 \leq j \leq m\}$*, and* $F : V^* \to 2^{\{0,1,\ldots,|z|_a\}}$*,* $z \in V^*$*, is a partially defined multivalued mapping; catalytic rules* $(ca \to cv,\ F)$*, where* a *is an object from* $V \setminus C$*,* v *is a string over* $\{a_{here}, a_{out} \mid a \in V \setminus C\} \cup \{a_{in_j} \mid a \in V \setminus C, 1 \leq j \leq m\}$ *and* $c \in C$*; promoted rules* $(a \to v|_t,\ F)$ *and* $(ca \to cv|_t,\ F)$*, with* $t \in P$*,* $c \in C$*,* a *is an object from* $V \setminus C$*, and* v *is a string over* $\{a_{here}, a_{out} \mid a \in V \setminus C\} \cup \{a_{in_j} \mid a \in V \setminus C, 1 \leq j \leq m\}$*, and* $F : V^* \to 2^{\{0,1,\ldots,|z|_a\}}$*,* $z \in V^*$ *(when there is no ambiguity on the target,* in_j *is simply written as* in*);*

• $i_0 \in \{0, 1, \cdots, m\}$ *specifies the output region of* Π *(*0 *indicates the environment).*

Similarly as defined for MOL systems in Section 2 we can define the *M-strong rate* and the *M-weak rate* modes of derivation for P systems. The main differences consist on considering multisets instead of strings and the usage of catalysts that represent another "tool" for controlling the degree of the parallelism.

Remark. The definitions of the derivation modes can be extended and generalized also for cooperative (other than catalytic rules) types of rules. One method of doing this is to consider a multifunction that maps each string x a maximal multiset of rules that are applicable to x. In this way, the function determines for each configuration which are the applicable multisets of rules. In addition the whole nondeterminism of the system is embedded in the way the multiset of applicable rules is chosen.

The model of computation that implies a variable parallelism which depends on the current configuration represents an extension of maximal parallel rewriting and it can be adapted to many other rewriting systems (other variants of P systems, Lindenmayer systems, etc).

We use the notation:

$$PsP_m^{\alpha}(\beta, proR), \beta \in \{ncoo, coo\} \cup \{cat_k \mid k \geq 0\},$$

to denote the family of sets of vectors of natural numbers generated by P systems with M–strong mode derivations ($\alpha = MS$), or with M-weak mode derivations ($\alpha = MW$) respectively, having at most m membranes, evolution rules that can be non-cooperative (*ncoo*), cooperative (*coo*), or

catalytic (cat_k), using at most k catalysts, and promoters ($proR$) at the level of rules.

Example 3.1 *Consider the P system* $\Pi = (V, P, C, \mu, w, R, \vartheta)$ *defined as follows:*

$$\begin{aligned}
V &= \{a, p\};\\
P &= \{p\};\\
C &= \emptyset;\\
\mu &= [\ ]_1;\\
w &= a^2p;\\
R &= \{(a \to aaa|_p,\ F_1(w)) \text{ with } F_1(w) = \{[0.5 * |w|_a]\} \text{ for all } w \in V^*),\\
&\quad (p \to p,\ F_2(w)) \text{ with } F_2(w) \text{ arbitrary },\\
&\quad (p \to \lambda,\ F_3(w)) \text{ with } F_3(w) \text{ arbitrary }\};\\
\vartheta &= 1.
\end{aligned}$$

Let us see how this system works. Assume that rule $r_1 : (a \to aaa|_p,\ F_1)$ *is applied* k *times. Then we have:*

$$\begin{aligned}
|w|_a = |w_1|_a &= 2;\\
|w_2|_a &= 6;\\
|w_3|_a &= 14;\\
\cdots\cdots\ &\cdots\ \cdots\cdots\cdots\cdots\\
|w_k|_a &= ([|w_{k-1}|_a * 0.5] + 1) * (right(r_1)\\
&+ |w_{k-1}|_a - ([|w_{k-1}| * 0.5] + 1)\\
&= 2 * (1 + [|w_{k-1}| * 0.5] + |w_{k-1}|_a
\end{aligned}$$

Observe that if $|w_1|_a \vdots 2 \Rightarrow |w_k|_a \vdots 2$, *then this mean that* $[|w_i|_a * 0.5] = |w_i|_a * 0.5$. *We have the following recurrent formulas.*

$$\begin{array}{rcl|l}
|w_k|_a &=& 2 * |w_{k-1}|_a + 2 & *2^0\\
|w_{k-1}|_a &=& 2 * |w_{k-2}|_a + 2 & *2^1\\
\cdots\cdots &\cdots& \cdots\cdots\cdots\cdots & \\
|w_2|_a &=& 2 * |w_1|_a + 2 & *2^{k-2}\\
\hline
&+& &
\end{array}$$

In order to obtain the general term $|w_k|_a$ *we will multiply each recurrent formula by a corresponding constant and we will sum the results. Then, we have:*

$|w_k|_a = 2^{k-1} * |w_1|_a + 2^1 + 2^2 + \cdots + 2^{k-1} = 2 + \cdots + 2^k = \frac{2*(2^k-1)}{2-1} = 2^{k+1} - 2$.

This means that Π *generates the set* $\{2^{k+1} - 2 \mid k \geq 2\}$.

Theorem 5 $PsP_m^{s,pf}(ncoo, inhR) = PsET0L$.

Proof. In [9] it is proved that $PsP_1(ncoo, inhR) = PsET0L$. Moreover, it is easy to see that $PsP_1^{s,pf}(ncoo, inhR) = MET0L^{s,pf}$. Since in Theorem 4 we have shown that $MET0L^{s,pf} = ET0L$, we have that $PsET0L = PsP_1^{s,pf}(ncoo, inhR)$. □

Theorem 6 $PsP_2^{s,pf}(cat_1, proR) = PsRE$.

Proof. In [4] it is proved that $PsP_1(cat_1, proR) = PsRE$. There the inclusion $PsP_1(cat_1, proR) \supseteq PsRE$ was shown by simulating a deterministic register machine in a deterministic manner. The role of the catalyst was to sequentialize when needed the behavior of the P system. Basically, at each time during the computation, one or more rules were executed, but only once at a time (and not maximally rewriting all occurrences of a given object by the same rule).

Therefore, using exactly the same construction, we have that

$$PsP_2^{s,pf}(cat_1, proR) = PsRE$$

□

In [4] was proved that $PsP_1(cat_1, inhR) \supseteq PsRE$ by simulating a deterministic register machine, then the same argument as in the proof of the above theorem can be invoked to show that:

Theorem 7 $PsRE = PsP_2^{s,pf}(cat_1, inhR)$.

4 Conclusions

Starting from natural motivations, we have generalized the concept of maximal parallel rewriting mechanism by considering multivalued functions that control the way the rules are applied during the derivation process. We were interested mainly in finding systems that are independent of such mappings, thus, in a certain sense, tolerant to the degree of parallelism. This concept can be applied to many other parallel rewriting models like DNA computing, Indian parallel grammars, and so on.

Acknowledgements. I would like to thank to all members of the Research Group in Natural Computing for creating such a friendly and stimulating scientific atmosphere in the University of Seville. In addition, I would like to mention Artiom Alhazov for his useful comments and suggestions on the draft of this paper.

References

[1] A. Alhazov, D. Sburlan: Ultimately Confluent Rewriting Systems. Parallel Multiset–Rewriting with Contexts. *Lecture Notes in Computer Science* **3365** (2005), 178–189.

[2] P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg: Membrane Systems with Promoters / Inhibitors. *Acta Informatica* **38** (10) (2002), 695–720.

[3] J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin (1989).

[4] M. Ionescu, D. Sburlan: On P Systems with Promoters / Inhibitors. *JUCS* **10** (5) (2004), 581–599.

[5] L. Bianco, F. Fontana, V. Manca: Evolution and Oscillation in P Systems. *Lecture Notes in Computer Science* **3365** (2005), 63–84.

[6] Gh. Păun: *Membrane Computing: An Introduction*. Springer-Verlag, Berlin (2002).

[7] G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems*. Academic Press, New York (1980).

[8] G. Rozenberg, A. Salomaa, Eds.: *Handbook of Formal Languages*. Springer-Verlag, Berlin (1997).

[9] D. Sburlan: Further Results On P Systems with Promoters / Inhibitors. *International Journal of Foundation of Computer Science*, to appear.

Non-Cooperative P Systems with Priorities Characterize PsET0L

Dragoş SBURLAN

Department of Computer Science and A.I.
University of Seville,
Reina Mercedes Ave., Seville, Spain

Faculty of Mathematics and Informatics
Ovidius University of Constantza
124 Mamaia Ave., Constantza, Romania
E-mail: dsburlan@univ-ovidius.ro

Abstract

The paper answers an open problem from [4], proving that transition P systems with non-cooperative rules using priorities generate exactly the Parikh images of ET0L languages.

1 Introduction

The classical model of P systems with priorities has been introduced in [3] and since then the field of membrane computing has growth considerably, nowadays becoming more and more a framework for expressing various phenomena occurring in cells.

The model of P systems with priorities was initially used to describe the biochemical reactions occurring in the cell. There, priority relations (in the form of a partial order relation) among the rules from each region expressed the following phenomena: if a biochemical reaction r_1 is more active than a reaction r_2 and it consumes a given resource (energy for example) from the region, the reaction r_2 cannot take place despite the availability of all necessary input objects.

In the attempt to make use of these features to design new bio-inspired computational devices, the current trend was to decrease as much a possible the level of cooperation between the objects participating into the rules

while maintaining the currently obtained results. The mathematical interest was also the opposite problem, namely to see which is the upper bound of cooperation such that the systems are not anymore universal, knowing that almost for all P systems variants the universality results were reached.

This bring us to the topic of the paper – it determines the computational power of the classical P system model with strong priorities when only non-cooperative rules are being used.

We assume the reader familiar with basic notions of P systems with priorities among rules and P systems with promoters/inhibitors at the level of rules. For more details regarding these topics we refer to [1], [2] and [7]. In addition, we assume known the basic theory and results of Lindenmayer systems (see [5] and [6] for details).

We will denote by $PsP_m(ncoo, \alpha)$, $\alpha \in \{pri, inh\}$ the family of sets of vectors of numbers, computed by P systems of degree at most m, $m \geq 1$, using non-cooperative rules and priorities among rules ($\alpha = pri$) or inhibitors at the level of rules ($\alpha = inh$).

2 Some Known Results

In [7] was proved that P systems with non-cooperative rules and inhibitors at the level of rules generates exactly the $PsET0L$, the family of Parikh images of ET0L languages. For the sake of clarity, we sketch the original proof of the inclusion $PsET0L \supseteq PsP(ncoo, inh)$, pointing out some relevant details for the present work.

Here are the outlines of the proof:

- First we have shown the equivalence between P systems with non-cooperative inhibited rules using m membranes, and P systems with non-cooperative inhibited rules and only one membrane with a similar construction as will be presented in the proof of Lemma 3.1.

- We have shown that any P system with non-cooperative inhibited rules is equivalent with a P system with non-cooperative inhibited rules, one region and having the alphabet made out of two disjoint sets, the set of terminals and of non-terminals; in addition, all the rules have a non-terminal on their left-hand side; moreover, the set is complete, i.e. for each symbol in the nonterminal alphabet there exists at least one rule having it on the left-hand side.

- For a given set of inhibited rules, we have defined *saturated* classes

of rules, i.e, we have found the sets containing rules that does not mutually forbids each other.

Let V be an alphabet and $R = \{r_1, r_2, \ldots, r_k\}$ a set of rules over V, of the form $r_i : (A_i \to \alpha_i|_{\neg B_i})$, $A_i, B_i \in V$, $A_i \neq B_i$, $\alpha_i \in V^*$, $1 \leq i \leq k$. For a rule $r : (A \to \alpha|_{\neg B}) \in R$ let us define $left(r) = A$ and $inh(r) = B$.

Two rules $r_i, r_j \in R$ are said to be in the *non-excluding inhibiting relation*, and we denote this by $r_i \equiv_{nei} r_j$, iff $left(r_i) \neq inh(r_j)$ and $left(r_j) \neq inh(r_i)$.

A subset $W \subseteq R$ is said to be *saturated* (or *complete*) with respect to non-excluding inhibiting relation $\equiv_{nei}$ iff $(\forall)$ $r_i, r_j \in W$, $r_i \equiv_{nei} r_j$, and $(\forall)$ $r_i \in R \setminus W$, $(\exists)$ $r_j \in W$ such that $r_i \not\equiv_{nei} r_j$.

- We have constructed an ET0L system $H = (V, T, \omega, \Delta)$, with $T = \{T_1, T_2, \ldots, T_k\}$, having as tables all the saturated sets $\{\overline{T_1}, \overline{T_1}, \ldots, \overline{T_k}\}$ (but with rules without inhibiting conditions and, in addition, with some other rules as will be explained later). Remark that from the way we have defined the saturated subsets, the conditions on the rules can be omitted (observe that two rules $r_1 : (a_1 \to \alpha_1|_{\neg b_1}$ and $r_2 : (a_2 \to \alpha_2|_{\neg b_2})$ can simultaneously rewrite symbols a_1 and a_2 iff $b_1 \neq a_2$ and $a_1 \neq b_2$) in case we divide them in different tables. In addition, we have added to each table all context-free rules of the P system that does not violate the saturation relation considered for the table. We also have added rules of the type $b \to \#$ if rules $\{a \to \alpha \mid a \to \alpha|_{\neg b} \in \overline{T_i}\} \in T_i$ and $\# \to \#$; in this way we have assured that if we have chosen the "wrong" table, the computation will never stop since the # is produced (and therefore $\# \to \#$ will always be executed no matter which table is chosen).

In [4] was shown in a straightforward manner that $PsP_1(ncoo, pri) \supseteq PsET0L$ by simulating with a constructed P system with non-cooperative rules and priorities the computation of an arbitrary two table ET0L system H. In addition, catalytic P systems with priorities proved to be universal when only one catalyst is used. The remaining open problem ($Q2$ in [4]) was whether or not non-cooperative systems with priorities are universal. Here we deal with this problem.

3 P Systems with Priorities – New Results

P systems with priorities equals in generative power the class of $PsET0L$ languages. In the proof of this result we will need the notions of P systems with inhibitors (see [1] for the introductory paper on this topic).

The following lemma shows that P systems with non-cooperative rules and priorities, having only one membrane equals in computational power the ones having the same features, but with $m > 1$ membranes.

Lemma 3.1 $PsP_m(ncoo, pri) = PsP_1(ncoo, pri), m \geq 1$.

Proof. The inclusion $PsP_m(ncoo, pri) \supseteq PsP_1(ncoo, pri)$ is trivial. For the proof of the inclusion $PsP_m(ncoo, pri) \subseteq PsP_1(ncoo, pri)$, we construct a P system $\Pi_1 = (V, C, \mu, w, R, \vartheta)$ that simulates the computation of P system $\overline{\Pi_m} = (\overline{V}, \overline{C}, \overline{\mu}, \overline{w_1}, \ldots, \overline{w_m}, \overline{R_1}, \ldots, \overline{R_m}, \overline{\vartheta})$ in the following way.

First, denote by $\mathcal{L} = \{1, 2, \ldots, m\}$ the set of labels of the regions in $\overline{\Pi_m}$. Then, we define:

- $V = \{a_i \mid a \in \overline{V}, i \in \mathcal{L}\}$.
- $C = \overline{C} = \emptyset$;

Let $h : \overline{V}^* \times \mathcal{L} \rightarrow V^*$ be a mapping such that

1) $h(a, i) = a_i, a \in \overline{V}, i \in \mathcal{L}$;
2) $h(\lambda, j) = \lambda$, $\forall j \in \mathcal{L}$;
3) $h(x_1x_2, j) = h(x_1, j)h(x_2, j)$, $x_1, x_2 \in \overline{V}^*$, $j \in \mathcal{L}$.

- denote by $w = h(\overline{w_1})h(\overline{w_2}) \ldots h(\overline{w_m})$, where $\overline{w_i}$ is the multiset present in region $i \in \mathcal{L}$ of $\overline{\Pi_m}$ at the beginning of the computation.
- R is defined as follows. For each rule $a \rightarrow \alpha \in \overline{R_i}$, $a \in \overline{V}$, α is a string over $\{c, c_{out}, c_{in} \mid c \in \overline{V}\}$, $i \in \mathcal{L}$, we add to R the rule $h(a, i) \rightarrow \alpha'$ where α' is the corresponding string over $\{h(c, i), h(c, j), h(c, k) \mid c \in \overline{V}, i, j, k \in \mathcal{L}\}$, j being the label of the outer region of i, and k being the label of an inner region of i. In addition, we inherit the existing priority relations among the rules.
- $\vartheta = 1$;

In other words, for the P system with a single region that simulates a P system with m regions, we have encoded the regions labels into objects (the subscript associated to an object indicates the region where the corresponding object belongs) and we have expressed the rules of regions by the corresponding encoded objects. In this way we ensured that, when simulating $\overline{\Pi_m}$ with Π_1, both the parallelism at the level of regions and at the level of whole system $\overline{\Pi_m}$ is respected. In addition, one can remark that whenever

$\overline{\Pi_m}$ halts, Π_1 halts as well. Moreover, when Π_1 halts, we will have in the output region of Π_1 all the objects corresponding to the multisets present in all regions of $\overline{\Pi_m}$.

However, in the output multiset w_{Π_1} of Π_1 we can distinguish the output multiset $w_{\overline{\Pi_m}}$ of $\overline{\Pi_m}$ because we know which are the objects corresponding to the output region of $\overline{\Pi_m}$ (they are the objects that have as index $\overline{\vartheta}$). Therefore, we have to delete the unnecessary objects that remain in the output region of Π_1 in a halting configuration since we want to show that Π_1 and $\overline{\Pi_m}$ generate exactly the same set of vectors of numbers. We will modify the rules presented above in the following manner.

We add to the vocabulary V a new symbol D (the object D stands for the "deletion command") and we replace each rule $a_i \to \alpha' \in R$ by

$a_i \to \alpha' D \in R$,

of course, maintaining the priority relations among the rules. In addition, we add the following rules (with the corresponding priority relation)

$\boxed{D \to \lambda} > \boxed{a_i \to \lambda}$, for all $a_i \in V$, $i \neq \overline{\vartheta}$

One can remark that in this way we produce at each computational step at least one object D and also, in the same time, we delete the already existing object(s) D. If there exist rules that can be executed (i.e. there will be objects D) rules of type $a_i \to \lambda$ cannot be applied because they are locked according to the priority relations. When the computation halts, objects D are not produced anymore, and so, the deletion rules can start and erase the remaining unnecessary objects. Consequently, we have shown that both systems generate the same family of vectors of natural numbers, hence we have $PsP_m(ncoo, pri) \subseteq PsP_1(ncoo, pri)$.

Finally, by the double inclusion, we have proved that $PsP_m(ncoo, pri) = PsP_1(ncoo, pri)$. □

As a consequence of the above result we can state the following

Corollary 1 *For any P system Π with non-cooperative rules, using priorities, there exists an equivalent P system Π' with non-cooperative rules, using priorities such that, for any halting configuration of Π', all regions of Π', excepting the output one, are empty.*

Now, we can prove the following result:

Theorem 1 $PsP_m(ncoo, pri) = PsP_m(ncoo, inh) = PsET0L$.

Proof. In [4] was shown that $PsP(ncoo, pri) \supseteq PsET0L$. In cite [7] was shown that $PsP(ncoo, inh) = PsET0L$ following a procedure as the one roughly described in Section 1. Here we will show that $PsP(ncoo, inh) \supseteq PsP(ncoo, pri)$ and hence, $PsP(ncoo, pri) = PsET0L$. Here is how we proceed.

Let us consider an arbitrary P system $\widetilde{\Pi}$ with m membranes, non-cooperative rules and with a priority relations among rules. According to Lemma 3.1 we know that we can construct an equivalent P system $\overline{\Pi} = (\overline{V}, \overline{C}, \overline{\mu}, \overline{w}, \overline{R}, \overline{\vartheta})$ where:
- $\overline{V} = \{X_1, X_2, \ldots, X_r\}$;
- $\overline{C} = \emptyset$;
- $\overline{\mu} = [\,]_1$;
- $\overline{w} \in \overline{V}^*$;
- The set $\overline{R}$ is defined by the sequences of rules:

$$\boxed{X_{(1,1)} \to \alpha_{(1,1)}} > \boxed{X_{(1,2)} \to \alpha_{(1,2)}} > \cdots > \boxed{X_{(1,k_1)} \to \alpha_{(1,k_1)}}$$

$$\ldots$$

$$\boxed{X_{(p,1)} \to \alpha_{(p,1)}} > \boxed{X_{(p,2)} \to \alpha_{(p,2)}} > \cdots > \boxed{X_{(p,k_p)} \to \alpha_{(p,k_p)}}$$

with $X_{(i,j)} \in \overline{V}$, such that $X_{(i,j_1)} \neq X_{(i,j_2)}$, for all $j_1 \neq j_2$, $1 \leq i \leq p$ and $\alpha_{i,j} \in \overline{V}^*$, $1 \leq i \leq p$, $1 \leq j \leq k_i$. In addition, without loosing the generality, we will assume that $k_1 \geq k_2 \geq \ldots \geq k_p$.

Recall that we assumed that $X_{(i,j_1)} \neq X_{(i,j_2)}$, for all $j_1 \neq j_2$, $1 \leq i \leq p$ because in case $X_{(i,j_1)} = X_{(i,j_2)}$, the rule $X_{(i,j_2)} \to \alpha_{(i,j_2)}$ will never be applied since the rule $X_{(i,j_1)} \to \alpha_{(i,j_1)}$, having a grater priority, is applied first (of course, if it fulfills all required conditions).

We construct a P system $\Pi = (V, C, \mu, w, R, \vartheta)$ with non-cooperative inhibited rules that simulates the moves of $\overline{\Pi}$ and which is defined as follows.

- $V = \overline{V} \cup \{\overline{X} \mid X \in \overline{V}\} \cup \{A_{(i,j)}, U_{(i,j)} \mid 1 \leq i \leq p, 1 \leq j \leq k_i\}$
  $\cup \{S, T, H, \#\} \cup \{W_i \mid 1 \leq i \leq k_1 + 1\}$;
- $C = \emptyset$;
- $\mu = [\,]_1$;
- $w = STH\overline{w}$;

- The set of rules R is defined as follows:

$\diamond$ we add to R the rules:

$X_i \to \overline{X_i} T A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$, $1 \leq i \leq k$

$S \to U_{(1,0)} U_{(2,0)} \dots U_{(p,0)} W T H A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$

$W \to W_1 T H A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$

$W_1 \to W_2 T H A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$

$\dots$

$W_{k_1} \to W_{k_1+1} T H A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$

$W_{k_1+1} \to SH$

$T \to \lambda$

$A_{(i,j)} \to \lambda$, $1 \leq i \leq p, 1 \leq j \leq k_i$.

$\diamond$ for each sequence of rules belonging to $\overline{R}$:

$\boxed{X_{(i,1)} \to \alpha_{(i,1)}} > \boxed{X_{(i,2)} \to \alpha_{(i,2)}} > \dots > \boxed{X_{(i,k_i)} \to \alpha_{(i,k_i)}}$

we add to R the rules:

$U_{(i,0)} \to U_{(i,1)}|_{\neg \overline{X_{(1,1)}}}$

$U_{(i,1)} \to U_{(i,2)}|_{\neg \overline{X_{(1,2)}}}$

$\dots$

$U_{(i,k_i)} \to U_{(i,k_i+1)}|_{\neg \overline{X_{(1,k_1)}}}$

$U_{(i,0)} \to A_{(i,2)} A_{(i,3)} \dots A_{(i,r)}|_{\neg T}$

$U_{(i,1)} \to A_{(i,1)} A_{(i,3)} A_{(i,4)} \dots A_{(i,r)}|_{\neg T}$

$\dots$

$U_{(i,k_i)} \to A_{(i,1)} \dots A_{(i,k_1)} A_{(i,k_1+2)} \dots A_{(i,r)}|_{\neg T}$

$U_{(i,k_i+1)} \to A_{(i,1)} \dots A_{(i,r)}|_{\neg T}$

$\overline{X_{(i,j)}} \to \alpha_{(i,j)}|_{\neg A_{(i,j)}}$, $1 \leq j \leq k_i$

$\diamond$ also, we add the rules:

$S \to \lambda$

$X_i \to \#|_{\neg H}$ iff there exits a rule $X_i \to \alpha_i \in R$

$\# \to \#$

$H \to \lambda$

$\overline{X_{(i,j)}} \to X_{(i,j)}|_{\neg H}$,

Let us see how the simulation works. First, observe that (as a general technique) when we want to execute a certain non-cooperative rule r at a certain moment during the computation, then we might activate it using an inhibitor; however, this means that all the time during the computation we have to generate the symbol representing the inhibitor, to delete at each

step all previously created inhibitors, and only when we actually want to execute r we omit its generation.

We start the computation by executing the rule:

$X_i \to \overline{X_i} T A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$, $1 \leq i \leq k$

This rule is responsible for "painting" all objects X_i that correspond to objects in $\overline{V}$. In the same time we create the objects:

$$A_{(1,1)}, A_{(1,2)}, \dots, A_{(1,k_1)}, \dots\dots, A_{(p,1)}, \dots, A_{(p,k_p)},$$

$1 \leq i \leq k$ that will be used as "flags", indicating which rules cannot be applied (here the simulation of any rule from $\overline{\Pi}$ is forbidden – all objects are present). In addition, we create the object T that represents as well a flag, its role being to indicate when the selected rules will be effectively applied.

In the same time, the rule

$S \to U_{(1,0)} U_{(2,0)} \dots U_{(k,0)} W T H A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$

is executed. All objects $U_{(i,0)}$, $1 \leq i \leq p$ represent the starting points for the sequences of rules of type:

$U_{(i,0)} \to U_{(i,1)}|_{\neg\overline{X_{(1,1)}}}$

$U_{(i,1)} \to U_{(i,2)}|_{\neg\overline{X_{(1,2)}}}$

$\dots$

$U_{(i,k_i)} \to U_{(i,k_i+1)}|_{\neg\overline{X_{(1,k_1)}}}$

Such a sequence (that correspond to $X_{(i,1)} \to \alpha_{(i,1)} > \dots > X_{(i,k_i)} \to \alpha_{(i,k_i)} \in \overline{R}$ is used to check which rules from $\overline{R}$ can be applied. Depending where this sequence stops we will know what rules we have to apply. This information will be stored in the objects $U_{(i,j}$.

Remark that along with objects $U_{(i,0)}$, $1 \leq i \leq p$ it is produced the object W. This object will be used by the cycle (let us call it the "waiting" cycle):

$W \to W_1 T H A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$

$W_1 \to W_2 T H A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$

$\dots$

$W_{k_1} \to W_{k_1+1} T H A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$

$W_{k_1+1} \to SH$

which it produce "enough" time (more than the maximum length of the sequences of rules in $\overline{R}$) objects $A_{(1,1)} A_{(1,2)} \dots A_{(1,k_1)} \dots\dots A_{(p,1)} \dots A_{(p,k_p)}$ which forbids the application of any rule that corresponds to a rule in $\overline{R}$. In the last step of the cycle we omit the creation of object T. The absence of object T means that we can apply one of the rules:

$U_{(i,0)} \rightarrow A_{(i,2)}A_{(i,3)} \dots A_{(i,r)}|_{\neg T}$
$U_{(i,1)} \rightarrow A_{(i,1)}\, A_{(i,3)}A_{(i,4)} \dots A_{(i,r)}|_{\neg T}$
$\dots$
$U_{(i,k_i)} \rightarrow A_{(i,1)} \dots A_{(i,k_1)}\, A_{(i,k_1+2)} \dots A_{(i,r)}|_{\neg T}$
$U_{(i,k_i+1)} \rightarrow A_{(i,1)} \dots A_{(i,r)}|_{\neg T}$

In this way we are able to select which are the rules (that corresponds to rules in $\overline{R}$) that can be applied, namely:

$\overline{X_{(i,j)}} \rightarrow \alpha_{(i,j)}|_{\neg A_{(i,j)}}$, $1 \leq j \leq k_i$

Now, observe that all the time the "waiting" cycle is active (that is, we intend to make a simulation of a step in $\overline{\Pi}$) the object H is created. Also, the already existing objects H are deleted by the rule $H \rightarrow \lambda$. This object will help us to finish the simulation. Here are the details.

Nondeterministically, object S might also be deleted by the rule $S \rightarrow \lambda$. If this happen, then the object H is not produced anymore and so, the rules $\overline{X_{(i,j)}} \rightarrow X_{(i,j)}|_{\neg H}$ and $X_{(i,j)} \rightarrow \#|_{\neg H}$ are executed. So, basically, if symbol $\#$ appears, then the computation will not stop because the rule $\# \rightarrow \#$ will be always executed.

In case th symbol $\#$ is not produced then the computation eventually stops if the computation of $\overline{\Pi}$ stops. This is due to the fact that the cycle involving object S might be always executed. However, the system Π will generate in a nondeterministic manner (if object S is deleted and there is no symbol $\#$) the same language as $\overline{\Pi}$. Consequently, the families of languages generated by these types of systems are equal.

Therefore, we have that

$$PsP_m(ncoo, pri) = PsP_m(ncoo, inh) = PsET0L.$$

□

4 Concluding Remarks

Here we have proved that P systems with strong priorities generate the same class of languages as $PsET0L$. Recall now that in the sequential case, forbidden random context grammars equals in computational power ordered grammars. From this perspective, the result $PsP(ncoo, pri) = PsP(ncoo, inh) = PsET0L$ surprises also because here the maximal parallelism proved not to influence the equality between the classes of languages generated by systems with priorities and with inhibitors.

In addition, the exact equality with the class of languages generated by ET0L systems gives us "for free" all decidability and properties results. For

example, knowing that $ET0L$ is a AFL, then the Corollary 1 becomes trivial since after the encoding the membrane structure into the objects (as we did) we can argue that such systems are closed under arbitrary morphism (so we can delete the unnecessary objects).

In addition, the equality with the class of Parikh images of ET0L languages gives us "for free" all decidability known for the family of ET0L languages. For example, it is of a mathematical interest (but not only) to mention that reachability and membership problems for ET0L systems are decidable.

Acknowledgements. I would like to thank to all members of the Research Group in Natural Computing for creating such a friendly and stimulating scientific atmosphere in the University of Seville.

References

[1] P. Bottoni, C. Martín-Vide, Gh. Păun, G. Rozenberg: Membrane Systems with Promoters / Inhibitors. *Acta Informatica* **38** (10) (2002), 695–720.

[2] M. Ionescu, D. Sburlan: On P Systems with Promoters / Inhibitors. *Journal of Universal Computer Science* **10** (5) (2004), 581–599.

[3] Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences* **61** (1) (2000), 108–143.

[4] Gh. Păun: *Membrane Computing: An Introduction*. Springer–Verlag, Berlin (2002).

[5] G. Rozenberg, A. Salomaa: *The Mathematical Theory of L Systems*. Academic Press, New York (1980).

[6] G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages*. Springer-Verlag, Berlin (1997).

[7] D. Sburlan: Further Results on P Systems with Promoters / Inhibitors. *Intern. J. Found. Computer Sci.*, to appear.