Fifth Brainstorming Week on Membrane Computing

Sevilla, January 29–February 2, 2007

Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Alvaro Romero-Jiménez, Agustín Riscos-Núñez

Editors

Fifth Brainstorming Week on Membrane Computing

Sevilla, January 29–February 2, 2007

Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Alvaro Romero-Jiménez, Agustín Riscos-Núñez

Editors

RGNC REPORT 01/2007

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2007

ISBN: 978-84-611-6776-0
Depósito Legal: SE-2563–07
Edita: Fénix Editora
C/ Patricio Sáenz 13, 1 B
41004 Sevilla
fenixeditora1@telefonica.net
Telf. 954 41 29 91

Preface

This volume contains most of the papers emerged from the Fifth Brainstorming Week on Membrane Computing (BWMC), held in Sevilla, from January 29 to February 2, 2007, in the organization of the Research Group on Natural Computing from the Department of Computer Science and Artificial Intelligence of Sevilla University. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and the next three editions took place in Sevilla at the beginning of February 2004, February 2005, and February 2006, respectively.

Keeping the tradition of previous meetings in this series, the fifth BWMC was conceived as a period of active interaction among the participants, with the emphasis on exchanging ideas and cooperation, hence with only a few ("provocative") presentations scheduled in the first days of the meeting and with most of the time devoted to the joint work. The efficiency of this type of meetings was again proved to be very high and the present volume proves this assertion.

Pleasantly enough, after almost eight years since the research in this area was initiated, membrane computing is still an expanding research field, both with many theoretical issues of current interest, new problems and new research ideas, and with a growing number of applications, especially in biology and medicine.

The papers included in this volume, arranged in the alphabetic order of the authors, were collected in the form available at a short time after the brainstorming; several of them are still under elaboration. The idea is that the proceedings are a working instrument, part of the interaction started during the stay of authors in Sevilla, meant to make possible a further cooperation, this time having a written support.

A selection of the papers from these volumes will be considered for publication in a special issues of *International Journal of Unconventional Computing* (after the first BWMC, a special issue of *Natural Computing* – volume 2, number 3, 2003, and a special issue of *New Generation Computing* – volume 22, number 4, 2004, were published; papers from the second BWMC have appeared in a special issue of *Journal of Universal Computer Science* – volume 10, number

5, 2004, as well as in a special issue of *Soft Computing* – volume 9, number 5, 2005; a selection of papers written during the third BWMC have appeared in a special issue of *International Journal of Foundations of Computer Science* – volume 17, number 1, 2006); after the fourth BWMC a special issue of *Theoretical Computer Science* was edited – volume 372, numbers 2-3, 2007. Other papers elaborated during the fifth BWMC will be submitted to other journals or to suitable conferences. The reader interested in the final version of these papers is advised to check the current bibliography of membrane computing available in the Milano web page `http://psystems.disco.unimib.it` (with a mirror in China, at `http://bmc.hust.edu.cn/psystems`).

The list of participants as well as their email addresses are given below, with the aim of facilitating the further communication and interaction:

1. Ardelean Ioan, Institute of Biology of the Romanian Academy, Bucharest, Romania,
   `ioan.ardelean@ibiol.ro`
2. Balbontín-Noval Delia, University of Sevilla, Spain,
   `delia@us.es`
3. Bernardini Francesco, Leiden University, The Netherlands,
   `bernardi@liacs.nl`
4. Bianco Luca, University of Verona, Italy,
   `bianco@sci.univr.it`
5. Busi Nadia, University of Bologna, Italy,
   `busi@cs.unibo.it`
6. Cazzaniga Paolo, University of Milano-Bicocca, Italy,
   `cazzaniga@disco.unimib.it`
7. Colomer Cugat M. Angels, University of Lleida, Spain,
   `Colomer@matematica.UdL.es`
8. Cordón Franco Andrés, University of Sevilla, Spain,
   `acordon@us.es`
9. Díaz-Pernil Daniel, University of Sevilla, Spain,
   `sbdani@us.es`
10. Ferretti Claudio, University of Milano-Bicocca, Italy,
    `ferretti@disco.unimib.it`
11. Freund Rudolf, Technical University Wien, Austria,
    `rudi@emcc.at, rudi@logic.at`
12. García Arnau Marc, Polytechnical University of Madrid, Spain,
    `mgarnau@gmail.com`
13. Gheorghe Marian, University of Sheffield, United Kingdom,
    `marian@dcs.shef.ac.uk`
14. Graciani Díaz Carmen, University of Sevilla, Spain,
    `cgdiaz@us.es`
15. Gutiérrez-Naranjo Miguel Angel, University of Sevilla, Spain,
    `magutier@us.es`

16. Ionescu Mihai, University of Tarragona, Spain,
mihai_caltun@yahoo.com
17. Leporati Alberto, University of Milano-Bicocca, Italy,
leporati@disco.unimib.it
18. Mardare Radu, University of Trento, Italy,
mardare@cosbi.eu
19. Mauri Giancarlo, University of Milano-Bicocca, Italy,
mauri@disco.unimib.it
20. Nepomuceno-Chamorro Isabel, University of Sevilla, Spain,
isabelnepomuc@yahoo.es
21. Nepomuceno-Chamorro Juan Antonio, University of Sevilla, Spain,
janepochamorro@yahoo.es
22. Păun Gheorghe, Institute of Mathematics of the Romanian Academy, Bucharest, Romania, and University of Sevilla, Spain,
george.paun@imar.ro, gpaun@us.es
23. Pérez-Jiménez Mario de Jesus, University of Sevilla, Spain,
marper@us.es
24. Pescini Dario, University of Milano-Bicocca, Italy,
pescini@disco.unimib.it
25. Potapov Igor, University of Liverpool, United Kingdom,
igor@csc.liv.ac.uk
26. Ramirez Martinez Daniel, Univ. of Sevilla, Spain,
thebluebishop@gmail.com
27. Riscos-Núñez Agustín, University of Sevilla, Spain,
ariscosn@us.es
28. Romero-Campero Francisco José, University of Sevilla, Spain,
fran@us.es
29. Romero Jiménez Alvaro, University of Sevilla, Spain,
Alvaro.Romero@cs.us.es
30. Sancho Caparrini Fernando, University of Sevilla, Spain,
fsancho@us.es
31. Sburlan Dragoş, Constanţa University, Romania,
dsburlan@gmail.com
32. Sempere Luna José María, Polytechnical University of Valencia, Spain,
jsempere@dsic.upv.es
33. Vaszil György, Hungarian Academy of Sciences, Budapest, Hungary,
vaszil@sztaki.hu
34. Verlan Sergey, University of Paris 12, Creteil, France,
verlan@univ-paris12.fr
35. Yokomori Takashi, Waseda University, Japan,
yokomori@waseda.jp
36. Zandron Claudio, University of Milano-Bicocca, Italy,
zandron@disco.unimib.it

As mentioned above, the meeting was organized by the Research Group on Natural Computing from Sevilla University (`http://www.gcn.us.es`)– and all the members of this group were enthusiastically involved in this (not always easy) work. The meeting was supported from various sources: (i) Proyectos de Investigación TIN2005–09345–C04–01 y TIN2006–13425 del Ministerio de Educación y Ciencia of Spain, (ii) Grupo de Investigación en Computación Natural (PAI TIC 193) de Junta de Andalucia, (iii) III Plan Propio de la Universidad de Sevilla, as well as by the Department of Computer Science and Artificial Intelligence from Sevilla University.

Gheorghe Păun
Mario de Jesús Pérez-Jiménez
(Sevilla, April 10, 2007)

Contents

Partial Versus Total Halting in P Systems

Artiom Alhazov[1], Rudolf Freund[2], Marion Oswald[2], Sergey Verlan[3]

[1] Department of Information Technologies
Abo Akademi University
Turku Center for Computer Science
FIN-20520 Turku, Finland
aalhazov@abo.fi
and
Institute of Mathematics and Computer Science
Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD-2028, Moldova
aartiom@math.md

[2] Faculty of Informatics
Vienna University of Technology
Favoritenstr. 9, 1040 Vienna, Austria
{rudi,marion}@emcc.at

[3] LACL, Département Informatique
UFR Sciences et Technologie
Université Paris XII
61, av. Général de Gaulle
94010 Créteil, France
verlan@univ-paris12.fr

Summary. We consider a new variant of the halting condition in P systems, i.e., a computation in a P system is already called halting if not for all membranes a rule is applicable anymore at the same time, whereas usually a computation is called halting if no rule is applicable anymore in the whole system. This new variant of partial halting is especially investigated for several variants of P systems working in different derivation modes.

1 Introduction

In the seeding papers of Gheorghe Păun (e.g., see [19], [9]) introducing membrane computing, membrane systems were introduced as systems with a hierarchical (tree-like) structure and the rules being applied in a maximally parallel manner; the results were taken as the contents of a specified output membrane in the final configurations of halting computations, i.e., at the end of computations to which no rule was applicable anymore. In this paper, we investigate a new variant of halting – *partial halting* –, see [13], i.e., we consider a computation to halt as soon

as not for all membranes a rule is applicable anymore at the same time. Moreover, we especially also consider the derivation mode of minimal parallelism (e.g., see [7]), i.e., for each membrane, at least one rule – if possible – has to be applied, but it is not required to use a maximal multiset of rules. Finally, in the asynchronous derivation mode an arbitrary number of rules can be applied in parallel, and in the sequential derivation mode exactly one rule has to be applied in each computation step.

The paper is organized as follows: We first recall some well-known definitions, notions, and results for matrix grammars and register machines and then define a special model of P systems – *P systems with permitting contexts* – that covers a lot of variants known from the literature such as antiport P systems, P systems with conditional uniport rules, evolution/communication P systems, and P systems with boundary rules. For establishing our results, we especially consider the new stopping mode of partial halting; we first state some general result and then show that P systems using membrane rules with permitting contexts working in the minimally parallel mode and with partial halting can only generate matrix languages. On the other hand, we improve or newly establish results showing that specific variants of P systems with permitting contexts such as antiport P systems, P systems with conditional uniport rules, and evolution/communication P systems together with the newly introduced variant of minimal parallelism and with total halting are computationally complete.

2 Definitions

In this section, we first recall some basic notions and notations and then give precise definitions for matrix grammars, register machines, and a general model of P systems using membrane rules with permitting contexts as they are considered in this paper; moreover, we show how several well-known models of P systems (P systems with symport/antiport rules, P systems with conditional uniport rules, evolution/communication P systems, P systems with boundary rules) can be interpreted as special variants of this general model.

2.1 Preliminaries

For the basic elements of formal language theory needed in the following, we refer to [8] and [23]. We just list a few notions and notations: $\mathbb{N}$ denotes the set 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 NRE and $NREG$ we denote the family of recursively enumerable sets and regular sets of non-negative integers, respectively.

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. For a family of languages F, the family of Parikh images of languages in F is denoted by PsF. A (finite) multiset $\langle m_1, a_1\rangle \ldots \langle m_n, a_n\rangle$ 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 $(m_1, ..., m_n)$; if $m_i = 1$ for all $1 \leq i \leq n$, then x can also be represented by the corresponding set $\{m_1, ..., m_n\}$.

The family of recursively enumerable languages is denoted by RE, the family of context-free and regular languages by CF and REG, respectively. The corresponding families of languages over a k-letter alphabet are denoted by $X(k)$, $X \in \{RE, CF, REG\}$; for $k = 1$ we obtain $PsX(1) = NX$ and, moreover, $NREG = NCF$.

2.2 Matrix Grammars

A context-free *matrix grammar* (without appearance checking) is a construct $G = (N, T, S, M)$ where N and T are sets of *non-terminal* and *terminal symbols,* respectively, with $N \cap T = \emptyset$, $S \in N$ is the *start symbol*, M is a finite set of *matrices*, $M = \{m_i \mid 1 \leq i \leq n\}$, where the matrices m_i are sequences of the form $m_i = (m_{i,1}, \ldots, m_{i,n_i})$, $n_i \geq 1$, $1 \leq i \leq n$, and the $m_{i,j}$, $1 \leq j \leq n_i$, $1 \leq i \leq n$, are context-free productions over (N, T). For $m_i = (m_{i,1}, \ldots, m_{i,n_i})$ and $v, w \in (N \cup T)^*$ we define $v \Longrightarrow_{m_i} w$ if and only if there are $w_0, w_1, \ldots, w_{n_i} \in (N \cup T)^*$ such that $w_0 = v$, $w_{n_i} = w$, and for each $j, 1 \leq j \leq n_i$, w_j is the result of the application of $m_{i,j}$ to w_{j-1}. The language generated by G is

$$L(G) = \{w \in T^* \mid S \Longrightarrow_{m_{i_1}} w_1 \ldots \Longrightarrow_{m_{i_k}} w_k,\ w_k = w,\ w_j \in (N \cup T)^*,\ m_{i_j} \in M \text{ for } 1 \leq j \leq k, k \geq 1\}.$$

The family of languages generated by matrix grammars without appearance checking (over a one-letter alphabet) is denoted by MAT^{λ} ($MAT^{\lambda}(1)$). It is known that $CF \subset MAT^{\lambda} \subset RE$ as well as $PsCF \subset PsMAT^{\lambda} \subset PsRE$, and especially $NREG = NCF = PsMAT^{\lambda}(1) \subset NRE$. For further details about matrix grammars we refer to [8] and to [23].

2.3 Register Machines

The proofs of the results establishing computational completeness in the area of P systems often are based on the simulation of register machines; we refer to [17] for original definitions, and to [11] for definitions like those we use in this paper:

An *n-register machine* is a construct $M = (n, B, P, p_0, p_h)$, where n is the number of registers, B is a set of labels for injectively labelling the instructions in P, p_0 is the initial/start label, and p_h is the final label.

The instructions are of the following forms:

– $p : (A(r), q, s)$ (ADD instruction)
  Add 1 to the contents of register r and proceed to one of the instructions (labelled with) q and s.

- $p : (S(r), q, s)$ (SUB instruction)
  If register r is not empty, then subtract 1 from its contents and go to instruction q, otherwise proceed to instruction s.
- $p_h : halt$ (HALT instruction)
  Stop the machine. The final label l_h is only assigned to this instruction.

A (non-deterministic) register machine M is said to generate a vector $(n_1, ..., n_\beta)$ of natural numbers if, starting with the instruction with label p_0 and all registers containing the number 0, the machine stops (it reaches the instruction $p_h : halt$) with the first β registers containing the numbers $n_1, ..., n_\beta$ (and all other registers being empty).

Without loss of generality, in the succeeding proofs we will assume that for non-deterministic register machines in each ADD instruction $p : (A(r), q, s) \in P$ and in each SUB instruction $p : (S(r), q, s) \in P$ the labels p, q, s are mutually distinct (for a proof see [16]).

A register machine is called *deterministic* if and only if in every ADD instruction $p : (A(r), q, s) \in P$ we have $q = s$; in this case we also write $p : (A(r), q)$ instead. A deterministic register machine M is said to accept a vector $(n_1, ..., n_\beta)$ of natural numbers if, starting with the instruction with label p_0 and registers 1 to β containing the numbers $n_1, ..., n_\beta$, the machine stops (it reaches the instruction $p_h : halt$) with the all 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 non-negative integers which can be generated by Turing machines, i.e., the family $PsRE$.

The results proved in [10] (based on the results established in [17]) and [11], [14] immediately lead to the following results:

Proposition 1. *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 $(n_1, ..., n_\beta) \in L$. Moreover, the registers 1 to β are never decremented.*

Proposition 2. *For any recursively enumerable set $L \subseteq \mathbb{N}^\beta$ of vectors of non-negative integers there exists a deterministic $(\beta + 2)$-register machine M accepting L in such a way that, when starting with $n_1, ..., n_\beta$ in registers 1 to β and with register $\beta + 1$ and $\beta + 2$ being empty, M halts with all registers being empty if and only if $(n_1, ..., n_\beta) \in L$.*

2.4 A General Model of P Systems with Permitting Contexts

We now introduce a general model of P systems with permitting contexts covering the most important models of communication P systems as well as evolution/communication P systems. For the state of the art in the P systems area, we refer to the P systems web page [25].

A *P system (of degree* d, $d \geq 1$*) with permitting contexts* (in the following also called P system for short) is a construct

$$\Pi = (V, T, E, \mu, w_0, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o) \text{ where}$$

1. V is an alphabet; its elements are called *objects*;
2. $T \subseteq V$ is an alphabet of *terminal objects*;
3. $E \subseteq V$ is the set of objects occurring in an unbounded number in the environment;
4. μ is a *membrane structure* consisting of d membranes (usually labelled with i and represented by corresponding brackets $[_i$ and $]_i$, $1 \leq i \leq d$);
5. w_i, $1 \leq i \leq d$, are strings over V associated with the regions $1, 2, \ldots, d$ of μ; they represent multisets of objects initially present in the regions of μ; w_0 represents the multiset of objects from $V \setminus E$ initially present in the environment (in the following we usually shall assume $w_0 = \lambda$);
6. R_i, $1 \leq i \leq d$, are finite sets of *membrane rules with permitting contexts* over V associated with the membranes $1, 2, \ldots, d$ of μ; these evolution rules in R_i are of the form ${}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z}$, where $w, z \in V^*$ are the contexts in the region outside membrane i and inside membrane i, respectively, u outside membrane i is replaced by v and x inside membrane i is replaced by y;
7. i_o is a number between 1 and d and it specifies the *output* membrane of Π.

The rule ${}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z}$ from R_i is applicable if and only if the multiset uw occurs in the region outside membrane i (in the following also denoted by $\hat{\imath}$) and the multiset xz occurs in the region inside membrane i. The application of this rule results in subtracting the multiset identified by u from the multiset in $\hat{\imath}$ and adding v instead as well as subtracting x and adding y in the region inside membrane i. The permitting contexts w and z themselves or subsets of w and z can be (part of) permitting contexts in other rules and, moreover, even be modified by another rule in the same derivation step. On the other hand, any object can be modified, i.e., be part of u or x in a rule ${}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z}$, by only one application of one rule in each derivation step. The rules to be applied in parallel and the objects to be modified by these rules are chosen in a non-deterministic way.

Instead of writing ${}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z} \in R_i$ we can also write ${}^{u}_{w}[_i {}^{x}_{z} \rightarrow {}^{v}_{w}[_i {}^{y}_{z}$ and in this way collect all rules from the R_i, $1 \leq i \leq d$, in one single set of rules $R = \left\{ {}^{u}_{w}[_i {}^{x}_{z} \rightarrow {}^{v}_{w}[_i {}^{y}_{z} \mid {}^{u}_{w}[^{x}_{z} \rightarrow {}^{v}_{w}[^{y}_{z} \in R_i \right\}$.

The membrane structure and the multisets represented by w_i, $0 \leq i \leq d$, in Π constitute the *initial configuration* of the system.

In the *maximally parallel derivation mode*, a transition from one configuration to another one is obtained by the application of a maximal multiset of rules, i.e., no additional rules could be applied anymore to the objects occurring in the current configuration. The system continues maximally parallel derivation steps until there remain no applicable rules in any region of Π; then the system halts (*total halting*). We consider the number of objects from T contained in the output membrane i_o at the moment when the system halts as the *result* of the underlying

computation of Π yielding a vector of non-negative integers for the numbers of terminal symbols in the output membrane i_0; observe that here we do not count the non-terminal objects present in the output membrane. The set of results of all halting computations possible in Π is denoted by $Ps(\Pi)$, respectively. Below, we shall consider variants of P systems using only rules of very restricted types α. The family of all sets of vectors of non-negative integers computable by P systems with d membranes and using rules of type α is denoted by $Ps_gOP_d(\alpha, max, H)$.

When using the *minimally parallel derivation mode*, in each derivation step we choose a multiset of rules from the R_i in such way that to this chosen multiset no rule from a set R_j from which no rule has been taken so far, could be added anymore to be applied in parallel with the rules already chosen.

In the *asynchronous* and the *sequential derivation mode*, in each derivation step we apply an arbitrary number of rules/ exactly one rule, respectively. The corresponding families of sets of vectors of non-negative integers generated by P systems with d membranes and using rules of type α are denoted by $Ps_gOP_d(\alpha, X, H)$, $X \in \{min, asyn, sequ\}$.

If instead of the total halting we take *partial halting*, i.e., computations halting as soon as in at least from one set of rules no rule is applicable anymore, the corresponding families are denoted by $Ps_gOP_d(\alpha, X, h)$, $X \in \{max, min, asyn, sequ\}$.

All these variants of P systems can also be considered as accepting devices, the input being given as the numbers of objects in the distinguished membrane i_0. The corresponding families of sets of vectors of non-negative integers accepted by P systems with d membranes and using rules of type α are denoted by $Ps_aOP_d(\alpha, X, Y)$, $X \in \{max, min, asyn, sequ\}$, $Y \in \{H, h\}$. In this case, it also makes sense to consider deterministic P systems, i.e., systems where for each configuration obtained in this system we can derive at most one configuration. The corresponding families are denoted by $DPs_aOP_d(\alpha, X, Y)$.

If we only count the number of terminal objects and do not distinguish between different (terminal) objects, in all the definitions given above, we replace Ps by N. When the parameter d is not bounded, it is replaced by $*$.

In the following, we now consider several restricted variants of membrane rules with permitting contexts well known from the literature.

P systems with symport/antiport rules

For definitions and results concerning P systems with symport/antiport rules, we refer to the original paper [18] as well as to the overview given in [22]. An *antiport rule* is a rule of the form ${}^{u}[^{x} \rightarrow {}^{x}[^{u}$ usually written as $(x, out; u, in)$, $ux \neq \lambda$. A *symport rule* is of the form $[^{x} \rightarrow {}^{x}[$ or ${}^{u}[\rightarrow [^{u}$ usually written as (x, out), $x \neq \lambda$, or (u, in), $u \neq \lambda$, respectively.

The weight of the antiport rule $(x, out; u, in)$ is defined as $\max\{|x|, |u|\}$. Using only antiport rules with weight k induces the type α usually written as $anti_k$. The weight of a symport rule (x, out) or (u, in) is defined as $|x|$ or $|u|$, respectively. Using only symport rules with weight k induces the type α usually written as

sym_k. If only antiport rules $(x, out; u, in)$ of weight ≤ 2 and with $|x| + |u| \leq 3$ as well as symport rules of weight 1 are used, we shall write $anti_{2'}$.

P systems with conditional uniport rules

A *conditional uniport rule* is a rule of one of the forms ${}^{ab}[\to {}^{b}[^{a}$, $[^{ab} \to {}^{a}[^{b}$, ${}^{a}[^{b} \to [^{ab}$, ${}^{b}[^{a} \to {}^{ab}[$, with $a, b \in V$; in every case, the object a is moved across the membrane, whereas the object b stays where it is. Using only rules of that kind induces the type $uni_{1,1}$. Conditional uniport rules were first considered in [24] for the case of tissue P systems, showing computational completeness with maximal parallelism and total halting (using 24 cells).

P systems with boundary rules and evolution/communication P systems

In P systems with boundary rules as defined in [4], evolution rules as well as communication rules with permitting contexts are considered. Usually, we only consider evolution rules that are non-cooperative, i.e., of the form $a \to v$ with $a \in V$ and $v \in V^*$; a rule $a \to v \in R_i$ corresponds to $[^{a} \to [^{v} \in R_i$ in our general notation. The communication rules are symport or antiport rules with permitting contexts, i.e., of the form ${}^{u}_{w}[^{x}_{z} \to {}^{x}_{w}[^{u}_{z}$.

In [5], boundary rules of the form ${}^{u}[^{x} \to {}^{v}[^{y}$ are considered, i.e., rewriting on both sides of the membrane. In evolution/communication P systems as introduced in [6], we allow non-cooperative evolution rules as well as antiport (of weight k) and symport rules (of weight l), and we denote this type of rules by $(ncoo, anti_k, sym_l)$.

3 Results

After recalling some general results for the new variant of *partial halting* already established in [13], which immediately yield comparable computational completeness results in the case of antiport P systems for total and partial halting, we prove that P systems with permitting contexts working in the sequential, in the asynchronous or even in the minimally parallel derivation mode and with partial halting can only generate Parikh sets of matrix languages (regular sets of non-negative integers). On the other hand, specific variants of P systems with permitting context such as P systems with antiport rules, P systems with symport rules, P systems with conditional uniport rules, and evolution/communication P systems together with the newly introduced variant of minimal parallelism and with total halting are computationally complete.

3.1 General Observations

Looking carefully into the definitions of the derivation modes as well as the halting modes explained above, we observe the following general results already established in [13]:

Theorem 1. *Any variant of P systems yielding a family of sets of non-negative integers F when working in the derivation mode X, $X \in \{max, min, asyn, sequ\}$, with only one set of rules assigned to a single membrane and stopping with total halting yields the same family F when working in the derivation mode X with only one set of rules assigned to a single membrane when stopping with partial halting, too.*

Theorem 2. *Any variant of P systems yielding a family of sets of non-negative integers F when working in the derivation mode X, $X \in \{asyn, sequ\}$, with only one set of rules assigned to a single membrane and stopping with total or partial halting, respectively, yields the same family F when working in the minimally parallel derivation mode and stopping with the corresponding halting mode, too.*

For any P system using rules of type α, with a derivation mode X, $X \in \{min, asyn, sequ\}$, and partial halting, we only get Parikh sets of matrix languages (regular sets of non-negative integers):

Theorem 3. *For every $X \in \{min, asyn, sequ\}$,*
$Ps_gOP_*(\alpha, X, h) \subseteq PsMAT^\lambda$ and $N_gOP_*(\alpha, X, h) \subseteq NREG$.

Proof. We only prove $Ps_gOP_*(\alpha, X, h) \subseteq PsMAT^\lambda$; the second inequality $N_gOP_*(\alpha, X, h) \subseteq NREG$ is a direct consequence of the first one, having in mind that $NREG = PsMAT^\lambda(1)$. Hence, let us start with a P system $\Pi = (V, T, E, \mu, w_1, \ldots, w_d, R_1, \ldots, R_d, i_o)$ using rules of a specific type α, working with the derivation mode X. The stopping condition h – partial halting – then guarantees that in order to continue a derivation there must exist a sequence of rules $\langle r_1, \ldots, r_d \rangle$ with $r_i \in R_i$, $1 \leq i \leq d$, such that all these rules are applicable in parallel. We now consider all functions δ with $\delta(i, r) \in \{0, 1\}$ and $\delta(i, r) = 1$ if and only if the rule $r \in R_i$, $1 \leq i \leq d$, is assumed to be applicable to the current sentential form in a matrix grammar $G_M = \left(V_M, \overline{T}, S, M\right)$ generating representations of all possible configurations computable in the given P system Π with the representation of an object a in membrane i as (i, a). We start with the matrix $(S \to Kh(w))$ where $h(w)$ is a representation of the initial configuration. A derivation step in Π then is simulated in G_M as follows:

(i) We non-deterministically choose some δ as described above and use the matrix $(K \to K(\delta))$. Afterwards, we use the matrix $(K(\delta) \to K'(\delta), s_1, ..., s_m)$ where each subsequence s_j, $1 \leq j \leq m$, checks the applicability of a rule $r \in R_i$ with $\delta(i, r) = 1$. For checking the applicability of $\begin{smallmatrix} u \\ w \end{smallmatrix}\left[\begin{smallmatrix} x \\ z \end{smallmatrix} \to \begin{smallmatrix} v \\ w \end{smallmatrix}\right[\begin{smallmatrix} y \\ z \end{smallmatrix} \in R_i$, we have to check for the appearance of uw in membrane $\hat{i}$ (the outer region of membrane i) and for the appearance of xz in the (inner) region of membrane i. This can be done by the subsequence $((\hat{i}, uw) \to (\hat{i}, \overline{uw}), (\hat{i}, \overline{uw}) \to (\hat{i}, uw), (i, xz) \to (i, \overline{xz}), (i, \overline{xz}) \to (i, xz))$, where $(i, v) \to (i, \overline{v})$, for $v = v_1 ... v_h$, $v_j \in V$, $1 \leq j \leq h$, $h \geq 0$, is a shortcut for the sequence $((i, v_1) \to (i, \overline{v_1}), ..., (i, v_h) \to (i, \overline{v_h}))$ etc.

(ii) After that, we non-deterministically guess a sequence of rules $\langle r_1, \ldots, r_d \rangle$ with $r_i \in R_i$, $r_i = \begin{smallmatrix} u(i) \\ w(i) \end{smallmatrix}\left[\begin{smallmatrix} x(i) \\ z(i) \end{smallmatrix} \to \begin{smallmatrix} v(i) \\ w(i) \end{smallmatrix}\right[\begin{smallmatrix} y(i) \\ z(i) \end{smallmatrix}$, and $\delta(i, r_i) = 1$, $1 \leq$

$i \leq d$, such that all these rules are applicable in parallel. This can be checked by the corresponding matrix $(K'(\delta) \to K''(\delta), t_1, ..., t_d, t'_1, ..., t'_d)$ with the subsequences t_i, t'_i, $1 \leq i \leq d$, being defined (in the shortcut notation as above) by $t_i = \left((\hat{\imath}, u(i)) \to \left(\hat{\imath}, \overline{u(i)}\right), (i, x(i)) \to \left(i, \overline{x(i)}\right)\right)$ and $t'_1 = \left(\left(\hat{\imath}, \overline{u(i)}\right) \to (\hat{\imath}, u(i)), \left(i, \overline{x(i)}\right) \to (i, x(i))\right)$. Observe that only the objects in $u(i)$ and $x(i)$ are assigned to the rule r_i, whereas the permitting contexts $w(i)$ and $z(i)$ may be contexts for another rule or be affected themselves by another rule, and, moreover, that the applicability of the rules themselves has already been checked in (i).

(iii) Finally, we take different matrices depending on the derivation mode:

1. In the sequential derivation mode, we only have to take all possible matrices simulating the application of one rule ${}^{u}_{w}[^{x}_{z} \to {}^{v}_{w}[^{y}_{z} \in R_i$ with $\delta(i, r) = 1$: $(K''(\delta) \to Kh_{\hat{\imath}}(v) h_i(y), (\hat{\imath}, u) \to \lambda, (i, x) \to \lambda)$, where the morphisms h_j are defined by $h_j(a) = (j, a)$, $0 \leq j \leq d$, $a \in V$, except $h_0(a) = \lambda$ for $a \in E$ (these symbols, by definition, are available in an unbounded number in the environment).
2. In the asynchronous derivation mode, we have to allow an arbitrary number of rules to be applied in parallel; we simulate the application of rules sequentially, priming the results such that they cannot be used immediately. Finally, if for the current derivation step, the application of no further rule is intended, we can deprime the result symbols to be available for the simulation of the next derivation step. In sum, we use the matrices $(K''(\delta) \to K'''(\delta))$, $(K'''(\delta) \to K'''(\delta) h'_{\hat{\imath}}(v) h'_i(y), (\hat{\imath}, u) \to \lambda, (i, x) \to \lambda)$ – where the morphisms h'_j are defined by $h'_j(a) = (j, a')$, $0 \leq j \leq d$, $a \in V$, except $h'_0(a) = \lambda$ for $a \in E$ – for every rule ${}^{u}_{w}[^{x}_{z} \to {}^{v}_{w}[^{y}_{z} \in R_i$ with $\delta(i, r) = 1$, as well as $\left(K'''(\delta) \to \overline{K}(\delta)\right)$, $\left(\overline{K}(\delta) \to \overline{K}(\delta), (j, a') \to (j, a)\right)$, $0 \leq j \leq d$, $a \in V$, and finally $\left(\overline{K}(\delta) \to K\right)$.
3. For the minimally parallel mode, instead of $(K''(\delta) \to K'''(\delta))$ as in 2, we simulate the application of a sequence of rules $\langle r_1, \ldots, r_d \rangle$ with $r_i \in R_i$, $1 \leq i \leq d$, $r_i = {}^{u(i)}_{w(i)}[^{x(i)}_{z(i)} \to {}^{v(i)}_{w(i)}[^{y(i)}_{z(i)}$, and $\delta(i, r_i) = 1$ such that all these rules are applicable in parallel, which is accomplished by the matrix $\Big(K''(\delta) \to K'''(\delta) h'_{\hat{1}}(v) h'_1(y) ... h'_{\hat{d}}(v) h'_d(y), \left(\hat{1}, u(1)\right) \to \lambda, (1, x(1)) \to \lambda, ..., \left(\hat{d}, u(d)\right) \to \lambda, (d, x(d)) \to \lambda\Big)$.

As a technical detail we have to mention that it does not matter whether all the primed symbols are deprimed again, this would just make them unavailable during the next steps. Any sentential form containing primed symbols is considered to be non-terminal, hence, it cannot contribute to $L(G_M)$. Moreover, we have to point out that every symbol $e \in E$ from the environment being available there in an unbounded number neither needs to be checked for appearance in $\hat{1}$ ($= 0$) nor to be generated/eliminated or primed/deprimed, i.e., rules like $(0, e) \to \lambda$, $(0, e) \to (0, \overline{e})$, $(0, \overline{e}) \to (0, e)$ have to be omitted.

Finally, we may stop the simulation of computation steps of Π and use the matrices $(K \to F)$, $(F \to F, (i,a) \to (i,\overline{a}))$ for every object a and every membrane i, and the final matrix $(F \to \lambda)$ for generating a terminal string of G_M.

Now, we have to extract the representations of final configurations from $L(G_M)$: For every possibility of choosing a sequence of rules $\langle r_1, \ldots, r_d \rangle$ with $r_i \in R_i$, $1 \leq i \leq d$, such that all these rules are applicable in parallel, we construct a regular set checking for the applicability of this sequence in any possible representation of configurations of Π; then we take the union of all these regular sets and take its complement thus obtaining a regular set R. In $L(G_M) \cap R$ we then find at least one representation for every final configuration of computations in Π, but no representation of a non-final configuration.

Finally, let g be a projection with $g((i,\overline{a})) = \lambda$ for every $i \neq i_0$ as well as $g((i_0,\overline{a})) = \lambda$ for $a \in V \setminus T$ and $g((i_0,\overline{a})) = a$ for $a \in T$. Due to the closure properties of MAT^{λ}, we obtain $Ps(g(L(G_M) \cap R)) = Ps(\Pi) \in PsMAT^{\lambda}$.

3.2 Results for Symport/Antiport Systems

The following results are well known (e.g., see [20]; for an overview of actual results also see [22]):

Theorem 4. $Ps_gOP_1(anti_{2'}, max, H) = DPs_aOP_1(anti_{2'}, max, H) = PsRE$.

Theorem 5. *For every* $X \in \{asyn, sequ\}$,
$Ps_gOP_*(anti_*, X, H) = Ps_gOP_1(anti_{2'}, X, H) = PsMAT^{\lambda}$ *and*
$N_gOP_*(anti_*, X, H) = N_gOP_1(anti_{2'}, X, H) = NREG$.

Recently, for minimal parallelism, the following result was obtained, see [7]:

Theorem 6. $N_gOP_3(anti_2, min, H) = NRE$.

We shall improve this result by showing that only two membranes are needed:

Theorem 7. $Ps_gOP_2(anti_{2'}, min, H) = PsRE$.

Proof. We only give a sketch of the proof, because the basic ideas are the same as in the usual proofs showing computational completeness for antiport P systems. Now let $M = (n, B, P, p_0, p_h)$ be a register machine generating an output vector of dimension k ($\leq n$); then we construct the P system

$$
\begin{aligned}
\Pi &= (V, T, V, \mu, p_0, ZX, R_1, R_2, 2)\,,\\
V &= \{p, p', p'', p''', \tilde{p}, \tilde{p}', \tilde{p}'', \bar{p}, \bar{p}', \bar{p}'' \mid p \in B\}\\
&\quad \cup \{X, Y, Z, Z'\} \cup \{A_i \mid 1 \le i \le n\}\,,\\
T &= \{A_i \mid 1 \le i \le k\}\,,\\
\mu &= [_1\, [_2\ \ ]_2\,]_1,\\
R_1 &= R_{1,A} \cup R_{1,S} \cup R_{1,F},\\
R_{1,A} &= \{(p, out; A_r q, in)\,, (p, out; A_r s, in) \mid p : (A\,(r)\,, q, s) \in P\}\,,\\
R_{1,S} &= \{(p, out; p'p'', in), (p''A_r, out; p''', in), (p''X, out; \bar{p}, in),\\
&\quad (p'''X, out; \tilde{p}, in), (\bar{p}, out; \bar{p}'X, in), (\bar{p}', out; \bar{p}''Y, in),\\
&\quad (\bar{p}'', out; s, in), (\tilde{p}, out; \tilde{p}'X, in), (\tilde{p}', out; \tilde{p}''Y, in),\\
&\quad (\tilde{p}'', out; q, in) \mid p : (S\,(r)\,, q, s) \in P\}\,,\\
R_{1,F} &= \{(p'Y, out; Z', in) \mid p \in B \setminus \{p_h\}\} \cup \{(Z', out)\,, (ZX, out; Z', in)\}\,,\\
R_2 &= R_{2,A} \cup R_{2,S} \cup R_{2,F},\\
R_{2,A} &= \{(A_i, in) \mid 1 \le i \le k\}\,,\\
R_{2,S} &= \{(X, out; p', in) \mid p \in B \setminus \{p_h\}\} \cup \{(Z, out; XY, in)\,, (Z, in)\}\,,\\
R_{2,F} &= \{(p'Y, out; p_h, in) \mid p \in B \setminus \{p_h\}\} \cup \{(ZX, out; p_h, in)\,, (p_h, out)\}\,.
\end{aligned}
$$

An ADD instruction $p : (A\,(r)\,, q, s) \in P$ is simulated by using one of the rules $(p, out; A_r q, in)$, $(p, out; A_r s, in)$ assigned to membrane 1; in case r is an output register, the terminal symbol A_r is moved into the output region 2 by using (A_r, in) from R_2. A SUB instruction $p : (S\,(r)\,, q, s) \in P$ is simulated by using the rules from $R_{1,S}$ and $R_{2,S}$ in parallel. The final procedure in Π starts when the final label p_h appears; as the number of symbols p' equals the number of symbols Y as they have been introduced when simulating a SUB instruction, we finally eliminate pairs $p'Y$ from the system using the rules from $R_{1,F}$ and $R_{2,F}$ until finally only p_h remains in the skin membrane and the desired output is found in the second membrane region, without any additional symbols remaining there anymore.

The general result in Theorem 1 and the special result in Theorem 4 immediately yield the following one:

Corollary 1. $Ps_gOP_1\,(anti_{2'}, max, h) = DPs_aOP_1\,(anti_{2'}, max, h) = PsRE.$

With the other derivation modes and partial halting, we only get Parikh sets of matrix languages (regular sets of non-negative integers), which is an immediate consequence of Theorem 3:

Corollary 2. *For every* $X \in \{min, asyn, sequ\}$,
$Ps_gOP_*\,(anti_*, X, h) = Ps_gOP_1\,(anti_{2'}, X, h) = PsMAT^{\lambda}$ *and*
$N_gOP_*\,(anti_*, X, h) = N_gOP_1\,(anti_{2'}, X, h) = NREG.$

For symport rules, the following result is known (e.g., see [22]):

Theorem 8. $Ps_gOP_2\,(sym_2, max, H) = Ps_aOP_2\,(sym_2, max, H) = PsRE.$

Computational completeness can also be obtained with minimal parallelism and total halting, whereas as a direct consequence of Theorem 3, we only get Parikh sets of matrix languages (regular sets of non-negative integers) with partial halting:

Theorem 9. $Ps_gOP_2\left(sym_3, min, H\right) = PsRE.$

Proof. Let $M = (n, B, P, p_0, p_h)$ be a register machine generating an output vector of dimension k ($\leq n$); then we construct the P system

$$
\begin{aligned}
\Pi &= (V, T, E, \mu, p_0, w_1, w_2, R_1, R_2, 2)\,,\\
V &= \{p, p', p'', p''', \tilde{p}, \tilde{p}', \tilde{p}'', \tilde{p}''', \bar{p}, \bar{p}', \bar{p}'', \bar{p}''', \hat{p}, \hat{p}' \mid p \in B\}\\
&\quad \cup \{A_i \mid 1 \leq i \leq n\}\,,\\
T &= \{A_i \mid 1 \leq i \leq k\}\,,\\
E &= V \setminus \{p'_h\} \cup \{p', p''', \tilde{p}'', \bar{p}, \bar{p}', \bar{p}''', \hat{p}' \mid p \in B\}\\
\mu &= [_1\, [_2\,]_2\,]_1,\\
w_1 &= \{p_0, p'_h\} \cup \{p', p''', \tilde{p}'', \bar{p}, \bar{p}''' \mid p \in B\}\,,\\
w_2 &= \{\hat{p}', \bar{p}' \mid p \in B\}\,,\\
R_1 &= R_{1,A} \cup R_{1,S} \cup R_{1,F},\\
R_{1,A} &= \{(pp', out)\,, (p' A_r p'', in)\,, (p'' p''', out)\,, (p''' q, in)\,, (p''' s, in)\\
&\qquad \mid p : (A\,(r)\,, q, s) \in P, 1 \leq r \leq k\}\\
&\quad \cup \{(pp', out)\,, (p' A_r q, in)\,, (p' A_r s, in)\\
&\qquad \mid p : (A\,(r)\,, q, s) \in P, k < r \leq n\}\,,\\
R_{1,S} &= \{(pp', out)\,, (\tilde{p}\bar{p}' p', in)\,, (\tilde{p}' \tilde{p}'' A_r, out)\,,\\
&\qquad (\tilde{p}'' \tilde{p}''', in)\,, (\bar{p} p''' \tilde{p}''', out)\,, (\bar{p} p''' q, in)\,,\\
&\qquad (\tilde{p}\bar{p}' \bar{p}', out)\,, (\bar{p}' \bar{p}'', in)\,, (\bar{p}\bar{p}'' \bar{p}''', out)\,, (\bar{p}\bar{p}''' s, in)\\
&\qquad \mid p : (S\,(r)\,, q, s) \in P\}\,,\\
R_{1,F} &= \{(\hat{p}_i \hat{p}'_i \bar{p}'_i, out) \mid 1 \leq i \leq l\} \cup \left\{\left(\hat{p}'_i \hat{p}'_{i+1}, in\right) \mid 1 \leq i < l\right\}\\
&\quad \cup \{(p_h p'_h, out)\,, (p'_h \hat{p}_1, in)\}\,,\\
R_2 &= R_{1,A} \cup R_{1,S} \cup R_{2,F},\\
R_{2,A} &= \{(A_i, in) \mid 1 \leq i \leq k\}\,,\\
R_{2,S} &= \{(\tilde{p}\bar{p}, in)\,, (\tilde{p}\bar{p}', out)\,, (\bar{p}' \tilde{p}''', in)\,, (\bar{p}' \bar{p}'', in)\,,\\
&\qquad (\bar{p}\tilde{p}''', out)\,, (\bar{p}\bar{p}'', out) \mid p : (S\,(r)\,, q, s) \in P\}\,,\\
R_{2,F} &= \{(\hat{p}_i, in)\,, (\hat{p}_i, out)\,, (\hat{p}_i \hat{p}'_i \bar{p}'_i, out) \mid 1 \leq i \leq l\}\,.
\end{aligned}
$$

An ADD instruction $p : (A\,(r)\,, q, s) \in P$, $k < r \leq n$, is simulated by sending out the label of the instruction p together with p' which returns with A_r as well as the label of the next instruction q or s to be simulated. The simulation of an ADD instruction $p : (A\,(r)\,, q, s) \in P$, $1 \leq r \leq k$, takes two steps more – after sending out p, p' we return with p' and A_r as well as with p'' which is sent out together with p'''; p''' then returns with the label of the next instruction q or s to be simulated, whereas in the meantime A_r has got the chance to enter membrane 2.

In the case of a SUB instruction $p : (S\,(r)\,, q, s) \in P$, p' returns with $\tilde{p}$ and $\tilde{p}'$. Whereas $\tilde{p}$ enters membrane 2 together with $\bar{p}$, $\tilde{p}'$ gets the chance to take one copy of A_r out of region 1 using the rule $(\tilde{p}' \tilde{p}'' A_r, out)$. Depending on whether this rule had to be applied or not, the simulation proceeds until finally the label of the corresponding instruction to be simulated next is brought in together with p''' or $\bar{p}'''$, respectively. We should like to mention that all the symbols from $V \setminus E$ used during the simulation finally have returned to their original locations.

When the computation of the register machine stops, the label p_h appears; in order to clean the elementary membrane region 1 from non-terminal symbols we

have to use the rules from $R_{1,F}$ and $R_{2,F}$; the labels from B are assumed to be ordered in a sequence p_1 to p_l. The symbols $\bar{p}'_i$ are taken out one after the other using the rules $(\hat{p}_i\hat{p}'_i\bar{p}'_i, out)$; the rules $(\hat{p}_i, in)$ and $(\hat{p}_i, out)$ could be used for an unbounded number of steps, but for obtaining a terminating computation at some moment the rule $(\hat{p}_i\hat{p}'_i\bar{p}'_i, out)$ has to be used twice. The computation halts when all $\bar{p}'_i$ have been taken out.

Corollary 3. *For every* $X \in \{min, asyn, sequ\}$,
$Ps_gOP_*\,(sym_*, X, h) = PsMAT^{\lambda}$ *and* $N_gOP_*\,(sym_*, X, h) = NREG$.

3.3 Results for P Systems with Conditional Uniport Rules

Using only conditional uniport rules of type $uni_{1,1}$, we again obtain computational completeness, even with the minimally parallel derivation mode, together with total halting, whereas, as a direct consequence of Theorem 3, with partial halting we only get Parikh sets of matrix languages (regular sets of non-negative integers) with the minimally parallel derivation mode.

Theorem 10. $Ps_gOP_{13}\,(uni_{1,1}, X, H) = PsRE$, *for every* $X \in \{min, max\}$.

Proof. We only give a sketch of the proof showing how register machine instructions can be simulated, using the following membrane structure:

$$\left[_1\left[_2\left[_{2'}\right]_{2'}\right]_2\left[_3\left[_6\right]_6\left[_7\left[_{7'}\right]_{7'}\right]_7\left[_8\left[_{8'}\right]_{8'}\right]_8\right]_3\left[_4\right]_4\left[_5\left[_{5'}\left[_{5''}\right]_{5''}\right]_{5'}\right]_5\right]_1$$

Throughout the proof we will use a specific notation for the conditional uniport rules to illustrate the trajectories of the objects, in a more relaxed way also allowing symport rules of weight one (observe that we could use dummy objects as conditions). Now let i be a membrane and $\hat{\imath}$ be the surrounding membrane (for the skin membrane 1 this is the environment 0) ; then we use the following notations (a, b are arbitrary objects):

$(i, a, \hat{\imath})$	represents	$\left[_i{}^{a} \rightarrow {}^{a}\left[_i\right.\right.$,
$(\hat{\imath}, a, i)$	represents	${}^{a}\left[_i\right. \rightarrow \left[_i{}^{a}\right.\right.$,
$(i, a \rightarrow \hat{\imath}, b)$	represents	${}^{b}\left[_i{}^{a} \rightarrow {}^{ab}\left[_i\right.\right.$,
$(\hat{\imath}, a \rightarrow i, b)$	represents	${}^{a}\left[_i{}^{b} \rightarrow \left[_i{}^{ab}\right.\right.$,
$(i, a\,[b] \rightarrow \hat{\imath})$	represents	$\left[_i{}^{ab} \rightarrow {}^{a}\left[_i{}^{b}\right.\right.$,
$(\hat{\imath}, a\,[b] \rightarrow i)$	represents	${}^{ab}\left[_i\right. \rightarrow {}^{b}\left[_i{}^{a}\right.\right.$.

Moreover, we will use the shortcut notation

$$(A, B; i_1, \ldots, i_k; j_1, \ldots, j_{k-1})$$

which corresponds to the following group of rules:

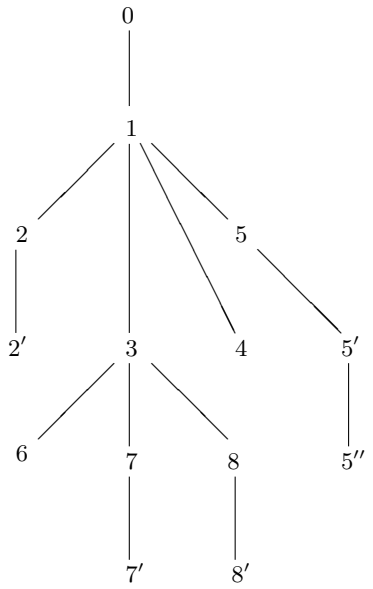


Fig. 1. Membrane structure

$$
\begin{array}{l}
(i_t, B, j_t) \\
(j_t, B, j'_t) \\
(j'_t, B, j_t) \\
(i_t, A[B], i_{t+1}) \\
(i_t, B \to i_{t+1}, A) \\
\text{for all } 1 \leq t \leq k-1
\end{array}
$$

Informally, this means that the symbols A and B travel together from membrane i_1 to membrane i_k following the path $i_1, \ldots, i_k$. The rules above permit to implement this behavior in $2k$ steps, because if something else happens, then symbol B will be trapped between membranes j_t and j'_t. Of course, we assume that B can never go out from membrane j_t and that B cannot interact with other symbols during its move along the path.

The system is constructed as follows: membrane 8 holds the current state, membrane 7 holds values of registers, membranes 4, 5 and 6 hold additional symbols. Membrane 2 and all primed membranes are used to trap symbols.

For simulating an ADD instruction, the state symbol from membrane 8 travels with an accompanying symbol (from membrane 6) to membrane 5. The accompanying symbol then brings a second accompanying symbol from the environment and both of them move the new state and a copy of the register object A_r to the corresponding membranes. After that, the first accompanying symbol returns to its original location.

The SUB instruction is simulated in an even easier way. The state symbol brings two symbols from membrane 6 to membrane 3. While it travels with one of these symbols to membrane 5, the second one tries to decrement the register. Now depending on the position of this second symbol (membrane 3 if the register is zero, or membrane 1 if the register is not zero) the corresponding new state is chosen.

We remark, that finally all involved symbols return to their original locations. Moreover, we always move groups of two symbols and if a rule not conforming to the scenario above is applied, then one of these symbols is trapped.

In more detail, for the simulation of an ADD instruction $p : (A(r), q, s)$ the following rules are used:

$I.$ $(I_p, p; 3, 1, 5; 7, 2)$
$II.$ $(T_p, I_p; 5, 1; 5')$
$III.$ $(I_p, A_r; 1, 3, 7; 2, 8)$
$IV.$ $(X_p, q; 1, 3, 8; 2, 7)$
$V.$ $(X_p, s; 1, 3, 8; 2, 7)$

1. $(8, p, 3)$
2. $(6, I_p \to 3, p)$
3. $(3, I_p, 6)$
4. $(7, I_p, 3)$
5. $(0, A_r \to 1, I_p)$
6. $(0, X_p \to 1, T_p'')$
7. $(5, T_p' \to 1, T_p)$
8. $(5, q \to 1, T_p)$
9. $(5, s \to 1, T_p)$
10. $(1, T_p', 5)$
11. $(1, T_p[q], 5)$
12. $(1, T_p[s], 5)$
13. $(4, T_p'' \to 1, T_p')$
14. $(1, T_p'', 4)$
15. $(1, T_p, 2)$
16. $(1, X_p, 2)$
17. $(2, T_p, 2')$
18. $(2, X_p, 2')$
19. $(2', T_p, 2)$
20. $(2', X_p, 2)$
21. $(8, X_p, 8')$

The simulation usually starts with p in membrane 8, with I_p being in membrane 6, and q, s in membrane 5, respectively.

Symbol p goes to membrane 3 and brings there symbol I_p from membrane 6. After that they both travel to membrane 5. There symbol I_p is moved together with symbol T_p to membrane 1. After that I_p brings one A_r from the environment and they travel together to membrane 7 and continue afterwards to membrane 3 and 6. In the meanwhile, symbol T_p in membrane 1 brings T_p' and q or s (in this order, otherwise the state symbol will be trapped in membrane 2). At the same time when q (or s) is brought in membrane 1, symbol T_p'' is brought into membrane 1 by T_p'. After that symbol T_p is sent to membrane 5 and at the same time symbol X_p is brought into membrane 1 by the symbol T_p''. Finally, symbols X_p and q (or s) go together to membrane 8. We remark that the rules 3, 10 or 14 may be used instead of some rules of the chain presented above. But in this case symbol p or symbol T_p will be trapped. We also remark that X_p and q will be ready to move to membrane 3 after I_p and A_r have arrived there.

The simulation of a SUB instruction $p : (A(r), q, s)$ is performed by the following rules:

$I.\ (M'_p, p; 1, 5; 2)$
$II.\ (M_p, A; 3, 1; 8)$
$III.\ (q, M'_p; 5, 1; 5')$
$IV.\ (s, M'_p; 5, 1; 5')$
$V.\ (M_p, q; 1, 3; 2)$

1. $(8, p, 3)$
2. $(6, M_p \to 3, p)$
3. $(6, M'_p \to 3, p)$
4. $(3, M'_p[M_p], 6)$
5. $(3, M_p \to 6, M'_p)$
6. $(7, A_r \to 3, M_p)$
7. $(3, M_p[p], 1)$
8. $(3, p \to 1, M'_p)$
9. $(1, M'_p, 3)$
10. $(1, s \to 3, M_p)$
11. $(1, A_r[M_p], 0)$
12. $(3, q[M_p], 8)$
13. $(3, s[M_p], 8)$
14. $(3, p, 7)$
15. $(7, p, 7')$
16. $(7', p, 7)$

The simulation usually starts with object p in membrane 8, whereas M_p, M'_p are found in membrane 6, q, x in membrane 5, and A_r possibly in membrane 7.

Symbol p first goes from membrane 8 to membrane 3 and after that brings there symbol M'_p. After that it moves symbol M'_p to membrane 1 and brings symbol M_p to membrane 3. Further, p moves to membrane 1 and M_p may bring a symbol A_r to membrane 3. If it succeeds, then both symbols M_p and A_r move to membrane 1 and after that symbol A_r is sent out. In the meanwhile p and M'_p move to membrane 5. From there, M'_p brings either q or s to membrane 1. Now if it brought q and the register was not zero (M_p is in membrane 1) then M_p will bring q to membrane 8. Otherwise, q will be trapped in membrane 2. If s was brought into membrane 1 by M'_p, and the value of register was zero, then this s will move to membrane 3 and further to membrane 8. It is easy to observe that the symbols M_p and M'_p return to membrane 6 by themselves. They can do this at any moment of the computation, but only after the state symbol q or s has returned to membrane 8 they can do this without provoking an infinite computation.

3.4 Results for Evolution/Communication P Systems

For evolution/communication P systems, the constructions from [2], Theorems 1 and 2, and from [3], Theorems 4.3.1 and 4.3.2, already show the computational completeness, using two membranes, for the minimally parallel setup (when working in the maximally parallel way, the system never applies simultaneously more than one rule from the same set of rules assigned to a membrane):

Corollary 4. *For* $X \in \{min, max\}$,
$Ps_gOP_2((ncoo, anti_1, sym_1), X, H) = PsRE,\ X \in \{min, max\}$.

We can extend these results by showing that *deterministic* evolution-communication P systems with non-cooperative evolution rules and communication rules of weight one (also see [1]) are computationally complete, using three membranes.

Theorem 11. *For* $X \in \{min, max\}$,
$DPs_aOP_3((ncoo, sym_1, anti_1), X, H) = PsRE$.

Proof. Consider a deterministic register machine $M = (n, B, P, p_0, p_h)$. Let us denote the set of labels of SUB instructions by B_-.

$$
\begin{aligned}
\Pi &= \left(V, T, V, [_1\, [_2\,]_2 [_3\,]_3\,]_1, p_0, \lambda, \lambda, R_1, R_2, R_3, 1\right), \\
V &= B \cup \{l_j \mid l \in B_-, 0 \le j \le 7\} \cup \{q\} \\
&\quad \cup \{A_i \mid 1 \le i \le n\} \cup \{i_j \mid 1 \le i \le n,\ 1 \le j \le 3\}, \\
T &= \{A_i \mid 1 \le i \le k\}, \\
R_1 &= \{l \to A_i l' \mid (l : A(i), l') \in P\} \cup \{l_j \to l_{j+1} \mid l \in B_-,\ j \in \{0,1,2,5,6\}\} \\
&\quad \cup \{l \to l_0 i_1,\ l_4 \to l_5 q,\ l_7 \to l'' \mid l : (S(i), l', l'') \in P\}, \\
R_2 &= \{i_1 \to i_2,\ i_3 \to \lambda \mid 1 \le i \le n\} \cup \{l_3 \to l_4 \mid l \in B_-\}, \\
&\quad \cup \{(i_1, in),\ (i_2, out; A_i, in),\ (i_3, in) \mid 1 \le i \le n\} \\
&\quad \cup \{(i_2, out; l_3, in),\ (l_4, out) \mid l : (S(i), l', l'') \in P\}, \\
R_3 &= \{i_2 \to i_3 \mid 1 \le i \le n\} \cup \{q \to \lambda\} \cup \{l_3 \to l' \mid l \in B_-\} \\
&\quad \cup \{(i_2, in),\ (i_3, out; q, in) \mid 1 \le i \le n\} \\
&\quad \cup \{(i_3, out; l_3, in),\ (l', out) \mid l : (S(i), l', l'') \in P\}.
\end{aligned}
$$

An ADD instruction $l : (A(i), l') \in P$ is implemented by the single non-cooperative evolution rule $l \to A_i l'$.

The simulation of a SUB instruction $l : (S(i), l', l'') \in P$ is described in a depictive way in the following tables. Decrementing register i works as follows:

step	0	1	2	3	4
region 2			i_1	i_2	a_i
rules 2		(i_1, in)	$i_1 \to i_2$	$(i_2, out; A_i, in)$	
region 1	lA_i	$l_0 i_1 A_i$	$l_1 A_i$	$l_2 A_i$	$l_3 i_2$
rules 1	$l \to l_0 i_1$	$l_0 \to l_1$	$l_1 \to l_2$	$l_2 \to l_3$	
region 3					
rules 3					(i_2, in)

step	5	6	7	8	
region 2				i_3	
rules 2			(i_3, in)	$i_3 \to \lambda$	
region 1	l_3	l_3	i_3		l'
rules 1					
region 3	i_2	i_3	l_3	l'	
rules 3	$i_2 \to i_3$	$(i_3, out; l_3, in)$	$l_3 \to l'$	(l', out)	

If register i is empty, i.e., if there is no object A_i, the simulation works as follows:

step	0	1	2	3	4	
region 2			i_1	i_2	i_2	
rules 2		(i_1, in)	$i_1 \to i_2$		$(i_2, out; l_3, in)$	
region 1	l	$l_0 i_1$	l_1	l_2	l_3	
rules 1	$l \to l_0 i_1$	$l_0 \to l_1$	$l_1 \to l_2$	$l_2 \to l_3$		
region 3						
rules 3						

step	5	6	7	8	9	10
region 2	l_3	l_4				i_3
rules 2	$l_3 \to l_4$	(l_4, out)			(i_3, in)	$i_3 \to \lambda$
region 1	i_2		l_4	$l_5 q$	$l_6 i_3$	l_7
rules 1			$l_4 \to l_5 q$	$l_5 \to l_6$	$l_6 \to l_7$	$l_7 \to l''$
region 3		i_2	i_3	i_3	q	
rules 3	(i_2, in)	$i_2 \to i_3$		$(i_3, out; q, in)$	$q \to \lambda$	

The main idea of the construction is similar to that in [1]: if the object A_i is present in region 2, it is exchanged with object i_2 while object l_2 changes to l_3; otherwise object l_3 is exchanged with i_2 in the next step. The trajectory of object i_2 is the same in both cases: it enters membrane 3, is renamed, exits and enters membrane 2, where it is erased. The behavior of object l_3 indirectly depends on the presence of A_i via i_2 or i_3: in the decrement case it enters membrane 3, is renamed to l' and returns to region 1, while in the zero-case it enters membrane 2, is renamed to l_4, exits, and produces two objects; one of them helps i_3, while the other one produces l'', thus finishing the simulation.

4 Conclusion

In this paper, we have investigated a new variant of halting – we call it *partial halting* – in membrane systems where all membranes are required to allow for the application of a rule at the same time in order to keep a computation alive. Obviously, for systems with only one membrane this way of halting is equivalent with the original one where a system halts if and only if no rule is applicable anymore in the whole system – we also call this *total halting*. Besides this general result, we also have shown that P systems working in the minimally parallel mode, the asynchronous or the sequential derivation mode and with partial halting can only generate Parikh sets of matrix languages/regular sets, the same what we obtain with the sequential and the asynchronous derivation mode and total halting.

Comparing the results for total and partial halting for the minimally parallel derivation mode elaborated above, we realize that for any of the specific restricted variants α of P systems with permitting contexts we have

$$Ps_gOP_*(\alpha, min, h) \subseteq PsMAT^{\lambda} \subsetneqq PsRE = Ps_gOP_*(\alpha, min, H) \text{ and}$$
$$N_gOP_*(\alpha, min, h) = NREG \subsetneqq NRE = N_gOP_*(\alpha, min, H),$$

i.e., in the case of the minimally parallel derivation mode the halting condition – *total* in contrast to *partial* halting – makes the difference. Intuitively speaking, the requirement for a computation to continue only if for every membrane a rule is applicable, together with the minimally parallel derivation mode means that we do not have the possibility of appearance checking and therefore cannot simulate the zero test for register machines, hence, we cannot obtain computational completeness.

In the future, the new variant of partial halting should also be investigated for other variants of P systems working in the different derivation modes, with multisets of objects, but also with strings, arrays, etc.

Acknowledgements.

Artiom Alhazov gratefully acknowledges support by the Academy of Finland, project 203667; he also acknowledges the project 06.411.03.04P from the Supreme Council for Science and Technological Development of the Academy of Sciences of Moldova. The work of Marion Oswald was supported by FWF-project T225-N04. 2006.

References

1. A. Alhazov: On determinism of evolution-communication P systems, *Journal of Universal Computer Science* **10**, 5, 2004, 502–508.
2. A. Alhazov, Number of protons/bi-stable catalysts and membranes in P systems. Time-freeness. In: [15], 79–95.
3. A. Alhazov: Communication in Membrane Systems with Symbol Objects, Ph.D. Thesis, Tarragona, Spain, 2006.
4. F. Bernardini, V. Manca: P systems with boundary rules. In: [21], 107–118.
5. F. Bernardini, F. J. Romero-Campero, M. Gheorghe, M.J. Pérez-Jiménez, M. Margenstern, S. Verlan, N. Krasnogor: On P systems with bounded parallelism. In: G. Ciobanu, Gh. Păun (Eds.): Pre-Proc. of First International Workshop on Theory and Application of P Systems, Timisoara, Romania, September 26–27, 2005, 31–36.
6. M. Cavaliere: Evolution-communication P systems. In: [21], 134–145.
7. G. Ciobanu, Linqiang Pan, Gh. Păun, M.J. Pérez-Jiménez: P systems with minimal parallelism, *accepted for TCS*.
8. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*, Springer-Verlag, Berlin, 1989.
9. J. Dassow, Gh. Păun: On the power of membrane computing, *Journal of Universal Computer Science* **5** (2) (1999), 33–49.
10. R. Freund, M. Oswald: GP Systems with Forbidding Context. *Fundamenta Informaticae* **49**, 1–3 (2002), 81–102.
11. R. Freund, M. Oswald: P Systems with activated/prohibited membrane channels. In: [21], 261–268.
12. R. Freund, M. Oswald: P systems with conditional communication rules assigned to membranes, *Journal of Automata, Languages and Combinatorics* **9**, 4 (2004), 387–397.

13. R. Freund, M. Oswald: P systems with partial halting, submitted, 2007.
14. R. Freund, Gh. Păun: From Regulated Rewriting to Computing with Membranes: Collapsing Hierarchies. *Theoretical Computer Science* **312** (2004), 143–188.
15. R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): Membrane Computing. 6th International Workshop WMC 2005, Vienna, Austria, Lecture Notes in Computer Science **3850**, Springer-Verlag, 2006.
16. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel states. *Theoretical Computer Science* **330** (2005), 101–116.
17. M.L. Minsky: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1967.
18. A. Păun, Gh. Păun: The power of communication: P systems with symport/ antiport, *New Generation Computing* **20**, 3 (2002), 295–306.
19. Gh. Păun: Computing with membranes, *J. of Computer and System Sciences* **61**, 1 (2000), 108–143, and TUCS Research Report 208 (1998) (`http://www.tucs.fi`).
20. Gh. Păun: *Computing with Membranes: An Introduction,* Springer-Verlag, Berlin, 2002.
21. Gh. Păun, G. Rozenberg, A. Salomaa, C. Zandron (Eds.): Membrane Computing. International Workshop WMC 2002, Curteă de Argeş, Romania, Revised Papers. Lecture Notes in Computer Science **2597**, Springer-Verlag, Berlin (2003).
22. Y. Rogozhin, A. Alhazov, R. Freund: Computational power of symport/antiport: history, advances, and open problems. In: [15], 1–30.
23. G. Rozenberg, A. Salomaa (Eds.): *Handbook of Formal Languages* (3 volumes), Springer-Verlag, Berlin, 1997.
24. S. Verlan, F. Bernardini, M. Gheorghe, M. Margenstern: On communication in tissue P systems: conditional uniport. Pre-proceedings of Membrane Computing. International Workshop, WMC7, Leiden, The Netherlands, 2006, 507–521
25. The P Systems Web Page: `http://psystems.disco.unimib.it.`

Magnetotactic Bacteria and Their Significance for P Systems and Nanoactuators

Ioan I. Ardelean[1], Mircea Ignat[2], Cristina Moisescu[1]

[1] Institute of Biology of the Romanian Academy
Centre of Microbiology
Splaiul Independentei 296, Bucharest 060031, Romania
ioan.ardelean@ibiol.ro
[2] INCDIE ICPE-CA – Dep.1/10
Splaiul Unirii Nr. 313, Bucharest 030139, Romania

Summary. In the framework of the dialog between P systems and Microbiology, in this paper we focus on the magnetotactic behavior of magnetotactic bacteria, namely the orientation along the Earth's geomagnetic field lines. Magnetic properties and magnetotactic behavior could be used to obtained micro- and nanoactuators for the desired distribution at nanometer level of either intact magnetotactic bacteria or isolated intact magnetosomes with significant potential application in the construction of magnetic logic gates. Furthermore, (precise) distribution of intact magnetotactic bacteria or isolated intact magnetosomes by carefully using rather strong external magnetic fields could be described by P systems as a discontinuous process, whose potential for nanoactuators and magnetic microchips is increasing in evidence.

1 Introduction

The cell structure and function is one root of the emergence of P systems (Păun, 2000, 2001, 2002); moreover since the early days of P systems it become evident that its formalism has the potential to describe different biological processes occurring within the cell (Păun, 2002) and this trend is under increase with significant results (see Ciobanu et al., 2006; Hoogeboom et al., 2006, and references herein).

Here we focus on our proposal that magnetotactic bacteria could add some new insights into P systems, mainly with respect to the potential to use cell components for the construction of a P system-based computer.

In 1975 Robert Blakemore published his paper on magnetotactic bacteria (MTB). He stated that MTB's main functional characteristic is magnetotaxis, the orientation along the Earth's geomagnetic field lines (Blakemore, 1975). Magnetotaxis is determined by the presence inside the cell of particles named magnetosomes. These were originally defined as intracellular, magnetic single-domain (SD) crystals of a magnetic iron mineral (magnetite or greigite) that are enveloped by a

trilaminate structure, the magnetosome membrane. The discovery of MTB stimulated interest among microbiologists, physicists, engineers, geologists, chemists (Mann et al., 1990; Schüler and Frankel, 1999) and today the subject has become a *bona fide* field of research in microbiology (Bazylinski and Frankel, 2004). Probably it is the time for MTB to receive more attention from P systems, too.

Here we put forward that MTB or isolated magnetosomes are significant for membrane computing for the following reasons:

1. The synthesis of magnetite from iron salts is a complex process involving both plasma and magnetosome membranes (Schüler, 2002, 2004). Furthermore, isolated magnetosomes devoid of their membranes are no longer organized in an oriented chain; thus, the magnetosome membrane seems to be essential for the chain arrangement of magnetosomes both *in vivo* and *in vitro*, and this membrane has not been yet the subject of any P systems approach.
2. This process – not yet known in detail – could offer the possibility for P systems to develop quantitative discrete models for synergic biochemical and biophysical processes occurring at those biological membranes. This mathematical description could be further used for:
   a) to better understand the overall process of magnetosome formation and its regulation, by identification of still unknown functional proteins and/or regulatory components, by the use of different strategies developed within P systems; (i) metabolic P graphs (MPG) seems appropriate to identify new functional and regulatory components in different biological processes (Manca, 2006) not yet exploited with respect to MTB; (ii) as well as the proposed mesoscopic approach which is more tractable than the microscopic chemistry, but it provides a finer and better understanding than the macroscopic chemistry modeled by ordinary differential equations (Pérez-Jiménez et al., 2006);
   b) on line control of an *in vitro* reactor able to mimic the function of both plasma and magnetosome membranes in order to convert soluble nonmagnetic iron salts to magnetic nanocrystals.
3. MTB or isolated magnetosomes can contribute to the connection between P systems and the emerging domain of nanobiotechnology with special emphasis on the following:
   i) the construction of nanoobjects and their precise deposition/localization: MTB synthesize magnetic nanocrystals which can follow precise movements, orientations, and depositions by the use of either magnetotactic behavior of MTB or by (rather strong) external magnetic fields (see bellow);
   ii) to offer the opportunity to P systems to develop a software for discontinuous controlling and modelling of microdevices to be constructed (see below) for precise and oriented movement and deposition of MTB and their magnetosomes;

iii) to monitor the precise and oriented movement and deposition of MTB which transport at their cell surfaces different chemically linked molecules of functional significance, as for example immunoglobulins (see below).

4. Oriented MTB/magnetosomes chains could be used for the construction of magnetic logic gates. There are already experimental reports showing the increasing scientific interest in precise deposition of biotic magnetic crystals for the construction of magnetic logic gates able to execute logical NAND and NOR operations (Haque et al., 2004). These pioneering experimental results are directly related to the emerging filed of quantum dot magnetic computing from which there are expected huge effects on the speed of computation and on a new proposed generation of computers based on magnetism rather than electricity. We put forward that magnetic properties of single domain magnetic nanocrystals produced by MTB could be more appropriate than abiotic magnetic nanocrystals for the construction of magnetic logic gates and for the already proposed generation of computers based on magnetism rather than electricity. The properties of these magnetic computers could be further improved by the use of P system based software; this proposal is sustained by the already contribution from P systems to the field of quantum dot magnetic computing (Leporati et al., 2006; Leporati and Felloni, 2007).

2 Magnetotactic Bacteria

The morphology of single celled MTB is diverse (spirilla, vibrioids, cocci, rods to ovoid) and there are reports on multi- celled magnetotactic prokaryotes. MTB's main functional characteristic is magnetotaxis, the orientation along the Earth's geomagnetic field lines (Blakemore, 1975). Magnetotaxis is determined by the presence inside the cell of particles named magnetosomes (Frenkel and Blackmore, 1980; Frenkel et al., 1997)

Magnetosomes were originally defined as intracellular, magnetic single-domain (SD) crystals of a magnetic iron mineral that are enveloped by a trilaminate structure, the magnetosome membrane (MM). In other words, a magnetosome consists of magnetic iron mineral particles (the inorganic phase) enclosed within a membrane (the organic phase). In Figure 1 there is presented an original image of *M. gryphiswaldense* cell.

The organic phase (the magnetosome membrane or the magnetosome vesicle), consists in *Magentospirillum* strains (*M. magnetotacticum* or *M. gryphiswaldense*) of a bilayer of about 3-4 nm containing phospholipids and proteins. In the last few years the study of the proteins found in magnetosome membranes has raised a special interest because it was expected that these proteins would enable the processes of mineral formation of nanocrystals to be regulated by biochemical pathways (Schüler and Frankel, 1999; Bazylinski and Frankel, 2004).

The magnetosome particle is characterized by a nearly perfect crystallinity and the size and morphology of magnetic crystals are species specific and uniform

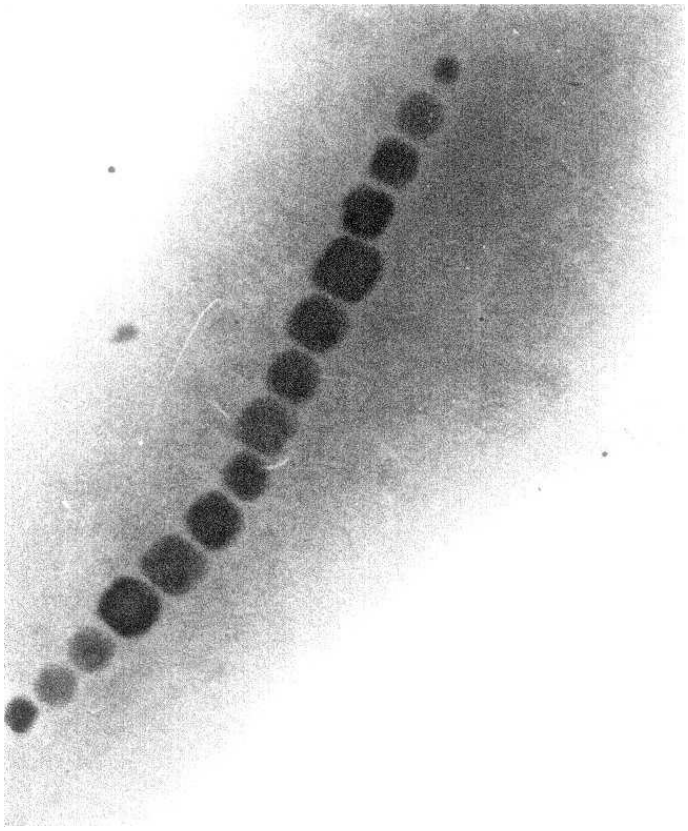


Fig. 1. Transmission electron microscope image of an intact cell of *Magnetospirilum gryphyswaldense*; one can see the chain of magnetosomes as dark bodies inside the bacterial cell (light grey).

within a single cell, for example, in *M. gryphiswaldense* the dimension of magnetosomes is around 45 nm (Schüler, 2004). This uniformity is an advantage of biogenic magnetic nanocrystals of MTB used for different bio(nano)technological application (Schüler and Frankel, 1999; Bazylinski and Frankel, 2004), as compared with biogenic magnetic nanocrystals produced by other types of bacteria or by artificial/abiogenic magnetic nanocrystals obtained by man using different physical/chemical protocols (Matsunaga, 1991; Schüler and Frankel, 1999). Biogenic magnetic nanocrystal can be produced by metabolic activities of dissimilatory iron-reducing bacteria and sulphate-reducing bacteria. This process is known as biologically induced mineralization. However, unlike the mineral particles in the magneto-tactic bacteria, biologically induced mineralization is not controlled by the organism and is characterized by no uniformity in size distributions and non-unique crystal habits.

We believe that the uniformity of biogenic magnetic nanocrystals of MTB can be further exploited for the construction of magnetic logic gates at nanometre level, with better results than the use of either abiotic magnetic nanocrystals or those produced by dissimilatory iron-reducing bacteria and sulphate-reducing bacteria.

It is still an enigma on how MM actually work in the process of controlled mineralization of iron during the process of magnetosome formation, but pioneering work had identified both at genetic and proteomic level the genes and magnetosome membrane proteins involved in magnetite formation. When the precise biological knowledge of the proteins/items involved in the magnetic nanocrystal formation will be achieved, it is expected that P systems could develop a model of this membrane processes, as it already started to carry out for other biological processes occurring at/within membranes: respiratory electron transport (Ardelean and Cavaliere, 2003; Ardelean et al., 2004; Cavaliere and Ardelean, 2006) the function of mechanosensitive channels (Ardelean et al., 2006) and many other processes (Ciobanu et al., 2006). medskip

Magnetotaxis. The passive orientation of MTB along the Earth's geomagnetic field lines is called magnetotaxis (Blakemore, 1975). Magnetotaxis is determined by the presence of magnetosomes. Dead cells containing magnetosomes also align along the geomagnetic field lines (around 0.05 mT), whereas alive MTB with no magnetosomes, do not align.

When magnetosomes are arranged in a single chain, as in the *Magnetospirillum* species, magnetostatic interactions between the single-magnetic domain particles cause the particle magnetic moments to spontaneously move parallel to each other along the chain direction. This results in a permanent magnetic dipole associated with the chain with a natural magnetization approaching the saturation magnetization and is sufficiently large enough to be oriented along the geomagnetic field at an ambient temperature (Frankel and Blakemore, 1980).

It is proposed that in natural environments magnetotaxis enables the cells to locate and maintain an optimal position in water columns or in sediments, with respect to their main metabolically needs: molecular oxygen and organic nutrients.

It is our hope that carefully deposition of MTB by the use of P systems-based models of magnetotactic behavior of intact cells during chemotaxis, and gently liberation of intact magnetosomes chains (retaining the surrounding membrane of each individual magnetosome particle) could be used for the construction of magnetic logic gates.

Magnetomanipulation of MTB. The orientation of MTB along the lines of magnetic field can be used to obtain in the lab surfaces covered by cells aligned with respect to the direction of the imposed magnetic field generated by two magnetic bars. This magnetomanipulation of magnetosome containing cells depends on their magnetotactic movement. In Figure 2 there are presented original images (optical microscope) concerning the experimental orientation of intact MTB on glass by the use of an external magnetic field; for the sake of simplicity the microscope images are accompanied by schematic drawings illustrating more clearly the orientation of MTB cells.

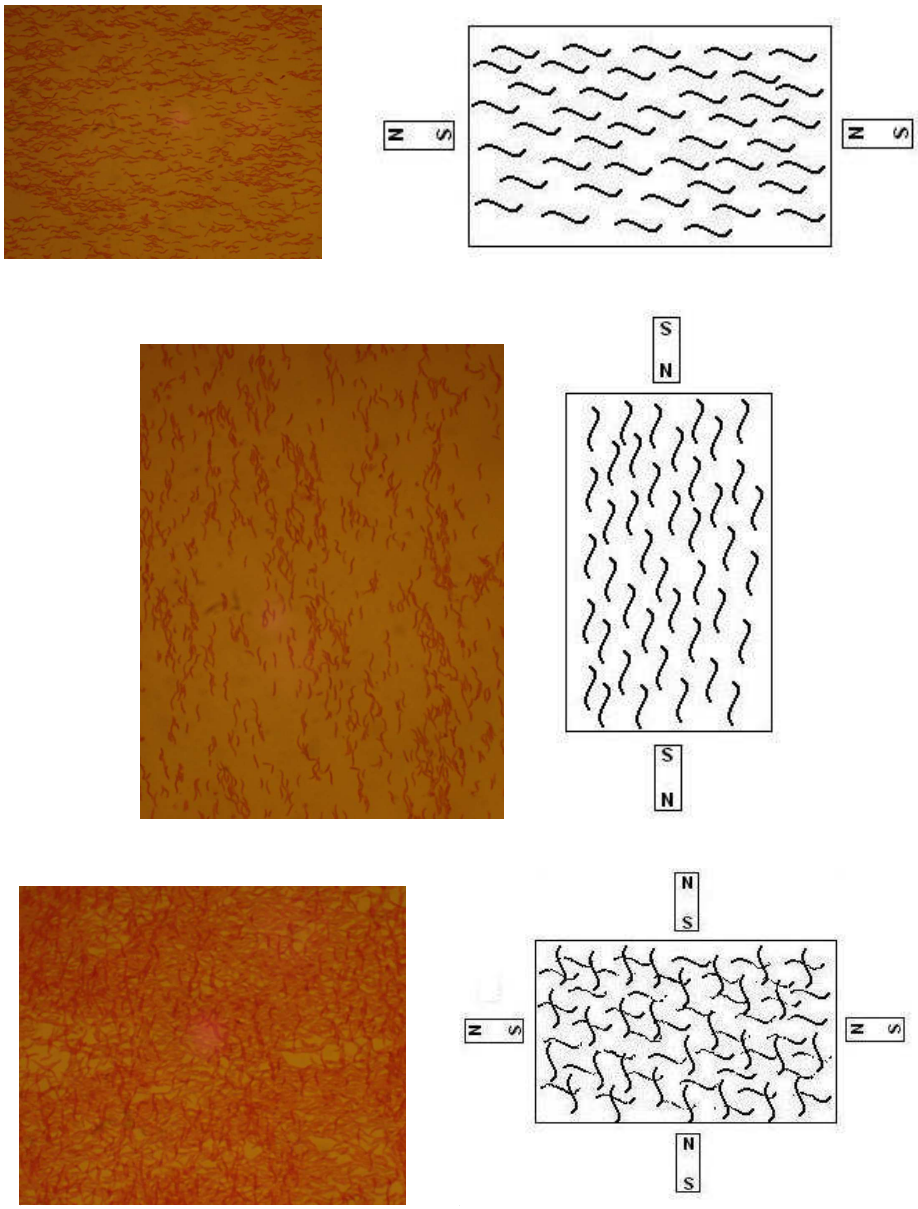


Fig. 2. The orientation of MTB along the lines of magnetic field; original optical microscope images (bright field, cells colored by basic fuxine) and schematic drawings: up = horizontal, center = vertical, and bottom = mixed (horizontal–vertical) orientation of MTB.

Controlled assembly of magnetic nanoparticles into ordered structures was demonstrated by manipulating magnetotactic bacteria in a fluid with microelectromagnets (Lee et al., 2004). The advantage of using magnetotactic bacteria cells is that the cellular bodies enclosing the magnetic chains prevent the magnetic aggregation of the bacteria, making it possible to use the bacteria as a carrier of ordered magnetic nanoparticles.

Microelectromagnets, consisting of multiple layers of lithographically patterned conductors, generate versatile magnetic fields on micrometer length scales, allowing sophisticated control of magnetotactic bacteria inside a microfluidic chamber (Lee et al., 2004). A single bacterium was stably trapped and its orientation was controlled; multiple groups of bacteria were assembled in a fluid. After positioning the bacteria, their cellular membranes were removed by cell lysis, leaving a chain and a ring of magnetic nanoparticles on a substrate. Thus, the authors demonstrated that the magnetic nanoparticles grown by the bacteria can be assembled into ordered structures. The new proposed approach, combining biomineralization and micromanipulation, can become a new method for growing and assembling nanoparticles into customized structures. Moreover, though integrated sensory means and new algorithms, the magnetotactic bacteria-based system could adapt or change the direction of motion from new occurring conditions (Lee et al., 2004).

It is our claim that a P system-based program could be more appropriate to model and control the direction of motion of these magnetotactic bacteria and to obtain ordered cells with magnetosomes useful for the construction of magnetic logic gates. The following picture suggests the use of a carefully designed (and constructed!) microdevice for true bacterial races!

The development of future autonomous bacterial microrobots (Martel, 2006) is another trend in which MTB can be involved. Acting like a compass, this chain of magnetosomes enables the bacteria to orient themselves and swim along the lines of a magnetic field. Hence, the basic control method consists of modifying the swimming paths of the MTB with the generation of local directional magnetic fields using small programmed electrical currents passing through special embedded conductor networks. This new method referred to as *controlled bacterial micro-actuation* is a serious candidate for its integration in future untethered microrobots operating in an aqueous medium, as originally proposed by the authors (Martel, 2006). The implementation of such bio-carriers with (non magnetic) micro-objects being propelled by a single MTB was also demonstrated. The effect of various diameters MTB-pushed beads on the velocity of this bio-carrier and the retarding effect caused by the proximity of the walls of the microchannels were also investigated. Thus by exploiting the motility of MTB, the electrical energy required to propel such a robot is null and the authors estimate that by pushing the limit of miniaturization or feature sizes to what is possible with actual microfabrication methods, a small current as low as 100 μA could be sufficient to control groups of pre-selected and most responsive MTB from a microcircuit embedded in the microrobot.

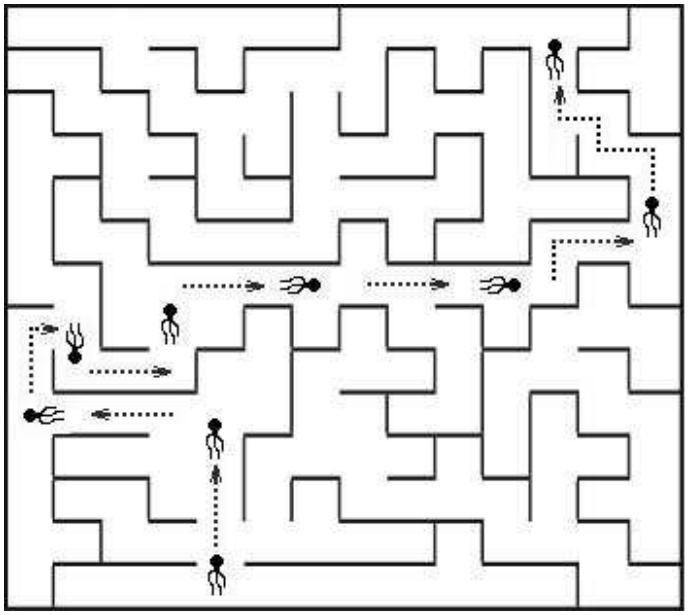


Fig. 3. Schematic picture of a microdevice for bacterial races (bacteriodrome?).

In our opinion such un-tethered microrobots could be used to transport biogenic magnetite produced by other bacterial cells, attached at their surface, and to release these nanocrystals at precise sites. Furthermore the concept of controlled bacterial micro-actuation (Martel, 2006) can be applied to obtain regular arrays of MTB which can be the basis for the construction of magnetic logic gates.

We have already designed a magneto-mechanical model of MTB with special emphasis on possible application in the field of nanoactuation (fore more details, see Ignat and Ardelean, 2004; Ignat et al. 2005, 2007). For example, the magnetosome chain microstructure can be moved and controlled with a 3 D system using magnetic levitation that represents an interesting microrobotic element.

The nanostructure of MTB suggests a nano- or micromanipulator structure which includes a flexible chain support with flexible joins and with magnetic elements which are small permanent magnets. These micromanipulator systems basically work within small magnetic gaps between the electromagnets (which generate the variable magnetic fields) and the motile chain. In Figure 4 there are presented some proposed structures for either flexible or rigid manipulators.

Our proposal to use magnetosomes, biogenic magnetic crystals covered by their biological membrane, as natural materials for the construction of magnetic logic gates and to construct nanoacutators based on either MTB or isolated intact magnetosomes could be helpful for the bottom up construction of a P systems-

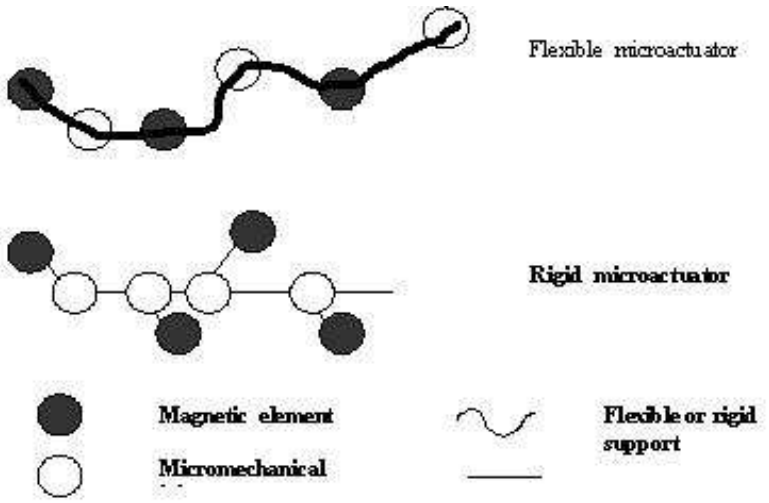


Fig. 4. Microarchitecture of nano or micro-magneto-manipulators.

based computer using natural components (biogenic magnetic nanocrystals) and P system-based software.

This proposal for the use of natural components for *in vitro* implementation of P systems and for the construction of a P system-based computer is in the line with the progresses made in the last four decades in incorporating different biological molecules into artificial membranes (Ottowa and Tien, 2002) which opened the way towards an *in vitro* implementation of P systems (see Ardelean, 2006, for more details).

Acknowledgements

Thanks are due to CEEX-MATNANTCH PROGRAME for partly supporting the work (Contract 2037/2006) and to both CEEX-MATNANTCH PROGRAME and THE ORGANIZING COMMITTEE for financial support to attend the 5th Brainstorming Week on Membrane Computing, Sevilla, 2007.

References

1. I.I. Ardelean (2006): Biological roots and applications of P systems. Further suggestions. In [12], 1–17.
2. I.I. Ardelean, D. Besozzi, M.H. Garzon, G. Mauri, S. Roy (2006): P system models for mechanosensitive channels. In [8], 43–81.
3. I.I. Ardelean, D. Besozzi, C. Manara (2004): Aerobic respirations a bio-logic circuit containing molecular logic gates. In *Pre-Proc. of Fifth Workshop on Membrane*

Computing, WMC5 (G. Mauri, Gh. Păun, C. Zandron, eds.), Universita di Milano-Bicocca, June 14-16, 2004, 119–125.
4. I.I. Ardelean, M. Cavaliere (2003): Modelling biological processes by using a probabilistic P system software. *Natural Computing*, 2 (2003), 173–197.
5. D.A. Bazylinski, R.B. Frankel (2004): Magnetosome formation in prokaryotes. *Nature Reviews*, 2 (2004), 217–230.
6. R.P. Blakemore (1975): Magnetotactic bacteria. *Science*, 190 (1975), 377–379.
7. M. Cavaliere, I. Ardelean (2006): Modelling respiration in bacteria and respiration/photosynthesis interaction in cyanobacteria. In [8], 129–159.
8. G. Ciobanu, M. Pérez-Jiménez, Gh. Păun, eds. (2006): *Applications of Membrane Computing*. Springer-Verlag, Berlin.
9. R.B. Frankel, D.A. Bazylinski, M.S. Johnson, B.L. Taylor (1997): Magneto-aerotaxis in marine coccoid bacteria. *Biophys. J.*, 73 (1997), 994–1000.
10. R.B. Frankel, R.P. Blakemore (1980): Navigational compass in magnetic bacteria. *J. Magnet Magnet Matter*, 15-18 (1980), 1562–1564.
11. S.A. Haque, M. Yamamoto, R. Nakatani, Y. Endo (2004): Magnetic logic gate for binary computing, *Science and Technology of Advanced Materials*, 5, 1-2 (2004), 79–82.
12. H.J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa, eds. (2006): *Proc. Workshop on Membrane Computing, WMC7, Leiden, The Netherlands*, LNCS 4361, Springer-Verlag, Berlin.
13. M. Ignat, I.I. Ardelean (2004): Distinct nanobiological structure: magnetotactic bacteria. Models and applications in the electromechanical nanoactuation. *Romanian Journal of Physics*, 9-10 (2004), 835–847.
14. M. Ignat, I.I. Ardelean, J. Pintea, Cr. Cojocaru (2005): Experimental aspects and magnetic characterization of *Magnetospirillum gryphiswaldense*. *Proceedings of The 4th National Conference on New Research Trends in Material Science*, ARM 4, September 2005, vol. I, 433–441.
15. M. Ignat, G. Zărnescu, S. Soldan, I.I. Ardelean, C. Moisescu (2007): Magneto-mechanic model of the magnetotactic bacteria. Applications in the microactuator field. *Journal of Optoelectronics and Advanced Materials*, 9, 4 (2007), 1169–1172.
16. H. Lee, A.M. Purdon, V. Chu, R.M. Westervelt (2004): Controlled assembly of magnetic nanoparticles from magnetotactic bacteria using microelectromagnets arrays. *Nano Lett.*, 4, 5 (2004), 995–998.
17. A. Leporati, S. Felloni (2007): Three "quantum" algorithms to solve 3-SAT. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.
18. A. Leporati, G. Mauri, C. Zandron (2005): Quantum sequential P systems with unit rules and energy assigned to membranes. In *Membrane Computing: 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005* (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 3850, Springer-Verlag, Berlin, 310–325.
19. V. Manca (2006): MP Systems approaches to biochemical dynamics: biological rhytms and oscillations. In [12], 86–100.
20. S. Mann, N.H.C. Sparks, R.G. Board (1990): Magnetotactic bacteria: microbiology, biomineralization, palaeomagnetism and biotechnology. *Adv. Microbiol. Physiol*, 31 (1990), 125–181.
21. S. Martel (2006): Controlled bacterial micro-actuation. In *Proc. of the Int. Conf. on Microtechnologies in Medicine and Biology*, Okinawa, Japan, May 9-12, 2006.
22. T. Matsunaga (1991): Applications of bacterial magnets. *Trends Biotechnology*, 9 (1991), 91–95.

23. A. Ottova, H.T. Tien (2002): The 40th anniversary of bilayer lipid membrane research. *Bioelectrochemistry*, 56 (2002), 171–173.
24. Gh. Păun (2000): Computing with membranes. *Journal of Computer and Systems Sciences*, 61 (2000), 108–143.
25. Gh. Păun (2001): From cells to computers. Computing with membrane (P systems). *BioSystems*, 59 (2001), 139–158.
26. Gh. Păun (2002): *Membrane Computing. An Introduction*. Springer-Verlag, Berlin.
27. M.J. Pérez-Jiménez, F.J. Romero-Campero (2006): P systems, a new computational modelling tool for Systems Biology. *Transactions on Computational Systems Biology, VI. Lecture Notes in Bioinformatics*, 4220, 176–197.
28. D. Schüler (2002): The biomineralization of magnetosomes in *Magnetospirillum gryphiswaldense. Int. Microbiol.*, 5 (2002), 209–214.
29. D. Schüler (2004): Molecular analysis of a subcellular compartment: the magnetosome membrane in *Magnetospirillum gryphiswaldense. Arch. Microbiol.*, 181 (2004), 1–7.
30. D. Schüler, R.B. Frankel (1999): Bacterial magnetosomes: microbiology, biomineralization and biotechnological applications. *Appl. Microbiol. Biotechnol.*, 52 (1999), 464–473.

Networks of Cells and Petri Nets

Francesco Bernardini[1], Marian Gheorghe[2],
Maurice Margenstern[3], Sergey Verlan[4]

[1] Leiden Institute of Advanced Computer Science, Universiteit Leiden
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
bernardi@liacs.nl
[2] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
M.Gheorghe@dcs.shef.ac.uk
[3] Université Paul Verlaine - Metz, UFR MIM, LITA, EA 3097
Ile du Saulcy, 57045 Metz Cédex, France
margens@univ-metz.fr
[4] LACL, Département Informatique, Université Paris 12
61 av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Summary. We introduce a new class of P systems, called networks of cells, with rules allowing several cells to simultaneously interact with each other in order to produce some new objects inside some other output cells. We define different types of behavior for networks of cells by considering alternative strategies for the application of the rules: sequential application, free parallelism, maximal parallelism, locally-maximal parallelism and minimal parallelism. We devise a way for translating network of cells into place-transition nets with localities (PTL-nets, for short) - a specific class of Petri nets. Then, for such a construction, we show a behavioral equivalence between network of cells and corresponding PTL-nets only in the case maximal parallelism, sequential execution, and free parallelism, whereas we observe that, in the case of locally-maximal parallelism and minimal parallelism, the corresponding PTL-nets are not always able to mimic the behavior of network of cells. Also, we address the reverse problem of finding a corresponding network of cells for a given PTL-net by obtaining similar results concerning the relationships between their semantics. Finally, we present network-of-cells-based models of two classical synchronization problems: producer/consumer and dining philosophers.

1 Introduction

Membrane computing is an emerging branch of natural computing which deals with distributed and parallel computing devices of a bio-inspired type, which are called membrane systems or P systems (see [17], [18], and also [1] for a comprehensive bibliography). P systems, originally devised by Gh. Păun in [17], are introduced as computing devices which abstract from the structure and functioning

of living cells - they are defined as a hierarchical arrangement of regions delimited by membranes (membrane structure), with each region having associated a multiset of objects and a finite set of rules. Rules typically encode mechanisms for consuming/producing objects (evolution) and mechanisms for moving objects across the membranes (communication). For consuming/producing objects, multiset rewriting (i.e., replacing a multiset with another one) is the most generally used mechanism, whereas, for communication, various mechanisms with different biological inspiration have been proposed such as: targets $here, in, out$ [18], symport/antiport [18], conditional uniport [25], boundary rules [4], and carriers [16]. In particular, symport/antiport, conditional uniport and boundary rules introduce in P systems the concept of coupled transport: communication is achieved through cooperation between two or more objects possibly placed in two distinct regions - the inside and the outside of a membrane. The class of P systems was later extended to tissue P systems [15, 19] – a variant of P systems where the underlying structure is defined as an arbitrary graph. Nodes in the graph represent cells which are able to communicate objects alongside the edges of this graph. Specifically, for this communication, the mechanisms of symport/antiport and conditional uniport are transferred to tissue P systems [18, 25] so to have models where communication is achieved through interactions between two neighboring cells (i.e., two cells which are directly connected by means of an edge in the underlying graph). Moreover, it is possible to have tissue P systems where a cell receives objects from a neighboring cell non-deterministically chosen [3], or where a cell produces signals which are replicated and simultaneously sent to all neighboring cells [15].

The work done in [7, 13, 12] then showed that the aforementioned features of transformation and communication in P systems can be interpreted as transitions in place-transition nets (PT-nets, for short) – a specific class of Petri nets (e.g., see [21, 22, 8, 11]). This is done by mapping each rule into a transition with places corresponding to the left-hand side of the rule as input, and with places corresponding to the right-hand side of the rule as output; each place in fact represent the occurrence of a certain object inside a certain membrane. Thus, production/consumption of objects and movement of objects across the membranes are both reflected into modifications on the distributions of tokens inside the places of a PT-net. Specifically, this construction was initially applied in [7] to P systems with boundary rules - which encompass symport/antiport and conditional uniport too - and then re-used in [13, 12] for the basic model of P systems where communication is controlled by targets $here, in, out$. Moreover, contributions [13, 12] devise a formal framework for describing the behavior of P systems in terms of causality/concurrency and for reasoning about reachability, conflicts and soundness of these systems by starting from their translation into PT-nets; this is done by using the PT-net representation of a membrane system is therefore to define the semantics of these systems in terms of sequences of events which consume some resources in order to produce some new ones (process semantics). This construction, which is a standard asynchronous PT-net, is extended in [12, 13] to PT-nets operating in a maximally parallel way and to PT-nets with localities operating in a locally-

maximal parallel way. PT-nets with localities are a class of PT-nets introduced in [13] where each transition belongs to certain location, in a way that resembles the distribution of the rules over the various regions of a membrane system. This makes possible to distinguish between the globally and locally synchronous behavior (maximal parallelism), globally asynchronous but locally synchronous behavior (locally-maximal parallelism), and asynchronous behavior.

In this paper, we introduce a new class of P systems which we call networks of cells. Network of cells are characterized by rules which allow several cells to simultaneously interact with each other in order to produce some new objects inside some other output cells. They are motivated by the observation made in [24] that basic forms of coupled transport like symport/antiport can be expressed in terms of two cells synchronizing on certain inputs in order to produce some outputs. In this respect, networks of cells are not limited to have only two synchronizing cells on their left-hand side and two output cells on their right-hand side. Then, similarly to what was done [13, 12], we define different types of behavior for networks of cells by considering different strategies for the application of the rules: sequential application, free parallelism, maximal parallelism and locally-maximal parallelism plus minimal parallelism [6]. Next, we extend to networks of cells the construction devised in [7, 13, 12] for translating membrane systems into PT-nets with localities (PTL-net, for short) by showing that interaction rules of networks of cells can still be represented as transitions of PTL-nets. However, we are able to establish a behavioral equivalence between network of cells and corresponding PTL-nets only in the case maximal parallelism, sequential execution, and free parallelism (i.e., only for globally and locally synchronous behavior, and asynchronous behavior) as we observe that, in the case of locally-maximal parallelism and minimal parallelism, the corresponding PTL-net is not always able to mimic the behavior of the original network of cells. This allows us to point out differences between the concept of cells used in our model and that of locality introduced in [13] for PT-nets. Also, we address the reverse problem of finding a corresponding network of cells for a given PTL-net by obtaining similar results concerning the relationships between their semantics. Finally we present network-of-cells-based models of two classical synchronization problems: producer/consumer and dining philosophers. These are devised by starting from existing PT-net solutions with the aim of illustrating differences in the two modeling approaches.

2 Preliminaries

We recall some basic notions concerning strings and multisets (e.g., see [23, 18] for further details).

An *alphabet* is any finite and non-empty set. The elements of an alphabet are called *symbols*. Let V be an alphabet. A *string* over V is any finite sequence consisting of zero or more symbols from V; the same symbol may occur repeated several times inside the same string. The sequence containing no symbols is called

empty string and it is denoted by λ. The set of all strings (respectively of all non-empty strings) over V is denoted by V^* (respectively V^+). If $x, y \in V^*$, then their *catenation* is $xy \in V^*$ (the catenation of two strings is the string obtained by the juxtaposition of the two strings, that is, by writing one string after the other one). Catenation is an associative operation and the empty string λ acts as an identity: $x\lambda = \lambda x = x$, for all $x \in V^*$. Moreover, for any $i \geq 1$, we denote by x^i the catenation of i copies of the string x; we set $x^0 = \lambda$ by definition. The *length of a string* $x \in V^*$, denoted by $|x|$, is the number of all occurrences in x of symbols from V; the number of occurrences in x of a symbol $a \in V$ is denoted by $|x|_a$. Yet again, by definition, we set $|\lambda| = 0$ and $|\lambda|_a = 0$, for all $a \in V$. The set of symbols from V occurring in a string x is denoted by $\aleph(x)$. We also use V to denote the set of strings from V^* of length equal to 1.

Let V be an alphabet. A *(finite) multiset (over V)* is a mapping $M : V \longrightarrow \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers (0 included); for every $a \in V$, $M(a)$ is called the *multiplicity of a (in M)*. A multiset over V is usually given in the form of a string over V; each string $x \in V^*$ in fact identifies a multiset M such that, for every $a \in V$, $M(a) = |x|_a$. On the other hand, every multiset M over V is representable by means of any string $x \in V^*$ such that, for every $a \in V$, $|x|_a = a$ – the order of the symbols in such a string is not important. Therefore, from this moment on, we will use strings to represent multisets and, given $x \in V^*$, we will use the expression "multiset x" to refer to a multiset representable by means of string x. Thus, for $x \in V^*$, for $a \in V$, the multiplicity of a in x is $|x|_a$, and the *size of multiset* x (i.e., the sum of all multiplicities) is the value $|x|$. Moreover, for $x, y \in V^*$, we say that multiset x and multiset y are equal, and we write $x = y$, if, for all $a \in V$, $|x|_a = |y|_a$. We say that multiset x *includes* multiset y, and we write $x \sqsupseteq y$, if, for all $a \in V$, we have $|x|_a \geq |y|_a$. If that is the case, we also say that y *is included* in x, and we write $y \sqsubseteq x$. The union of multiset x and multiset y, denoted by $x \sqcup y$, is a multiset w such that, for all $a \in V$, $|w|_a = |x|_a + |y|_a$.

The notion of a rewriting rule between strings can be naturally transferred to multisets. A *multiset rewriting rule* is a pair (u, v), with u, v two multisets, which is written in the form $u \rightarrow v$. Given a multiset w and a rewriting rule $u \rightarrow v$, if $u \sqsubseteq w$, then the rule $u \rightarrow v$ is *applicable* to the multiset w; if that is the case, the result of the application of the rule $u \rightarrow v$ to the multiset w is the multiset w' such that, for all $a \in V$, $|w'|_a = |w|_a - |u|_a + |v|_a$. If that is case, then we also say that the multiset w *can evolve* by means of the multiset rewriting rule $u \rightarrow v$.

3 Networks of Cells

Here we introduce a general model of membrane systems which allows us to capture the essential features of most variants of cell-like P systems and tissue P systems.

Definition 1 (network of cells). *A* network of cells of degree $n \geq 1$ *(an* NC of degree $n \geq 1$*, for short) is a construct:*

$$\Pi = (V, w_1, w_2, \ldots, w_n, R),$$

where:

1. *V is an* alphabet*;*
2. *$w_i \in O^*$, for all $1 \leq i \leq n$, is the multiset* initially associated to cell *i;*
3. *R is a finite set of* interaction rules *of the form*

$$(u_1, i_1) \dots (u_k, i_k) \rightarrow (v_1, j_1) \dots (v_h, j_h)$$

where (a) $1 \leq k, h \leq n$, (b) for all $1 \leq l, l' \leq k$, $u_l \in V^+$, (c) $1 \leq i_l \leq n$ and $l \neq l'$ implies $i_l \neq i_{l'}$, (d) for all $1 \leq r, r' \leq h$, $v_r \in V^*$, $1 \leq j_r \leq n$, and $r \neq r'$ implies $j_r \neq j_{r'}$.

A network of cells consists of n cells numbered from 1 to n with each one of them containing a multiset of objects over V (initially cell i contains multiset w_i). Cells can interact with each other by means of the rules in R. An interaction rule of the form $(u_1, i_1) \dots (u_k, i_k) \rightarrow (v_1, j_1) \dots (v_h, j_h)$ specifies that, whenever, at the same time, for all $1 \leq l \leq k$, cell i_l contains at least one occurrence of multiset u_l, an occurrence of the multiset u_l is consumed inside cell i_l, for all $1 \leq l \leq k$, and a multiset v_r is produced inside cell j_r, for all $1 \leq r \leq h$. In other words, an interaction rule simultaneously rewrites some multisets inside cells $i_1, \dots, i_k$ in order to produce some new multisets inside cells $j_1, \dots, j_h$. Notice also that, for an interaction rule $(u_1, i_1) \dots (u_k, i_k) \rightarrow (v_1, j_1) \dots (v_h, j_h)$, for all $1 \leq l, l' \leq k$, we have $u_l \in V^+$ (i.e., a multiset on the left-hand side of the rule cannot be empty) and $l \neq l'$ implies $i_l \neq i_{l'}$ (i.e., the left-hand side of the rule must involve a set of distinct cells), and, for all $1 \leq r, r' \leq h$, we have $v_r \in V^*$ (i.e., a multiset on the right-hand side of the rule can be empty) and $r \neq r'$ implies $j_r \neq j_{r'}$ (i.e., the right-hand side of the rule must involve a set of distinct cells).

For an interaction rule ρ of the form $(u_1, i_1) \dots (u_k, i_k) \rightarrow (v_1, j_1) \dots (v_h, j_h)$, cells $i_1, \dots, i_k$ are called *input cells*, the set $\{i_1, \dots, i_k\}$ is denoted by $Input(\rho)$ and k is called the *input radius of* ρ; cells $j_1, \dots, j_h$ are called *output cells*, the set $\{j_1, \dots, j_h\}$ is denoted by $Output(\rho)$ and h is called the *output radius of* ρ; the *cooperation degree of* ρ is the value $\max\{|u_i| \mid 1 \leq i \leq k\}$; the left-hand side $(u_1, i_1) \dots (u_k, i_k)$ is denoted by $lhs(\rho)$ whereas the right-hand side $(v_1, j_1) \dots (v_h, j_h)$ is denoted by $rhs(\rho)$. Also, for such a rule ρ, for all $1 \leq i \leq n$, we use $lhs_i(\rho)$ to denote the multiset u if $(u, i) \in lhs(\rho)$, or λ if $i \notin Input(\rho)$; we use $rhs_i(\rho)$ to denote the multiset v if $(v, i) \in rhs(\rho)$, or λ if $i \notin Output(\rho)$.

Notice that the structure of an NC corresponds neither to a tree as in cell-like P systems nor to a graph as in tissue P systems (e.g., see [18] for definitions of cell-like P systems and tissue P systems), though some models of cell-like P systems and tissue P systems can be seen as special variants of NC's. The possibility of representing existing variants of P systems as NC's is illustrated through the following examples.

Example 1. A basic P system [18] is defined as a hierarchical arrangement of membranes; each membrane delimits a region which contains a multiset of objects and finite set of evolution rules. If V is the alphabet of the system, then an evolution

rule has the form $u \to (u_1, t_1)(u_2, t_2) \dots (u_q, t_q)$ with $u \in V^*$, for all $1 \leq r \leq q$, $u_r \in V^*$ and $t_r \in \{here, out, in_j\}$. If such a rule is associated to a region i, then multiset u can be replaced by multisets $u_1, u_2, \dots, u_q$ and each multiset u_r is moved across the membranes depending on the target t_r: u_r remains inside membrane i when $t_r = here$, u_r is moved outside membrane i when $t_r = out$, u_r is moved into region j when $t_r = in_j$ with j a membrane directly contained into region i.

A basic P system can be represented as an NC which has as many cells as the membranes in the P system and which contains, for every region i of the P systems, for every evolution rule $u \to (u_1, t_1)(u_2, t_2) \dots (u_q, t_q)$ associated to region i, an interaction rule $(u, i) \to (u_1, j_1)(u_2, j_2) \dots (u_q, j_q)$, with distinct $j_1, j_2, \dots, j_q$, such that, for all $1 \leq r \leq q$, if $t_r = here$, then $j_r = i$; if $t_r = in_j$, then $j_r = j$; if $t_r = out$, then j_r is equal to the index of the directly upper region.

Example 2. P systems with boundary rules [4] extend basic P systems with rules which allow direct interactions between the inside and the outside of a region. In their most general form (e.g., see [5]), boundary rules are of the form $u \lfloor_i v \to u' \lfloor_i v'$ with i the index of a membrane in the system and $u, v, u', v' \in V^*$, for V the alphabet of the system.

These rules can be represented in NC's as interactions of the form $(u, j)(v, i) \to (u', j)(v', i)$ with j the membrane which directly contains membrane i. In this way, we can for instance capture the features of symport/antiport rules [18]:

- an antiport rule $(x, in; y, out)$ associated to membrane i is no more than an interaction rule $(x, j)(y, i) \to (y, j)(x, i)$ with j the membrane which directly contains membrane i;
- a symport rule (x, out) associated to membrane i is no more than an interaction rule $(x, i) \to (x, j)$ with j the membrane which directly contains membrane i;
- a symport rule (x, in) associated to membrane i is no more than an interaction rule $(x, j) \to (x, i)$ with j the membrane which directly contains membrane i.

Example 3. An evolution-communication model of tissue P systems is proposed in [3] that is based on graphs of cells where each cell contains a multiset of objects and a finite set of rules of the forms $x \to y$ (transformation rules) and $(x; y, in)$ (communication rules) with x, y two multisets over a given alphabet. A transformation rule $x \to y$ associated to a cell i specifies that a multiset x placed inside cell i can be replaced by a multiset y which remains inside cell i. A communication rule $(x; y, in)$ associated to a cell i instead specifies that, in presence of a multiset x, a multiset y can be moved from a neighboring cell j non-deterministically chosen into cell i.

Such a tissue P system can be represented as an NC with the same number of cells which contains an interaction rule $(x, i) \to (y, i)$, for every transformation rule $x \to y$ associated to cell i of the tissue P system, and a set of interaction rules $\{(x, i)(y, j) \to (xy, j) \mid \{i, j\}$ is an edge of the graph underlying the tissue P system$\}$, for every communication rule $(x; y, in)$ in the tissue P system associated to cell i.

We now pass to precisely define the execution semantics of networks of cells by identifying different strategies for the application of the rules. To this aim, we first give the following definitions.

Definition 2 (configuration). *Let $\Pi = (V, w_1, w_2, \ldots, w_n, R)$ be a network of cells. A* configuration *of Π is any tuple $(w'_1, \ldots, w'_n)$ with $w_i \in O^*$, for all $1 \leq i \leq m$. The* initial configuration of Π *is the tuple $(w_1, \ldots, w_n)$.*

Definition 3 (multiset of applicable rules). *Let $\Pi = (V, w_1, w_2, \ldots, w_n, R)$ be a network of cells and let $C = (w'_1, \ldots, w'_n)$ be a configuration of Π. A* multiset of applicable rules (w.r.t. Π and C) *is any function $\Gamma_C : R \to \mathbb{N}$ such that, for all $1 \leq i \leq n$, $(\bigsqcup_{r \in R}(lhs_i(r))^{\Gamma_C(r)}) \sqsubseteq w'_i$.*

Free parallelism

Free parallelism means that, in each step, any multiset of applicable rules can be used to make an NC transit from a configuration to another one by applying all the selected rules in parallel at the same time. In membrane computing literature, this semantics is also called asynchronous behavior (e.g., see [9]).

Specifically, let $\Pi = (V, w_1, w_2, \ldots, w_n, R)$ be a network of cells and let $C = (w'_1, \ldots, w'_n)$, $C' = (w''_1, \ldots, w''_n)$ be two configurations of Π. We say that Π transits in one step from configuration C to configuration C' in a freely-parallel way, and we write $C \Rightarrow_{free} C'$, if there is a multiset of applicable rules Γ_C such that, for all $1 \leq i \leq n$, $w''_i = (w'_i \setminus (\bigsqcup_{r \in R}(lhs_i(r))^{\Gamma_C(r)})) \cup (\bigsqcup_{r \in R}(rhs_i(r))^{\Gamma_C(r)})$.

Thus, in a freely-parallel step, an arbitrary multiset of applicable rules is selected and these rules are applied in parallel by consuming all the multisets on their left-hand sides and producing all the multisets on their right-hand sides in the respective places.

Sequential execution

Sequential execution means that, in each step, only one rule is applied to make an NC transit from a configuration to another one.

Specifically, let $\Pi = (V, w_1, w_2, \ldots, w_n, R)$ be a network of cells and let $C = (w'_1, \ldots, w'_n)$, $C' = (w''_1, \ldots, w''_n)$ be two configurations of Π. We say that Π transits in one step from configuration C to configuration C' in a sequential way, and we write $C \Rightarrow_{seq} C'$, if $C \Rightarrow_{free} C'$ for some multiset of applicable rules Γ_C with $|\Gamma_C| = 1$.

Thus, a sequential step is a freely-parallel step where the number of rules used is equal to 1.

Maximal parallelism

Maximal parallelism means that, in each step, any maximal multiset of applicable rules can be used to make an NC transit from a configuration to another one by

applying all the selected rules in parallel at the same time; a maximal multiset of applicable rules is any multiset of applicable rules to which no other rules can be added so to obtain another multiset of applicable rules. Maximal parallelism is the type of behavior which was associated to membrane systems in their original definition [17], and it is the semantics most commonly adopted in the area of membrane computing (e.g., see [1], [18]).

Specifically, let $\Pi = (V, w_1, w_2, \ldots, w_n, R)$ be a network of cells and let $C = (w'_1, \ldots, w'_n)$, $C' = (w''_1, \ldots, w''_n)$ be two configurations of Π. We say that Π transits in one step from configuration C to configuration C' in a maximally-parallel way, and we write $C \Rightarrow_{max} C'$, if $C \Rightarrow_{free} C'$ for some multiset of applicable rules Γ_C such that, for all $r \in R$, there is $1 \leq i \leq n$ with $lhs(r)_i \neq \lambda$ and $lhs_i(r) \not\sqsubseteq (w'_i \setminus (\bigsqcup_{r \in R}(lhs_i(r))^{\Gamma_C(r)}))$.

Thus, a maximally-parallel step is a freely-parallel step where rules are applied in parallel in an exhaustive way: once the multisets on the left-hand side of these rules are consumed, no other rule has to be applicable to the objects left inside the cells.

Locally-maximal parallelism

Locally-maximal parallelism, which was introduced in [13], specifies that, in each step, if a cell is involved in the application of (at least) one rule, then a maximal number of objects are consumed in this cell by applying in parallel as many rules that involve this cell as possible.

Specifically, let $\Pi = (V, w_1, w_2, \ldots, w_n, R)$ be a network of cells and let $C = (w'_1, \ldots, w'_n)$, $C' = (w''_1, \ldots, w''_n)$ be two configurations of Π. We say that Π transits in one step from configuration C to configuration C' in a locally-maximally-parallel way, and we write $C \Rightarrow_{lmax} C'$, if $C \Rightarrow_{free} C'$ for some multiset of applicable rules Γ_C such that, for all $1 \leq i \leq n$, if there is $r' \in R$ with $\Sigma_C(r') > 0$ and $lhs_i(r') \neq \lambda$, then, for all $r \in R$ with $lhs(r)_i \neq \lambda$, there is $1 \leq j \leq n$ with $lhs(r)_j \neq \lambda$ and $lhs_j(r) \not\sqsubseteq (w'_j \setminus (\bigsqcup_{r \in R}(lhs_j(r))^{\Gamma_C(r)}))$.

Thus, a locally-maximally parallel step is a freely-parallel step where if a cell is involved in the application of one rule, then a maximal number of rules involving this cell is applied in parallel at the same time. In other words, in a locally-maximally parallel step, every cell that gets involved tries to participate in as many interactions as possible depending on the objects currently available inside the cell and on the presence of other cells competing for the same objects.

Minimal parallelism

Minimal parallelism, which was introduced in [6], means that, in each step, every cell that can participate in at least one interaction must get involved in the application of at least one rule.

Specifically, let $\Pi = (V, w_1, w_2, \ldots, w_n, R)$ be a network of cells and let $C = (w'_1, \ldots, w'_n)$, $C' = (w''_1, \ldots, w''_n)$ be two configurations of Π. We say that

Π transits in one step from configuration C to configuration C' in a minimally-parallel way, and we write $C \Rightarrow_{min} C'$, if $C \Rightarrow_{free} C'$ for some multiset of applicable rules Γ_C such that, for all $1 \leq i \leq n$, if there is no $r' \in R$ with $\Gamma_C(r') > 0$ and $lhs_i(r') \neq \lambda$, then, for all $r \in R$ with $lhs(r)_i \neq \lambda$, there is $1 \leq j \leq n$ with $lhs(r)_j \neq \lambda$ and $lhs_j(r) \not\sqsubseteq (w'_j \setminus (\bigsqcup_{r \in R}(lhs_j(r))^{\Gamma_C(r)}))$.

Thus, a minimally-parallel step is a freely-parallel step where a maximal number of cells, which are selected depending on the current distribution of objects inside the cells, evolve in parallel at the same time; each one of the selected cells has to participate in at least one interaction, and no other rule has to be applicable in parallel at the same time that involve any cell different from the selected ones.

Example 4. Let $UNO = (V, aa, aa, bb, c, R)$ be an NC where R contains the following interaction rules:

1. $(a,1) \rightarrow (a,1)(a,2)$,
2. $(a,1) \rightarrow (a,1)(b,3)$,
3. $(c,4)(a,2)(b,3) \rightarrow (b,2)(a,3)$,
4. $(b,2) \rightarrow (b,2)(c,4)$,
5. $(c,4)(b,3) \rightarrow (cc,4)(b,3)$,
6. $(c,4)(b,2) \rightarrow (bb,3)$,
7. $(c,4)(a,1) \rightarrow (a,4)(a,2)(a,3)$.

This NC has rules with input radius at most 2 and cooperation degree 1. The initial configuration is given by the tuple $C_0 = (aa, aa, bb, c)$. To the initial configuration, we can apply rule 1 with multiplicity at most 2, rule 2 with multiplicity at most 2, rule 3 with multiplicity at most 1, rule 5 with multiplicity at most 1, and rule 7 with multiplicity at most 1. However, with respect to C_0, it is not possible to apply all these rules in parallel (e.g., only one rule between rules 3, 5 and 7 can be applied because cell 4 contains only one object c).

Thus, in the case of free-parallelism, for any configuration C' obtained by applying any combination of the aforementioned rules that is a multiset of applicable rules, we have $C_0 \Rightarrow_{free} C'$. For instance, if rule 1 is applied with multiplicity 2 and rule 3 is applied with multiplicity 1, we have $C_0 \Rightarrow (aa, aaab, ab, \lambda)$.

In the case of sequential execution, only one of the aforementioned rules is going to be applied with multiplicity 1 to the initial configuration. This gives us five possible transitions: $C_0 \Rightarrow_{seq} (aa, aaa, bb, c)$ when rule 1 is applied, $C_0 \Rightarrow_{seq} (aa, aa, bbb, c)$ when rule 2 is applied, $C_0 \Rightarrow_{seq} (aa, ab, ab, \lambda)$ when rule 3 is applied, $C_0 \Rightarrow_{seq} (aa, aa, bb, cc)$ when rule 5 is applied and $C_0 \Rightarrow_{seq} (a, aaa, abb, a)$ when rule 7 is applied.

If maximal parallelism is adopted, then a maximal number of the aforementioned rules is applied to the initial configuration C_0. Specifically, we have the following possibilities: rule 1 (rule 2) applied with multiplicity 2 in parallel with another rule chosen between rules 3 and 5; rule 1 applied in parallel with rule 2 (both with multiplicity 1) together with another rule chosen between rules 3 and

5; rule 1 (rule 2) applied with multiplicity 1 in parallel with rule 7. For instance, if we choose this latter combination, we have $C_0 \Rightarrow_{max} (a, aaaa, abb, a)$.

In the case of locally-maximal parallelism, we have to make sure that if a cell evolves by means of at least one rule, then all other rules affecting that same cell that can be applied in parallel are effectively applied within the same step of execution. Specifically, for the initial configuration C_0, this means that whenever rule 1, or 2 (rule 7) is used, then another rule chosen between rules 1, 2 and 7 (rules 1 and 2) is always applied in parallel in the same step. On the other hand, we have $C_0 \Rightarrow_{lmax} (aa, ab, ab, \lambda)$ by just applying rule 3 because there are no other rules involving cell 2, or 3 or 4 that can be applied in parallel at the same time. Also, we have $C_0 \Rightarrow_{seq} (aa, aa, bb, cc)$ by just applying rule 5 because there are no other rules involving cell 3 or 4 that can be applied in parallel at the same time.

In the case of minimal parallelism, a maximal number of cells evolve in parallel at the same time with each one of the selected cells participating in at least one interaction. Specifically, for the initial configuration C_0, this means that rule 3 (rule 5) is always used in parallel with at least an application of rule 1 or 2. However, with respect to C_0, we have $C_0 \Rightarrow_{min} (a, aaa, abb, aa)$ by just applying rule 7 because there are no rules involving cells 2 or 3 which can be applied in parallel at the same time.

Remark 1. Locally-maximal parallelism was introduced in [13, 12] only for the basic model of P systems where rules are precisely assigned to regions delimited by membranes and the left-hand side of every rule involves only objects inside the region which the rule is assigned to. Therefore, locally-maximal parallelism is defined in [13, 12] by simply stating that, in each step, a certain number of membranes is selected and a maximal number of rules is applied inside each one of these membranes. In the case of NC's, interaction rules may involve objects placed inside different cells, hence locally-maximal parallelism is defined with respect to a certain group of cells: in each step, a multiset of applicable rules can be used only if it is maximal with respect to the cells which appear on the left-hand side of the rules in it. However, if the rules of NC's are restricted to have input radius equal to 1, then the present notion of locally-maximal parallelism is consistent with the semantics given in [13, 12] for the basic model of P systems.

Remark 2. The minimally-parallel semantics for NC's is not defined in terms of number of rules which are applied inside the cells, as in [6], but it is defined with respect to the number of cells which can evolve in parallel at the same time. Specifically, in each minimally-parallel step, a multiset of applicable rules can be used only if there are no cells which do not appear on the left-hand side of any rule in it and which some rules can be applied to; the multiset of applicable rules has not to be maximal neither locally nor globally though. Minimal parallelism was instead defined in [6] for P systems with symport/antiport where every membrane has its own set of rules, hence, in each step, for each one of these sets of rules, if there is a rule which is applicable, then at least one rule from that set is going to be applied irrespective of the fact that an antiport rule involves two distinct

regions at the same time. Therefore, although symport/antiport can be expressed as interaction rules of NC's, the notion of minimal parallelism proposed in [6] for symport/antiport differs from the one considered here because interaction rules of NC's are not assigned a priori to any cell. However, if rules of NC's are restricted to have input radius equal to 1, then our minimally-parallel semantics for NC's is consistent with the idea from [6], that is, in each step, if at least one rule may be used inside a region, then at least one rule is applied inside that region.

Remark 3. From a computational point of view (in the usual sense of membrane computing), if we consider NC's operating according to maximal parallelism, then it is obvious that they are computationally complete and that the hierarchy on the number of cells collapses at level 1. Moreover, the universality results obtained for catalytic P systems and evolution-communication P systems (e.g., see [14], [3], [10]) can be directly transferred to NC's: NC's with rules of input radius at most 2 are computationally complete and, for such systems, the hierarchy on the number of cells collapses at level 2. More precisely, for NC's corresponding to catalytic P systems, we have universality for input radius at most 1 and cooperation degree at most 2 [10], whereas, for evolution-communication P systems, we have universality for input radius at most 2 and cooperation degree at most 1 when antiport rules are used, or for input radius at most 1 and cooperation degree at most 2 when symport rules are used [14]. On the other hand, the computational power of NC's operating in a sequential manner, in a freely-parallel manner, in a locally-maximal parallel manner, or in a minimally parallel manner requires further investigations.

4 PT Nets with Localities

We introduce the class of Petri nets called place-transition nets with localities in the form reported in [12].

Definition 4 (PTL-net). *A* PT-net with localities *(a* PTL-net*, for short) is a construct:*

$$N = (P, T, W, M_0, L),$$

where

1. *P is a finite set of symbols whose elements are called* places*,*
2. *T is a finite set of symbols whose elements are called* transitions*,*
3. *$W : (P \times T) \cup (T \times P) \to \mathbb{N}$ is the* weight function*,*
4. *$M_0 \in P^*$ is a multiset over P called the* initial marking*,*
5. *$L : T \to \mathbb{N}$ is a* locality mapping*;*

and such that $P \cap T = \emptyset$.

PTL-nets are usually represented by diagrams where places are drawn as circles, transitions are drawn as squares, and a directed arc (x, y) is added between x and y if $W(x, y) \geq 1$. Moreover, transitions are annotated with their localities and the

arcs are annotated with their weights if these are 2 or more. Localities are used to partition the set of transitions into subsets of transitions which logically belongs to distinct locations.

Given a PTL-net N, the *pre-* and *post-multiset* of a transition t are respectively the multiset $pre_N(t)$ and the multiset $post_N(t)$ such that, for all $p \in P$, $|p|_{pre_N(t)} = W(p,t)$ and $|p|_{post_N(t)} = W(t,p)$. A configuration of N, which is called a *marking*, is any multiset over P; in particular, for every $p \in P$, $|p|_M$ represents the number of *tokens* present inside place p. Then, we recall from [12] the notion of an execution mode for a PTL-net by giving the following definitions.

Definition 5 (free-enabled). *Let $N = (P, T, W, M_0, L)$ be a PTL-net and let M be a marking of N. A multiset of transitions $U \in T^*$ is* free-enabled *at marking M if $(\bigsqcup_{t \in T}(pre_N(t))^{|t|_U}) \sqsubseteq M$.*

Definition 6 (seq-enabled). *Let $N = (P, T, W, M_0, L)$ be a PTL-net and let M be a marking of N. A multiset of transitions $U \in T^*$ is* seq-enabled *at marking M if U is free-enabled at marking M and $|U| = 1$.*

Definition 7 (max-enabled). *Let $N = (P, T, W, M_0, L)$ be a PTL-net and let M be a marking of N. A multiset of transitions $U \in T^*$ is* max-enabled *at marking M if U is free-enabled at marking M and, for all $t \in T$, $pre_N(t) \not\sqsubseteq (M \setminus (\bigsqcup_{t \in T}(pre_N(t))^{|t|_U}))$.*

Definition 8 (lmax-enabled). *Let $N = (P, T, W, M_0, L)$ be a PTL-net and let M be a marking of N. A multiset of transitions $U \in T^*$ is* lmax-enabled *at marking M if U is free-enabled at marking M and, for all $l \in L(T)$, if there is $t \in T$ with $L(t) = l$ and $|U|_t > 0$, then, for all $t' \in T$ with $L(t') = l$, $pre_N(t') \not\sqsubseteq (M \setminus (\bigsqcup_{t \in T}(pre_N(t))^{|t|_U}))$.*

Definition 9 (min-enabled). *Let $N = (P, T, W, M_0, L)$ be a PTL-net and let M be a marking of N. A multiset of transitions $U \in T^*$ is* min-enabled *at marking M if U is free-enabled at marking M and, for all $l \in L(T)$, if there is no $t \in T$ with $L(t) = l$ and $|U|_t > 0$, then, for all $t' \in T$ with $L(t') = l$, $pre_N(t') \not\sqsubseteq (M \setminus (\bigsqcup_{t \in T}(pre_N(t))^{|t|_U}))$.*

Definition 10 (m-execution). *Let $N = (P, T, W, M_0, L)$ be a PTL-net and let* m $\in \{free, seq, max, lmax, min\}$. *The* m-execution *of N is the relationship $\rightsquigarrow_{\mathrm{m}} \subseteq P^* \times P^*$ such that, for all $M, M' \in P^*$, $M \rightsquigarrow_{\mathrm{m}} M'$ iff, $M' = (M \setminus (\bigsqcup_{t \in T}(pre_N(t))^{|t|_U}))$ for some $U \in T^*$ which is* m*-enabled at marking M.*

Thus, an m-execution of a PTL-net N represents a transition step which makes possible to derive a new marking from a given one by firing a certain number of transitions in parallel at the same time. The firing of each transition results in the consumption of its pre-multiset of places from the given marking and in the production of its post-multiset of places in the new marking; the number of transitions that can fire in parallel within a transition step depends on the execution

mode m $\in \{free, seq, max, lmax, min\}$ and it has to be consistent with the current availability of tokens inside each place (i.e., for each place, the number of its tokens consumed cannot be greater than its multiplicity in the current marking). Specifically, the aforementioned execution modes identify the following behaviors for a PTL-net:

- *free-execution*: in each transition step, an arbitrary number of transitions fire by providing that these constitute a free-enabled multiset of transitions (i.e., the union of their pre-multisets has to be contained in the current marking).
- *seq-execution*: in each transition step, only one transition fires that is chosen amongst those whose pre-multiset of place is contained in the current marking (i.e., in order to fire, a transition has to be enabled at the current marking);
- *max-execution*: in each transition step, a maximal number of transitions fire by providing that these constitute a free-enabled multiset of transitions which no other transition can be added to in order to obtain another free-enabled multiset of transitions (i.e., the selected transitions has to consume a maximal number of places so that no other transition can fire in parallel at the same time);
- *lmax-execution*: in each transition step, an arbitrary set of localities is selected and, for each one of these localities, a maximal number of transitions fire (i.e., for each selected locality, a maximal number of places is consumed so that no other transition belonging to the same locality can fire in parallel at the same time);
- *min-execution*: in each transition step, for each locality such that there is at least one enabled transition associated to that locality, at least one transition fire (i.e., the selected multiset of transitions has to involve a maximal set of localities, although the number of firing transition belonging to the same locality has not necessarily to be maximal with respect to the current marking).

Notice that seq-execution is called min-execution in [13, 12], whereas the present notion of min-execution is introduced as a counterpart of the minimally-parallel semantics previously used in the area of membrane computing [6]; this latter notion of minimal parallelism is in fact not considered in [13, 12].

5 Network of Cells versus PTL-nets

We start by extending to the class of network of cells the basic construction devised in [7, 12, 13] to transform membrane systems into "equivalent" PTL-nets.

Definition 11. *Let $\Pi = (V, w_1, w_2, \ldots, w_n, R)$ be a network of cells.*

1. *The* extended alphabet of Π*, denoted by E_Π, is the set* $\{(a, i) \mid a \in V, 1 \leq i \leq n\}$.
2. *For all* $1 \leq i \leq n$*, the* i labeling of Π *is the mapping* $h_{i,\Pi} : V^* \to E_\Pi^*$ *such that, for all* $a \in V$*,* $h_{i,\Pi}(a) = (a, i)$ *and, for all* $u, v \in V^*$*,* $h_{i,\Pi}(uv) = h_{i,\Pi}(u)h(v)$.

3. *For all* $r \in R$ *of the form* $(u_1, i_1) \dots (u_k, i_k) \rightarrow (v_1, j_1) \dots (v_h, j_h)$, *the* cell-labeled version of r, *denoted by* $CL(r)$, *is the (multiset rewriting) rule* $h_{i_1,\Pi}(u_1) \dots h_{i_k,\Pi}(u_k) \rightarrow h_{j_1,\Pi}(v_1) \dots h_{j_h,\Pi}(v_h)$.

Thus, for all $1 \leq i \leq n$, the i labeling assigns to every object of a given multiset the label i; the cell-labeled version of an interaction rule is a multiset rewriting rules where the localization of the multisets inside the cells is given by the labels assigned to the objects. This idea of assigning a label to the objects in order to represent their localization inside the cells is central to the following construction which shows how to define a corresponding PTL-net for every network of cells.

Definition 12 (corresponding PTL-net). *Let* $\Pi = (V, w_1, w_2, \dots, w_n, R)$ *be a network of cells where rules in* R *are labeled in a one-to-one manner with values in* $\{1, 2, \dots, |R|\}$. *The PTL-net corresponding to* Π *is* $\mathcal{N}(\Pi) = (E_\Pi, \{1, 2, \dots, |R|\}, W, M_0, L)$, *where: for all* $p \in E_\Pi$, $t \in \{1, 2, \dots, |R|\}$, $W(p,t) = m$ *iff* t *is the label of a rule* $r \in R$ *with* $m = |lhs(CL(r))|_p$, *and* $W(t,p) = m$ *iff* t *is the label of a rule* $r \in R$ *with* $m = |rhs(CL(r))|_p$; $M_0 = h_{1,\Pi}(w_1)h_{2,\Pi}(w_2) \dots h_{n,\Pi}(w_n)$; *for all* $t \in \{1, 2, \dots, |R|\}$, $L(t) = 0$.

Thus, an NC Π is transformed into a PTL-net which contains a place for each cell in Π and for each object possibly present inside this cell, and a transition for each rule in Π. More precisely, in the corresponding PTL-net, occurrences of the same symbol inside different cells are represented as occurrences of tokens inside different places, each one of them identifying the presence of that symbol inside a certain cell. The consumption of objects from certain places and the production of new objects in other cells are then reflected in the movement of tokens between the respective places; pre- and post-multisets of the transitions in fact correspond to left-hand sides and right-hand sides of the rules in Π respectively.

As an example, we show in Figure 1 the PTL-net corresponding to the NC UNO of Example 4.

Next, similar to what was done in [13, 12], we introduce the notion of equivalence between the behavior of an NC and that of its corresponding PTL-net.

Definition 13 (m-equivalence). *Let* $\Pi = (V, w_1, w_2, \dots, w_n, R)$ *be a network of cells and let* $\mathcal{N}(\Pi)$ *be its corresponding PTL-net. For* $\mathrm{m} \in \{free, seq, max, lmax, min\}$, *we say that* Π *is* m-equivalent to $\mathcal{N}(\Pi)$, *and we write* $\Pi \equiv_\mathrm{m} \mathcal{N}(\Pi)$, *if, for every two configurations* $C = (w'_1, \dots, w'_n)$, $C' = (w''_1, \dots, w''_n)$ *of* Π, $C \Rightarrow_\mathrm{m} C'$ *iff* $h_{1,\Pi}(w'_1) \dots h_{i,n}(w'_n) \rightsquigarrow_\mathrm{m} h_{1,\Pi}(w''_1) \dots h_{i,n}(w''_n)$.

Thus, it is easy to see that the following proposition holds.

Proposition 5.1 *For any network of cells* Π, *for* $\mathrm{m} \in \{free, seq, max\}$, *we have that* $\Pi \equiv_\mathrm{m} \mathcal{N}(\Pi)$.

On the other hand, since all the transitions of the corresponding PTL-net are assigned to the same locality, we have that, for some network of cells Π, $\Pi \not\equiv_{lmax} \mathcal{N}(\Pi)$ and $\Pi \not\equiv_{min} \mathcal{N}(\Pi)$. However, we can think of choosing a different locality

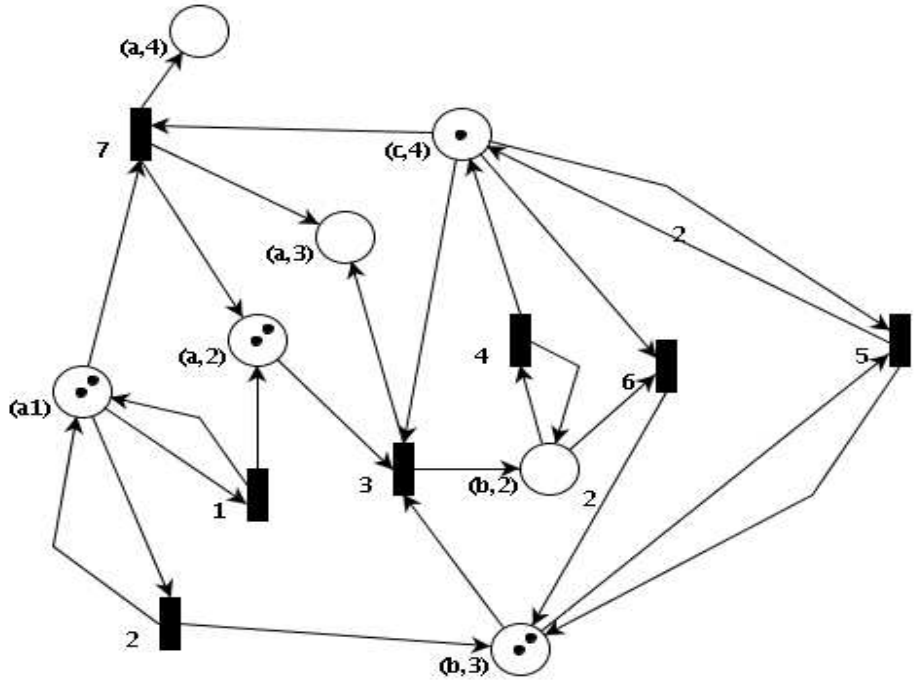


Fig. 1. PTL-net $\mathcal{N}(UNO)$ with its initial marking.

mapping for the construction of Definition 12 in order to establish the equivalence between the locally-maximal (or minimally-parallel) semantics of networks of cells and the locally-maximal (or minimally-parallel) execution of the corresponding PTL-net. To this aim, given an NC $\Pi = (V, w_1, w_2, \ldots, w_n, R)$, and a function $F : \{1, \ldots, |R|\} \to \mathbb{N}$, it is useful to consider the PTL-net $\mathcal{N}(\Pi, F)$ which is constructed as the one of Definition 12 except for the locality mapping which is replaced by F. Thus, we can ask whether, for any NC $\Pi = (V, w_1, w_2, \ldots, w_n, R)$, there exists a $F : \{1, \ldots, |R|\} \to \mathbb{N}$ such that $\Pi \equiv_{lmax} \mathcal{N}(\Pi, F)$ (or $\Pi \equiv_{min} \mathcal{N}(\Pi, F)$), or not.

Proposition 5.2 *There exists a network of cells Π with* 3 *rules such that, for all $F : \{1, 2, 3\} \to \mathbb{N}$, we have that $\Pi \not\equiv_{lmax} \mathcal{N}(\Pi, F)$.*

Proof. Let $DUE = (\{a, b\}, aa, bb, R)$ be an NC where R contains: rule $(a, 1) \to (aa, 1)$ labeled by 1, rule $(a, 1)(b, 2) \to (a, 2)(b, 1)$ labeled by 2, and rule $(b, 2) \to (bb, 2)$ labeled by 3. Then, let us suppose that $DUE \equiv_{lmax} \mathcal{N}(DUE, F)$ for some $F : \{1, 2, 3\} \to \mathbb{N}$.

By Definition 12, PTL-net $\mathcal{N}(DUE, F)$ is the PTL-net (P, T, W, M_0, L) with $P = \{(a, 1), (a, 2), (b, 1), (b, 2)\}$, $T = \{1, 2, 3\}$, $W((a, 1), 1) = 1$, $W(1, (a, 1)) = 2$,

$W((a,1),2) = 1$, $W((b,2),1) = 1$, $W(2,(b,1)) = 1$, $W(2,(a,2)) = 1$, $W((b,2),3) = 1$, $W(3,(b,2)) = 2$, $W(x,y) = 0$ in all other cases, and $M_0 = (a,1)(a,1)(b,2)(b,2)$.

Now, we observe that if $F(1) = F(3)$, then, for the initial configuration C_0 of DUE, $C_0 \Rightarrow_{lmax} (aaaa, bb)$, but $M_0 \not\rightsquigarrow_{lmax} (a,1)(a,1)(a,1)(a,1)(b,2)(b,2)$. This is because if transition 1 and transition 2 are assigned the same locality, then it is not possible to fire transition 1 with multiplicity 2 without firing in parallel transition 3 with multiplicity 2. This contradicts our hypothesis, hence it has to be $F(1) \neq F(3)$ irrespectively of the value of $F(2)$.

Next, if $F(1) \neq F(2)$, then $M_0 \rightsquigarrow_{lmax} (a,1)(b,1)(a,2)(b,2)(b,2)$ by selecting localities $F(2)$ and $F(3)$, but $C_0 \not\Rightarrow_{lmax} (ab, abb)$. This is because rule 2 involves cell 1, hence it is not possible to apply rule 2 in parallel with rule 3 without applying at the same time rule 1. Yet again, this contradicts our hypothesis, hence it has to be $F(1) = F(2)$. The same reasoning applies to the case $F(2) \neq F(3)$: $M_0 \rightsquigarrow_{lmax} (a,1)(a,1)(b,1)(a,2)(b,2)$, but $C_0 \not\Rightarrow_{lmax} (aab, ab)$. Therefore, it has to be $F(1) = F(2) = F(3)$ but this is in contrast with our earlier observation that it has to be $F(1) \neq F(3)$ in order to have $DUE \equiv_{lmax} \mathcal{N}(DUE, F)$.

Thus, we can conclude that there is no $F : \{1,2,3\} \to \mathbb{N}$ such that $DUE \equiv_{lmax} \mathcal{N}(DUE, F)$. □

Proposition 5.2 shows that, in the case of locally-maximal parallelism, it is not always possible for the corresponding PTL-net to mimic the behavior of the original network of cells. The intuitive reason for this is that interaction rules may involve several cells at the same time and locally-maximal parallelism for NC's is defined with respect to the cells involved rather than with respect to the rules applied; localities in PTL-nets are instead associated with transitions and determine which transitions fire in parallel within a step of executions irrespectively of the places involved. However, for an NC with interaction rules of input radius at most 1 (i.e., interaction rules that involve at most one cell in their left-hand side), it is easy to construct a corresponding PTL-net which exhibits an equivalent behavior even for locally-maximally parallelism. This is the case for the basic model of P systems as shown in [13, 12].

A similar result can be obtained for minimal parallelism.

Proposition 5.3 *There exists a network of cells Π with 3 rules such that, for all $F : \{1,2,3\} \to \mathbb{N}$, we have that $\Pi \not\equiv_{min} \mathcal{N}(\Pi, F)$.*

Proof. Let $DUE = (\{a,b\}, aa, bb, R)$ be the NC used in the proof of Proposition 5.2 with its initial configuration $C_0 = (aa, bb)$. Then, let us suppose that $DUE \equiv_m \mathcal{N}(DUE, F)$ for some $F : \{1,2,3\} \to \mathbb{N}$. PTL-net $\mathcal{N}(DUE, F)$ is the same as the one defined in the proof of Proposition 5.2.

Now, we observe that if $F(1) = F(2) = F(3)$, then, by firing only transition 1 with multiplicity 1, $M_0 \rightsquigarrow_{min} (a,1)(a,1)(a,1)(b,2)(b,2)$, but $C_0 \not\Rightarrow_{min} (aaa, bb)$. This is because rule 1 involves only cell 1, hence, since we are operating according to the minimal parallelism, it is not possible to apply rule 1 with multiplicity 1 without applying in parallel at the same time at least another rule involving cell 2. This contradicts our hypothesis, hence it cannot be $F(1) = F(2) = F(3)$.

Next, if $F(1) \neq F(2)$, then, by applying rule 2 with multiplicity 1, $C_0 \Rightarrow_{min} (ab, ab)$ but $M_0 \not\rightsquigarrow_{min} (a,1)(b,1)(a,2)(b,2)$. This is because, since we are operating according to minimal parallelism, it is not possible to fire transition 2 with multiplicity 1 without firing in parallel at the same time at least another transition belonging to locality $F(1) \neq F(2)$. The same reasoning apply to case $F(2) \neq F(3)$. Therefore, it has to be $F(1) = F(2)$ and $F(2) = F(3)$ but this is in contrast with our earlier observation that it cannot be $F(1) = F(2) = F(3)$ in order to have $DUE \equiv_{min} \mathcal{N}(DUE, F)$.

Thus, we can conclude that there is no $F : \{1,2,3\} \to \mathbb{N}$ such that $DUE \equiv_{min} \mathcal{N}(DUE, F)$. □

Proposition 5.3 shows that, even for minimal parallelism, it is not always possible for the corresponding PTL-net to mimic the behavior of the original network of cells. Proposition 5.2 and Proposition 5.3 are both based on the fact that an NC may contain both rules of input radius 1 with a local scope and rules of radius greater than 1 which in a sense belong to different cells at the same time.

We pass now to consider the reverse problem of finding for every PTL-net a corresponding network of cells with an equivalent behavior.

Definition 14. *Let $N = (P, T, W, M_0, L)$ be a PTL-net.*

1. *A* cell mapping for N *is any surjective function $C : P \to \{1, \ldots, n\}$ with $n \geq 1$.*
2. *For all $t \in T$, for any cell mapping C for N, the* pre-partition of t (w.r.t. C)*, denoted by $prep_C(t)$, is the string $(w_1, c_1)(w_2, c_2) \ldots (w_k, c_k)$ such that $pre_N(t) = w_1 w_2 \ldots w_k$, for all $1 \leq i \leq k$, $w_i \neq \lambda$ and $C(\aleph(w_i)) = \{c_i\}$, and, for all $1 \leq i, j \leq k$ with $i \neq j$, $C(\aleph(w_i)) \neq C(\aleph(w_j))$.*
3. *For all $t \in T$, for any cell mapping C for N, the* post-partition of t (w.r.t. C)*, denoted by $postp_C(t)$, is the string $(w_1, c_1)(w_2, c_2) \ldots (w_k, c_k)$ such that $post_N(t) = w_1 w_2 \ldots w_k$, for all $1 \leq i \leq k$, $w_i \neq \lambda$ and $C(\aleph(w_i)) = \{c_i\}$, and, for all $1 \leq i, j \leq k$ with $i \neq j$, $C(\aleph(w_i)) \neq C(\aleph(w_j))$.*
4. *For every marking M of N, for any cell mapping C for N, the* partition of M (w.r.t. C)*, denoted by $part_C(M)$, is the string $(w_1, c_1)(w_2, c_2) \ldots (w_k, c_k)$ such that $M = w_1 w_2 \ldots w_k$, for all $1 \leq i \leq k$, $w_i \neq \lambda$ and $C(\aleph(w_i)) = \{c_i\}$, and, for all $1 \leq i, j \leq k$ with $i \neq j$, $C(\aleph(w_i)) \neq C(\aleph(w_j))$.*
5. *For every marking M of N, for any cell mapping $C : P \to \{1, \ldots, n\}$ for N, the* extended partition of M (w.r.t. C)*, denoted by $exp_C(M)$, is the tuple $(w_1, w_2, \ldots, w_n)$ such that $M = w_1 w_2 \ldots w_n$ with $part_C(M) = (w_{c_1}, c_1)(w_{c_2}, c_2) \ldots (w_{c_k}, c_k)$ for some $\{c_1, c_2, \ldots, c_k\} \subseteq \{1, 2 \ldots, n\}$ and $w_i = \lambda$ for all $i \notin \{c_1, c_2, \ldots, c_k\}$.*

Thus, given a network of cells N, we use the cell mapping to associate places to certain cells and we use the concept of partition to distribute the places (i.e., their multiple occurrences) to these cells. Specifically, we give the following definition.

Definition 15 (corresponding NC). *Let $N = (P, T, W, M_0, L)$ be a PTL-net and let $C : P \to \{1, \ldots, n\}$ be a cell mapping for N. The* network of

cells corresponding to N (w.r.t. C) *is* $\mathcal{P}(N,C) = (P, w_1, w_2, \ldots, w_n, R)$ *where:* $exp_C(M_0) = (w_1, w_2, \ldots, w_n)$ *and*

$$R = \{(u_1,i_1)\ldots(u_k,i_k) \to (v_1,j_1)\ldots(v_h,j_h) \mid \\ t \in T, prep_C(t) = (u_1,i_1)\ldots(u_k,i_k), postp_C(t) = (v_1,j_1)\ldots(v_h,j_h)\}.$$

Thus, we can construct different NC's corresponding to the same PTL-net depending on the way we decide to assign places to cells; the unique constraint is that multiple occurrences of the same place have to remain confined within the same cell. At one end, a PTL-net can be translated into an NC with one cell where all the rules have input radius equal 1; every rule corresponds to a transition and these rules are no more than multiset rewriting rules. At the opposite end, a PTL-net can be translated into an NC with as many cells as its places; every rule corresponds to a transition and the its input radius is equal to the cardinality of the support of the pre-multiset of this transition. Notice that the locality mapping of a PT-net has no direct counterpart in the corresponding NC's.

Next, we introduce the notion of equivalence between a PTL-net and its corresponding NC's.

Definition 16 (m-equivalence). *Let* $N = (P, T, W, M_0, L)$ *be a PTL-net and let* $C : P \to \{1, \ldots, n\}$ *be a cell mapping for* N. *For* $\mathrm{m} \in \{free, seq, max, lmax, min\}$, *we say that* N *is* m-equivalent to $\mathcal{P}(N,C)$, *and we write* $N \approx_{\mathrm{m}} \mathcal{P}(N,C)$ *if, for every two markings* M_1, M_2, $M_1 \rightsquigarrow_{\mathrm{m}} M_2$ *iff* $exp_C(M_1) \Rightarrow_{\mathrm{m}} exp_C M_2$.

Thus, we can state the fundamental property which relates PTL-nets and corresponding NC's.

Proposition 5.4 *For any PTL-net* $N = (P, T, W, M_0, L)$, *for any cell mapping* C *for* N, *for* $\mathrm{m} \in \{free, seq, max\}$, *we have that* $N \approx_{\mathrm{m}} \mathcal{P}(N,C)$. *Moreover, if* $L(T) = \{l\}$ *for some* $l \geq 0$, *then, for any cell mapping* C *for* N, *for* $\mathrm{m} \in \{lmax, min\}$, *we have that* $N \approx_{\mathrm{m}} \mathcal{P}(N,C)$.

Therefore, the general equivalence between a PTL-net and its corresponding NC's can only be established for free-parallelism, sequential execution and maximal parallelism. For locally-maximal parallelism and minimal parallelism, the aforementioned equivalence can be established only for PTL-nets where all transitions are assigned to the same locality. In fact, the following results holds.

Proposition 5.5 *There exists a PTL-net* N *such that, for any cell mapping* C *for* N, *for* $\mathrm{m} \in \{lmax, min\}$, *we have that* $N \not\approx_{\mathrm{m}} \mathcal{P}(N,C)$.

Proof. Let $NONE$ be the PTL-net of Figure 2 where: transition R is assigned locality 1, transition S is assigned locality 2, and transition T is assigned locality 2; the initial marking of $NONE$ is $M_0 = aabb$. Then, let us suppose that there is a cell mapping C for $NONE$ such that $NONE \approx_{\mathrm{m}} \mathcal{P}(NONE, C)$.

Now, if $C(\{a,b\} = \{1\}$ (i.e., if $\mathcal{P}(NONE, C)$ contains only one cell), then it is obvious that $NONE \not\approx_{\mathrm{m}} \mathcal{P}(NONE, C)$. Therefore, it has to be $C(\{a,b\} = \{1,2\}$

(i.e., $\mathcal{P}(NONE, C)$ has to contain two cells) in order to be able to distinguish between locality 1 and 2.

Then, if $C(a) = 1$ and $C(b) = 2$, then, by Definition 15, $\mathcal{P}(NONE, C)$ contains rules: $(a,1) \to (aa,1)$, $(a,1)(b,2) \to (b,2)$, and $(b,2) \to (bb,2)$. Thus, $M_0 \rightsquigarrow_{lmax} aabb$ by firing transition R and transition S belonging to the same locality, but $exp_C(M_0) = (aa, bb) \not\Rightarrow_{lmax} (aa, bb)$. This is because rule $(a,1)(b,2) \to (b,2)$ involve cell 1 as well as cell 2, hence we cannot apply rule $(a,1) \to (aa,1)$ in parallel with rule $(a,1)(b,2) \to (b,2)$ without applying at the same time rule $(b,2) \to (bb,2)$. This contradicts our hypothesis. For symmetry, the same reasoning apply to the case $C(a) = 2$ and $C(b) = 1$.

Therefore, we can conclude that, for any cell mapping C for N, we have that $N \not\approx_{lma}$

For ıe cases as abov $\mathcal{N} \not\approx_{min}$ $\mathcal{P}(N, C$

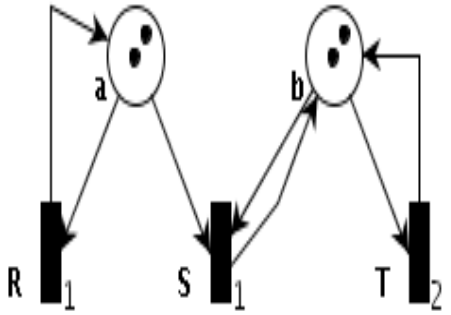


Fig. 2. PTL-net $NONE$ with its initial marking where transition R is assigned to locality 1, transition S is assigned to locality 2, and transition T is assigned to locality 2.

Finally, we stress once more the differences between cells and places. Cells of an NC are bags of objects that can contain multiple occurrences of different symbols, each one of them representing a different sort of objects; cells are seen as distinct components of a larger system whose behavior is given by their interactions; the adopted semantics determines which and how many cells become active from time to time. Places of a PTL-net store a number of tokens, each one of them representing a distinct occurrence of a specific place; places with their number of tokens represent resources that have to be acquired by transitions in order to fire, and, in PTL-nets, one usually abstracts from the actual location of these resources. Therefore, when translating a PTL-net into an NC, one has to make some extra assumptions about the assignment of places to cells, although, in general, one can

always see a PTL-net as an NC with one cell containing a finite set of multisets of rewriting rules.

6 Case-Studies

We present NC's solutions to two classical synchronization problems: producer/consumer and dining philosophers. The proposed models are derived from standard solutions based on Petri Nets.

6.1 Producer/Consumer

We consider the simpler version of the producer/consumer system from [20]: distinguished items are produced, delivered to a buffer, later removed from the buffer, and finally consumed. The buffer is assumed to have capacity for one item. Moreover, like in [20], we abstract from any concrete instance of the aforementioned operations and we focus only on the "interplay" between produce/deliver and remove/consume, and on the events necessary for their synchronization.

Specifically, we consider two sub-systems named *producer* and *consumer*, respectively, which "synchronize" (or "interact", or "communicate") through a shared buffer. The producer has two states: "ready to produce" and "ready to deliver". The consumer has two states: "ready to remove" and "ready to consume". The buffer has two states: "filled" and "empty". In state "ready to produce", the producer executes the operation "produce" and moves to state "ready to deliver"; in state "ready to produce", if the buffer is "empty", the producer executes the operation "deliver", which fills the buffer, and moves back to state "ready to produce". Similarly, in state "ready to remove", if the buffer is "empty", the consumer execute the operation "remove", which empties the buffer, and moves to state "ready to consume"; in state "ready to consume", the consumer executes the operation "consume" and moves back to state "ready to remove".

PTL-Net Representation

The PTL-net model of the aforementioned producer/consumer system, which is presented in [20], is reported in Figure 3, with its initial marking AEG, where:

$A \equiv$ "ready to produce",
$B \equiv$ "ready to deliver",
$F \equiv$ "filled",
$E \equiv$ "empty",
$G \equiv$ "ready to remove",
$H \equiv$ "ready to consume",
$p \equiv$ "produce",
$d \equiv$ "deliver",

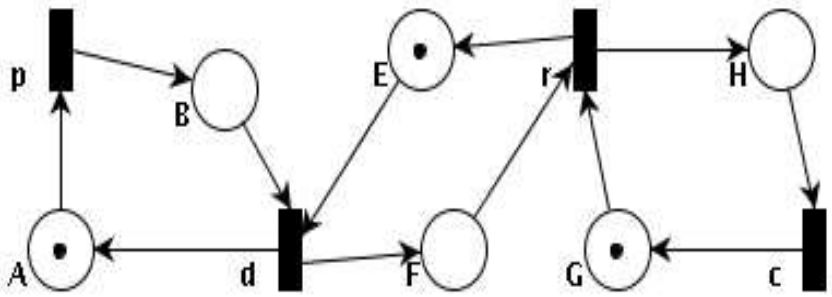


Fig. 3. PTL-net model of a producer/consumer system.

$r \equiv$ "remove",
$c \equiv$ "consume".

The unique transition enabled at the initial marking AEG is transition p ("produce"), and, when fired, it transfers a token from place A ("ready to produce") to place B ("ready to deliver"). Next, transition d ("deliver") is enabled and, when fired, it produces a token in place F (i.e., the buffer is filled) and a token in place A ("ready to produce"). Then, transition r ("remove") can fire which produces a token in place E (i.e., the buffer is emptied) and a token in place H ("ready to consume"). Finally, a token can be returned to place G ("ready to remove") by firing transition c ("consume").

We remark that:

- seq-execution and free-execution are non-deterministic; after transition d has fired, both transition p and transition r are enabled and, after transition r has fired, both transition d and transition c are enabled;
- max-execution is deterministic;
- if transitions p and d are assigned to a certain locality (i.e., the producer) and transition r and c to another one (i.e., the consumer), then lmax-execution is equivalent to free-execution;
- if transitions p and d are assigned to a certain locality (i.e., the producer) and transition r and c to another one (i.e., the consumer), then min-execution is equivalent to max-execution;
- irrespectively of the semantics, in each step, at least one transition is always enabled (*liveness*);
- irrespectively of the semantics, the consumer can consume only after having received an item from the buffer (i.e., only after having performed a "remove" operation);
- irrespectively of the semantics, after having produced an item, the producer has to deliver it into the buffer before returning to producer; the delivering can take place only when the buffer has been emptied.

NC Representation

As pointed out in Section 5, it is possible to find different NC's corresponding to the PTL-net of Figure 3. Here, in order to model the aforementioned producer/consumer system, we consider an NC PC with 3 cells. Cell 1 represents the producer, cell 2 represents the consumer, whereas cell 3 represents the buffer.

Cell 1 stores an object which specifies the states of the producer; this can be either A ("ready to produce") or B ("ready to deliver"). Similarly, cell 2 stores an object which specifies the state of the consumer; this can be either G ("ready to remove") or H ("ready to consume"). Cell 3 instead stores an object representing the state of the buffer, either F ("filled") or E ("empty"). The initial configuration is the tuple (A, G, E).

The desired behavior is then implemented by considering the following rules:

1. $(A, 1) \rightarrow (B, 1)$,
2. $(B, 1)(E, 2) \rightarrow (A, 1)(F, 3)$,
3. $(G, 2)(F, 3) \rightarrow (H, 2)(E, 3)$,
4. $(H, 2) \rightarrow (G, 2)$.

Yet again, we have that:

- the application of the rules is sequential and freely-parallel;
- the maximally-parallelism is deterministic;
- irrespectively of the semantics, in each step, at least one rule is always applicable;
- irrespectively of the semantics, the consumer can consume only after having received an item from the buffer;
- irrespectively of the semantics, after having produced an item, the producer has to deliver it into the buffer before returning to producer; the delivering can take place only when the buffer has been emptied.

Also, the PT-net $\mathcal{N}(PC)$ is the same as the PTL-net of Figure 3 except for the naming of places and transitions (i.e., they define two isomorphic graphs). In fact, rule 1 corresponds to transition p in the PT-net of Figure 3, rule 2 corresponds to transition d in the PT-net of Figure 3, rule 3 corresponds to transition r in the PT-net of Figure 3, and rule 4 corresponds to transition c in the PT-net of Figure 3. Moreover, for $F : \{1, 2, 3, 4\} \rightarrow \mathbb{N}$ such that $F(1) = F(2)$, $F(3) = F(4)$ and $F(1) \neq F(3)$, we have that $PC \equiv_{lmax} \mathcal{N}(PC, F)$ and $PC \equiv_{min} \mathcal{N}(PC, F)$. The vice versa is also true: for the PTL-net of Figure 3, if transitions p and d are assigned to a certain locality (i.e., the producer) and transition r and c to another one (i.e., the consumer), then, for m $\in \{free, seq, max, lmax, min\}$, PTL-net of Figure 3 is m-equivalent to PC. Therefore, for this version of the producer/consumer system, there is a sort of direct transcription of the PTL-net model into an NC model whose rules are able to represent exactly the same type of interactions between a producer and a consumer.

Remark 4. For the present version of producer/consumer, the fundamental properties are: there is always a transition enabled (*liveness*), the producer always alternates between "ready to consume" and "ready to deliver", and the consumer always alternates between "ready to remove" and "ready to consume". These properties are formally proved in [20] for the PTL-net of Figure 3 and these results can be directly transferred to our NC model; at some extent, this gives a flavor of the sort of properties which could be proved for NC's (or any other membrane systems) by using techniques developed in the area of Petri nets.

Next, we describe an evolution-communication model of the producer/consumer systems that is not a direct translation of the PTL-net solution from [20] but is inspired by the idea of P systems as systems where transformations involve only objects inside one specific cell and communication is responsible for moving objects from one cell to the other. In other words, we consider an NC with interaction rules of input radius 1 which does not rely on the simultaneous synchronization of two different cells.

Similarly to what was done before, we consider an NC $PC1$ with 3 cells: cell 1, the producer, cell 2, the consumer, and cell 3, the buffer. The initial configuration of $PC1$ is the tuple (A, G, E).

Cell 1 always moves from state A ("ready to produce") to B ("ready to deliver") by replacing in its inside object A with object B. In state B, after having received an object E from cell 3 (i.e., after having been notified that the buffer is empty), cell 3 produce in its inside an object A and object F; object F is then sent to cell 3 to fill the buffer. In state G ("ready to remove"), cell 2 always waits to receive an object F from cell 3 in order produce in its inside an object H (representing state "ready to consume") and an object E; object E is then sent to cell 3 to notify that the buffer is now empty. This behavior is implemented by means of the following rules:

1. $(A, 1) \to (B, 1)$,
2. $(E, 3) \to (E, 1)$,
3. $(BE, 2) \to (AF, 1)$,
4. $(F, 1) \to (F, 3)$,
5. $(F, 3) \to (F, 2)$,
6. $(GF, 2) \to (HE, 2)$,
7. $(H, 2) \to (G, 2)$,
8. $(E, 2) \to (E, 3)$.

Thus, all these rule have input radius equal to 1 and interactions between cell 1 (cell 2) are achieved through an effective exchange of objects. However, irrespectively of the semantics adopted, the aforementioned fundamental properties can still be observed: at any time, at least one rule is applicable, the producer always alternates between "ready to consume" and "ready to deliver", and the consumer always alternates between "ready to remove" and "ready to consume". Moreover, the producer can deliver only when the buffer is empty, and the consumer can

remove only when the buffer is filled. Also, notice that $PC1$ is somehow "more concurrent" than PC: there is always some communication which can be executed in parallel with the internal change of state. Nevertheless, maximal parallelism still leads to a deterministic behavior for $PC1$ and the movement of objects E and F does not affect the behavior of producer and consumer.

PTL-net $\mathcal{N}(PC1)$ is given in Figure 4.

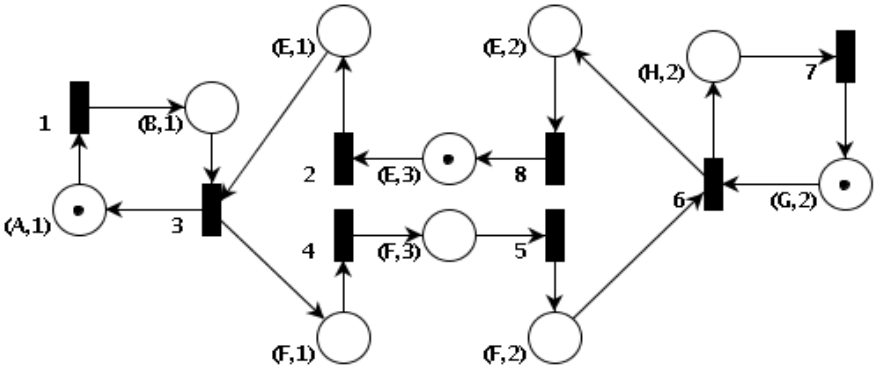


Fig. 4. PTL-net $\mathcal{N}(PC1)$ with its initial marking.

Remark 5. The solution based on the NC PC uses rules of input radius at most 2 and cooperation degree at most 1, whereas the solution based on the $PC1$ uses rules of input radius at most 1 and cooperation degree at most 2. In some sense, this shows a trade-off between the cooperation at level of cells and the cooperation at the level of objects.

Remark 6. If we do not consider the names of places and transitions, the PTL-net of Figure 4 is the same as the PTL-net of Figure 3 excepts for the presence of some "unary" transitions which correspond to the movement of objects E and F between cell 1, cell 2 and cell 3. Indeed, by using standard structural methods of Petri nets (e.g., see [11]), the aforementioned transitions could be removed so to have a PTL-net which is the same as the one of Figure 3 and inherits the same structural properties proved in [20]. Thus, the translation into PTL-net allows us to relate two different models of NC's in terms of structural properties which are somehow hidden in the two different types of rules used.

7 Dining Philosophers

We consider a distributed system where each resource is shared by two components, and each subsystem simultaneously requires two resources in order to start its

activities. The problem of modeling such a resource sharing scenario is classically formulated as the problem of the dining philosophers: five philosophers are sitting around a table, each philosopher has his own plate and can be eating or thinking (i.e., not eating). In order to eat, a philosopher needs two forks, but there are only two forks next to each plate so that no two neighboring philosophers may be eating simultaneously.

PTL-net representation

The five philosophers are denoted by A, B, C, D and E; the five forks are denoted by f_0, f_1, f_2, f_3 and f_4. For each $X \in \{A, B, C, D, E\}$, we denote by $l(X)$ its left fork and by $r(X)$ its right fork. Specifically, we set: $l(A) = f_0$, $r(A) = f_1$, $l(B) = f_1$, $r(B) = f_2$, $l(C) = f_2$, $r(C) = f_3$, $l(D) = f_3$, $r(D) = f_4$, $l(E) = f_4$ and $r(E) = f_0$ (i.e. A sits between E and B, B sits between A and C, C sits between B and D, D sits between C and E, and E sits between D and A).

Thus, a PTL-net FDP is constructed that contains: for each $X \in \{A, B, C, D, E\}$, a place X_t ("X_t is thinking") and a place X_e ("X_e is eating"), for each $0 \leq i \leq 4$, a place f_i ("f_i is available"), for each $X \in \{A, B, C, D, E\}$, a transition X_p ("X picks up forks") with $pre_{FDP}(X_p) = l(X)X_t r(X)$ and $post_{FDP}(X_p) = X_e$, and a transition X_r ("X returns forks") with $pre_{FDP}(X_r) = X_e$ and $post_{FDP}(X_r) = l(X)X_t r(X)$. The initial marking of FDP is $A_t B_t C_t D_t E_t f_0 f_1 f_2 f_3 f_4$ (i.e., all philosophers are thinking and all forks are available).

Figure 5 depicts the sub-net of an FDP modeling a philosopher. This classical PTL-net model of the dining philosophers is taken from [20].

At the initial marking $A_t B_t C_t D_t E_t f_0 f_1 f_2 f_3 f_4$, for $X \in \{A, B, C, D, E\}$, each transition X_p ("X picks up forks") is enabled by giving all philosophers a chance to start eating. However, irrespectively of the execution mode adopted, no two neighboring philosophers can start eating at the same time because they share one fork and, for all $0 \leq i \leq 4$, f_i contains only one token. Thus, in the first step, depending on the execution mode, a certain number of philosophers which are not neighbors start eating by firing at least one transition X_p, whereas the other philosophers keep thinking. Then, depending on the execution mode, some other philosophers may start eating, whereas the eating philosophers may release their forks and return thinking, and so on.

The fundamental property of the PTL-net FDP is that it avoids the deadlocks: FDP never produces a marking which no transition is enabled at. Moreover, from time to time, FDP offers every philosopher the chance to start eating (i.e., it avoids scenarios where no philosopher can ever start eating), although fairness is not guaranteed (i.e., there may be some philosophers which always alternate between thinking and eating with the other philosophers thinking indefinitely). Yet again, this structural properties are formally proven in [20].

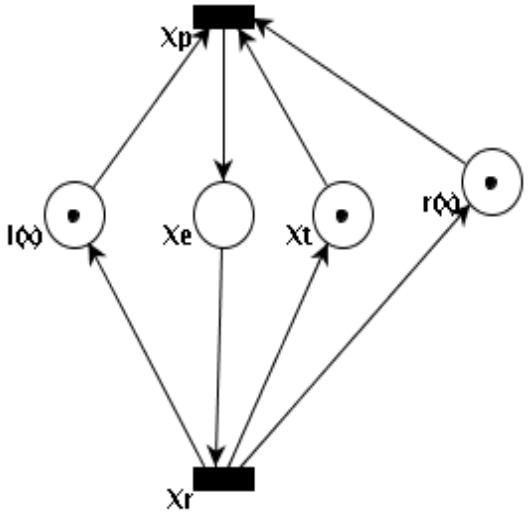


Fig. 5. PTL-net representation of a dining philosopher with its initial marking.

NC Representation

By starting from the PTL-net FDP, we construct an NC PH with 6 cells: cell 1 represents philosopher A, cell 2 represents philosopher B, cell 3 represents philosopher C, cell 4 represents philosopher D, cell 5 represents philosopher E, and cell 6 representing the table. If we denote by $ph(i)$ the philosopher represented by cell i, for all $1 \leq i \leq 5$, then cell i always contain either an object $ph(i)_t$ ("$ph(i)$ is thinking") or an object $ph(i)_e$ ("$ph(i)$ is eating"). Also, as in the previous PTL-net representation, for each $X \in \{A, B, C, D, E\}$, we denote by $l(X)$ its left fork and by $r(X)$ its right fork. Cell 6, the table, always contain the currently available forks: the availability of a fork f_i, for all $0 \leq i \leq 4$, is represented as occurrence of an object f_i inside cell 6. The initial configuration of PH is the tuple $(A_t, B_t, C_t, D_t, E_t, f_0 f_1 f_2 f_3 f_4)$.

The behavior of the five dining philosophers is then implemented by means of a set of rules which, for all $1 \leq i \leq 5$, contains the rules:

1. $(ph(i)_t, i)(l(ph(i))\, r(ph(i)), 6) \rightarrow (ph(i)_e, i)$,
2. $(ph(i)_e, i) \rightarrow (ph(i)_t, i)(l(ph(i))\, r(ph(i)), 6)$.

Specifically, for all $1 \leq i \leq 5$, rule 1 models transition $ph(i)_p$ ("$ph(i)$ picks up forks") of the PTL-net FDP, whereas rule 2 models transition $ph(i)_r$ ("$ph(i)$ returns forks) of the PTL-net FDP. In fact, the NC PH is the PTL-net $\mathcal{N}(FDP, C)$ with $C(A_t) = C(A_e) = 1$, $C(B_t) = C(B_e) = 2$, $C(C_t) = C(C_e) = 3$,

$C(D_t) = C(D_e) = 4$, $C(E_t) = C(E_e) = 6$, and $C(f_i) = 6$ for all $0 \leq i \leq 4$. Therefore, even for the NC PH, we can say that deadlock is avoided.

Another NC representation of the five dining philosophers can be obtained by distributing the forks to five different cells. Specifically this means considering an NC $PH1$ obtained from PH by removing cell 6 and adding a cell $6+i$, for all $0 \leq i \leq 4$; each cell $6+i$ contains an occurrence of object f_i whenever fork f_i is available. The initial configuration of $PH1$ is the tuple $(A_t, B_t, C_t, D_t, E_t, f_0, f_1, f_2, f_3, f_4,)$.

Then, for all $1 \leq i \leq n$, the new set of rules contains the rules:

1. $(ph(i)_t, i)(l(ph(i)), c(l(ph(i))))(r(ph(i)), c(r(ph(i)))) \rightarrow (ph(i)_e, i)$,
2. $(ph(i)_e, i) \rightarrow (l(ph(i)), c(l(ph(i))))(r(ph(i)), c(r(ph(i))))$.

where, for all $1 \leq i \leq 5$, $c(l(ph(i)))$ and $c(r(ph(i)))$ denote the respective locations of these two forks.

The NC $PH1$ contains rules of input radius at most 3 and cooperation degree at most 1, and it corresponds to the PTL-net $\mathcal{N}(FDP, C)$ with $C(A_t) = C(A_e) = 1$, $C(B_t) = C(B_e) = 2$, $C(C_t) = C(C_e) = 3$, $C(D_t) = C(D_e) = 4$, $C(E_t) = C(E_e) = 6$, and $C(f_i) = 6 + i$ for all $0 \leq i \leq 4$. Therefore, with respect to their PTL-net representation, the NC PH and NC $PH1$ cannot be distinguished.

8 Discussion

Networks of cell (NC's, for short) are a general class of P systems which encompass the essential features of evolution/communication of both P systems and tissue P systems. Rules in NC's allow different cells to synchronize in order to consume some multisets from their inside and produce some new multisets inside some other cells. As we have seen, this means that we can express within the framework of NC's both forms of coupled transports, like boundary rules and standard evolution rules with communication controlled by targets $here, in, out$. However, the structure of an NC does not necessarily corresponds to a graph or a tree; NC's of cells abstract from the underlying structure by focusing only on the interactions which can take place between the cells present in the system. In fact, these cells can be equally thought as being physically connected in some way which makes possible for the interactions to take place, or as randomly bumping into each other and interacting whenever it is possible. In a sense, such an approach is closer to the idea of a Petri net as a collection of transitions which can fire when certain resources become available, with some of these transitions competing for the same set of resources. The difference is that in NC's resources are objects which are specifically placed inside certain cells. This has led us to two important observations:

- Despite being able to translate every NC into a PTL-net by using a construction analogous to the one used in [7, 13, 12], in the case of locally-maximal parallelism and minimal parallelism, it is not always possible for the corresponding PTL-net to mimic the behavior of the original network of cells; locally-maximal

parallelism for PTL-nets is defined with respect to the localities which are assigned to the transitions; these localities then determine which transitions can possibly fire in parallel at the same time irrespectively of the places involved; rules of NC's involving more than one cells are instead not assigned a priori to any cell, and locally-maximally parallelism and minimal parallelism are defined with respect to a cell that can get involved in some interactions from time to time depending on a particular distribution of objects;

- For a given PTL-net, it is possible to find different corresponding NC's depending on the way places are assigned to the cells; the only restriction is that multiple occurrences of the same place has to remain confined within the same cell; for instance, for the five dining philosophers, it has been possible to produce two NC-based models: one with 6 cells and rules of input radius 2, and another one with 10 cells and rules of input radius 3.

In other words, in P systems, one naturally reasons about components (e.g., producer, consumer, buffer) and these are usually seen as being separate cells (or membranes). Also, one naturally distinguishes between operations affecting the inner state of a membrane and the operations involving interactions between different membranes. Moreover, in membrane systems, the inner state of a membrane can be given by an arbitrarily large multiset; this allows us to combine cooperation at the level of objects with interaction between different cells. Petri nets, with their graphical notation, are centered around the idea of resources which have to be acquired (tokens inside places) before certain actions can be taken; this facilitates the reasoning about properties like causality (the execution of certain actions depends on the execution of some others), concurrency (certain group of action can always be executed in parallel), and conflicts (certain actions compete with some others for the usage of certain resources); in membrane systems, these structural properties instead remains somehow hidden in the formalism used for representing the rules.

The present research can be continued in several directions. For instance, from a computational point of view, although we know that NC's with a maximally-parallel semantics are computationally complete, the computational power of NC's of cells with other semantics deserves further investigations especially for what concerns the size and types of rules used. Moreover, as pointed out in [24], interaction rules of NC's can express forms of cooperative communication other than symport/antiport or conditional uniport, and the computational power of these forms of communication is not yet fully understood. Then, for what concerns the relationships between P systems and Petri nets, as observed in [12], other features of P systems (without being limited to NC's) may or may not be representable in Petri nets. However, for some classes of P systems, their Petri-net representation offers the possibility of analyzing their behavior with respect to certain structural properties which can be formally proved for Petri nets and which are thoroughly managed through a plethora of tools [2]. Finally, we remark that, although one may see Petri nets as a low-level implementation of P systems, there are classes of high-level Petri Nets (e.g., colored Petri nets [11], objects nets [11], system nets

[20]) with features usually not considered for P systems, such as: types, variables, and predicates. The need for these features in P systems may be checked for specific modeling purposes against appropriate case-studies.

Acknowledgements

The research of Francesco Bernardini is supported by NWO, Organisation for Scientific Research of The Netherlands, project 635.100.006 "VIEWS".

References

1. The P systems web page. `http://psystems.disco.unimib.it`.
2. Petri nets world - Petri nets tools database. `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html`.
3. F. Bernardini and M. Gheorghe. Cell Communication in Tissue P Systems: Universality Results. *Soft Computing*, 9, 9 (2005), 640–649.
4. F. Bernardini and V. Manca. P Systems with Boundary Rules. In Gh. Păun, G. Rozenberg, A. Salomaa, and C. Zandron, editors, *Membrane Computing. International Workshop, WMC-CdeA 02, Curtea de Argeş, Romania, August 19-23, 2002. Revised Papers*. Volume 2597 of *Lecture Notes in Computer Science*, Springer, 2003, 107–118.
5. F. Bernardini, F.J. Romero-Campero, M. Gheorghe, and M.J. Pérez-Jiménez. A Modelling Approach Based on P Systems with Bounded Parallelism. In H.J. Hoogeboom, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, Seventh International Workshop, WMC7, Leiden, The Netherlands, July 2006*. Volume 4361 of *Lecture Notes in Computer Science*, Springer, 2007, 49–65.
6. G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez. P systems with minimal parallelism. *Theoretical Computer Sci.*, to appear.
7. S. Dal Zilio and E. Formenti. On the Dynamics of PB Systems: A Petri Net View. In C. Martín-Vide, G. Mauri, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, International Workshop, WMC 2003, Tarragona, Spain, July, 17-22, 2003, Revised Papers*. Volume 2933 of *Lecture Notes in Computer Science*, Springer, 2004, 153–167.
8. J. Desel, W. Reisig, and G. Rozenberg, editors. *Lectures on Petri Nets and Concurrency*. Volume 3098 of *Lecture Notes in Computer Science*. Springer, 2004.
9. R. Freund. Asynchronous P Systems and P Systems Working in the Sequential Mode. In G. Mauri, Gh. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing. International Workshop, WMC 2004, Milan, Italy, June 2004. Revised and Invited Papers*. Volume 3365 of *Lecture Notes in Computer Science*, Springer, 2005, 36–62.
10. R. Freund, L. Kari, M. Oswald, and P. Sosík. Computationally Universal P Systems without Priorities: Two Catalysts are Sufficient. *Theoretical Computer Science*, 330 2 (2005), 251–266.
11. C. Girault and R. Valk. *Petri Nets for Systems Engineering*. Springer, 2003.
12. J. Kleijn and M. Koutny. Synchrony and Asynchrony in Membrane Systems. In H.J. Hoogeboom, Gh. Paun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, Seventh International Workshop, WMC7, Leiden, The Netherlands, July 2006*. Volume 4361 of *Lecture Notes in Computer Science*, Springer, 2007, 66–85.

13. J. Kleijn, M. Koutny, and G. Rozenberg. Towards a Petri Net Semantics for Membrane Systems. In R. Freund, Gh. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing: 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised, Selected and Invited Papers*. Volume 3850 of *Lecture Notes in Computer Science*, Springer, 2006, 292–309.
14. S.N. Krishna and A. Păun. Results on Catalytic and Evolution-Communication P Systems. *New Generation Computing*, 22, 4 (2004), 377–394.
15. C. Martin-Vide, Gh. Păun, J. Pazoz, and A. Rodriguez-Paton. Tissue P Systems. *Theoretical Computer Sci.*, 296, 2 (2003), 295–326
16. C. Martín-Vide, Gh. Păun, and G. Rozenberg. Membrane Systems with Carriers. *Theoretical Computer Science*, 270, 1-2 (2002), 779–796.
17. Gh. Păun. Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
18. Gh. Păun. *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
19. Gh. Păun, Y. Sakakibara, and T. Yokomori. Membrane Systems on Graphs of Restricted Forms. *Publicationes Mathematicae Debrecen*, 60 (2002), 635–660.
20. W. Reisig. *Elements of Distributed Algorithms. Modelling and Analysis with Petri Nets*. Springer, 1998.
21. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*. Volume 1491 of *Lecture Notes in Computer Science*. Springer, 1998.
22. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets II: Applications*. Volume 1492 of *Lecture Notes in Computer Science*. Springer, 1998.
23. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages*, volume 1–3. Springer, 1997.
24. S. Verlan, F. Bernardini, M. Gheorghe, and M. Margenstern. Generalized Communicating P Systems. Submitted, 2007.
25. S. Verlan, F. Bernardini, M. Gheorghe, and M. Margenstern. Computational Completeness of Tissue P Systems with Conditional Uniport. In H.J. Hoogeboom, Gh. Păun, G. Rozenberg, and A. Salomaa, editors. *Membrane Computing, Seventh International Workshop, WMC7, Leiden, The Netherlands, July 2006*. Volume 4361 of *Lecture Notes in Computer Science*, Springer, 2007, 521–535.

Extended Spiking Neural P systems with Excitatory and Inhibitory Astrocytes

Aneta Binder[1], Rudolf Freund[1], Marion Oswald[1], Lorenz Vock[2]

[1] Vienna University of Technology
Faculty of Informatics
Favoritenstr. 9, 1040 Vienna, Austria
{ani,rudi,marion}@emcc.at
[2] Medical University of Vienna
Währinger Gürtel 18-20, 1090 Vienna, Austria
lorenz.vock@meduniwien.ac.at

Summary. We investigate an extended model of spiking neural P systems incorporating astrocytes and their excitatory or inhibitory influence on axons between neurons. Using very restricted variants of *extended spiking neural P systems with excitatory and inhibitory astrocytes* we can easily model Boolean gates like NAND-gates as well as discrete amplifiers.

1 Introduction

In this paper we integrate several models describing the functioning of the human brain based on the biological background. New models in the area of neural computation were introduced based on the observation that neurons send electrical impulses (also called *spikes*) along axons to other neurons, e.g., see [4], [11], [12]. P systems (membrane systems) were introduced as a formal model describing the hierarchical structure of membranes in living organisms and the biological processes in and between cells (an introduction to this field can be found in [17], for the actual state of the art in this area we refer the reader to [23]).

Combining the ideas of P systems and spiking neurons, a new variant of so-called tissue P systems (see [13]) called spiking neural P systems was investigated, e.g., see [8], [18]. An extended version of spiking neural P systems allowing to send different informations along the axons between two neurons was investigated in [1]. In spiking neural P systems (see [8]), the contents of a neuron consists of a number of so-called spikes. The rules assigned to a cell allow us to send information to other neurons in the form of electrical impulses – spikes – which are summed up at the target cell; the application of the rules depends on the contents of the neuron. In [1], an extended version of this original model of spiking neural P systems was

introduced based on some other observations from biology; for example, the spikes coming along different axons may cause effects of different magnitude. In [3], the role of inhibitory axons in extended spiking neural P systems was investigated (the arrival of spikes in the neuron affected by spikes along an inhibitory axon is inhibited).

Until recently, *astrocytes*, a sub-type of macroglia have been understood as star-shaped glial cells spanning around neurons in the central nervous system (CNS). Their main function was suspected to be the metabolic support of the neurons with glucose and nutrients as well as metabolic support for endothelial cells for keeping the blood barrier. They also were found to play a critical role in the neuronal survival and differentiation or neurite outgrowth. For more details see [19]. More recent biochemical literature, however, has put forward the idea that astrocytes have an important role in the plasticity of the CNS namely the synaptogenesis [7]. Astrocytes were also found to influence the concentration of neuroactive substances [14] and may serve as intermediaries in neuronal regulation of blood flow [16]. It also has become apparent that astrocytes themselves form an information-transmitting network by passing elevations of Calcium (Ca2+) [5][6]. It has been found that Ca2+ elevations in astrocytes modulate neuronal excitability and synaptic transmission. On the other hand, astrocytes are shown to be influenced by neurotransmitters [20] that might influence Ca2+ concentrations indicating that astrocytes might discriminate between different levels of neuronal activity [19]. It is also suggested that astrocytes may respond to synaptic activity in local domains [15] only and that these local domains may also discriminate between neurotransmitters (see [19]). Hence, a complex feedback loop of neuronal modulation exerted by astrocytes can be postulated.

The influence of *astrocytes* in the functioning of the human brain has also been investigated in [21], where to the interaction between the networks of neurons and astrocytes in addition the influence of the capillary system in connection with the networks of neurons and astrocytes was modelled. Based on the biological background, but without claiming to model it in a decent way, we develop a model of membrane systems incorporating some specific features of complex systems consisting of two interacting networks of neurons and astrocytes. For the signals sent from one neuron to another one, we base our model of *extended spiking neural P systems with excitatory and inhibitory astrocytes* on the ideas of (extended) spiking neural P systems and add the concept of astrocytes influencing the signals along the axons. For the astrocytes themselves, we assume their membrane potential to be changed according to external inputs which may either come from neural cells or the firing intensity and frequency along the axon. We shall assume two thresholds; then in the excitatory case, the effect of the astrocyte on the axon it controls is as follows:

If the membrane potential is below the first threshold, then there is no effect of the membrane potential of the astrocyte on the controlled axon. If the membrane potential is between the lower and the upper threshold, then the signals along the controlled axon are affected in an excitatory (amplifying) way. Yet if the membrane

potential goes beyond the upper threshold the effect turns to be an inhibitory one, decreasing the weight of the signals coming along the axon. In this way, the astrocyte network acts as a complex control mechanism on the network of the neural cells and is itself regulated by this network of neural cells. Hence, these networks of neurons and astrocytes form a complex system. We do not model the influence of the capillary system which is described as a third part of such a complex interacting system of networks as described in [21].

Our new model could be used for the representation of artificial neural networks, especially for self-organizing feature maps; yet in contrast to analytic models of such variants of neural networks, our model works in a discrete manner, but on the other hand, is based on a graph-like structure and not on a (usually two-dimensional) grid. An example of such a two-dimensional artificial neural network based on biological observations of the complex networks of neurons and astrocytes in the human neocortex can be found in [2]. Moreover, our model of neurons and axons with the axons being influenced by astrocytes in an excitatory or inhibitory way and the neurons influencing the astrocytes, is also related with the specific model of Petri nets with range arcs as described in [9] where process arcs influence transitions in an activating or inhibitory way and this influence depends on the number of tokens in a place which makes enabling a given transition possible.

In this paper we do not focus on applications as the possibility for modelling artificial neural networks as self-organizing feature maps (e.g., see [10]) for specific application tasks. Instead we show the potentials of our model to formalize discrete functions, e.g., networks of logical gates. Moreover, we exhibit the computational completeness of our model.

2 Extended Spiking Neural P Systems with Excitatory and Inhibitory Astrocytes

For the basic elements of formal language theory needed in the following, we refer to any monograph in this area, in particular, to [22]. We just list a few notions and notations: 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. $\mathbb{N}_+$ denotes the set of positive integers (natural numbers), $\mathbb{N}$ is the set of non-negative integers, i.e., $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$. The interval of non-negative integers between k and m is denoted by $[k..m]$. By $REG(\mathbb{N})$ and $RE(M)$ we denote the sets of subsets of $\mathbb{N}$ that are regular and recursively enumerable, respectively.

The basic elements of membrane computing are taken from [17]; comprehensive information can be found on the P systems web page `http://psystems.disco.unimib.it`. Moreover, for the motivation and the biological background of spiking neural P systems we refer the reader to [8].

An *extended spiking neural P system with excitatory and inhibitory astrocytes* (of degree $m \geq 1$) (in the following we shall simply speak of an *ESNPA system*) is a construct

$$\Pi = (m, n, S, R, U)$$

where

- m is the number of *neurons*; the neurons are uniquely identified by a number between 1 and m (obviously, we could instead use an alphabet with m symbols to identify the neurons);
- n is the number of *astrocytes*; the astrocytes are uniquely identified by a number between $m+1$ and $m+n$;
- S describes the *initial configuration* by assigning an initial value (of spikes) to each neuron as well as an initial value (membrane potential) to each astrocyte;
- R is a finite set of *rules* of the form $\left(i, E/a^k \to P\right)$ such that $i \in [1..m]$ (specifying that this rule is assigned to cell i), $E \subseteq REG(\mathbb{N})$ is the *checking set* (the current number of spikes in the neuron has to be from E if this rule shall be executed), $k \in \mathbb{N}$ is the "number of spikes" (the energy) consumed by this rule, and P is a (possibly empty) set of *productions* of the form (l, w) where $l \in [1..m+n]$ (thus specifying the target neuron or astrocyte), $w \in \mathbb{N}$ is the *weight* of the energy sent along the axon from neuron i to neuron or astrocyte l;
- U is a finite set of *rules* of the form $(r, p, q, h, h', f, f', f'')$ such that $r \in [m+1..n]$ and $p, q \in [1..m]$ (specifying that this rule is assigned to astrocyte r and influencing the axon between the neurons p and q), $h, h' \in \mathbb{N}$, $h \leq h'$ are thresholds, and f, f', f'' are functions $\mathbb{N} \to \mathbb{N}$ changing the energy w, sent along the axon from p to q, to w' as follows: if $w < h$, then $w' = f(w)$, if $h \leq w \leq h'$, then $w' = f'(w)$, if $w > h'$, then $w' = f''(w)$.

A *configuration* of the ESNPA system is described as follows:

- for each neuron, the actual number of spikes in the neuron is specified;
- for each astrocyte, the actual membrane potential of the astrocyte is specified.

A *transition* from one configuration to another one now works as follows:

- for each neuron i, we first check whether we can "activate a rule" $\left(i, E/a^k \to P\right)$, i.e., if the current value of spikes in neuron i is in E; waiting to be executed; then neuron i "spikes", i.e., for every production (l, w) occurring in the set P we put the corresponding package (l, w) on the axon from neuron i to neuron l or astrocyte l, respectively;
- if there is a rule $(r, i, l, h, h', f, f', f'') \in U$, the energy w in a package (l, w) on the axon from neuron i to neuron l is modified according to this rule to (l, w') as described above;
- for each neuron l, we now consider all eventually modified packages (l, w') on axons leading to neuron l; we then sum up all weights w' in such packages and add this sum to the corresponding number of spikes in neuron l;
- for each astrocyte l, we now consider all packages (l, w) on axons leading to astrocyte l; we then sum up all weights w in such packages and take this sum

as the new membrane potential for astrocyte l (i.e., we forget the previous potential).

After having executed all the substeps described above in the correct sequence, we obtain the description of the new configuration. A *computation* is a sequence of configurations starting with the initial configuration given by S.

An ESNPA system can be used to generate sets of numbers from $RE(\mathbb{N})$ as follows: A computation is called *successful* if it halts, i.e., if for no neuron, a rule can be activated. We then consider the contents, i.e., the number of spikes, of a specific neuron called *output neuron* in halting computations. According to [8], we can also take the distance between the first two spikes in an output neuron to define the number it computes. For generating k-dimensional vectors of non-negative integers, we have to designate k neurons as *output neurons*.

In the following, we shall use ESNPA systems to compute discrete functions, especially we shall exhibit how Boolean functions can be computed by using NAND-gates. When computing functions, we assume external input signals arriving in some designated *input neurons* as well as several *output neurons* for sending out the computed function with a spike indicating the signal 1 and with no spike being sent out indicating the signal 0.

The rules $\left(i, E/a^k \to P\right)$ in the examples given in the succeeding section will be of a very special form, i.e., we always have $E = \left\{a^k\right\}$, hence, we can omit E. Moreover, the productions (l, w) in P have the same weights for all l occurring in P, and even the sets P are the same for all rules $\left(i, E/a^k \to P\right)$ for each i; hence, we can indicate such rules as in Figure 1 where the rule $a^k \to a^l$ in neuron p means that k spikes are consumed in neuron p and l spikes are sent to every neuron q if there exists an axon from p to q. Moreover, a^m in neuron p indicates the initial value of m spikes in this neuron.

Fig. 1. Representation of simple rules in neurons.

The specific effect of very special astrocytes is depicted in Figures 2 and 3: in Figure 2, the influence of an *excitatory astrocyte* r on an axon between two neurons p and q is depicted: $\geq k|f$ in astrocyte r means that if $x \geq k$ spikes are sent out from neuron p then $f(x)$ spikes will reach neuron q, whereas for a number of spikes $x < k$ no spike will reach q. On the other hand, in Figure 2, the influence of an *inhibitory astrocyte* r on an axon between two neurons p and q is depicted: $\leq k|f$ in astrocyte r means that if $x \leq k$ spikes are sent out from neuron p then

$f(x)$ spikes will reach neuron q, whereas for a number of spikes $x > k$ no spike will reach q.

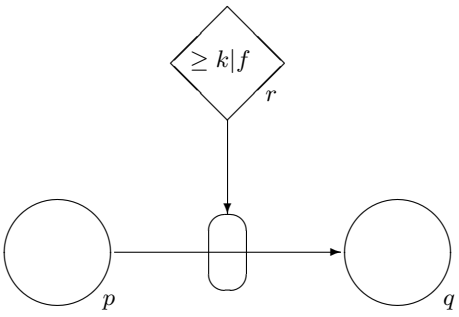


Fig. 2. Excitatory astrocyte.

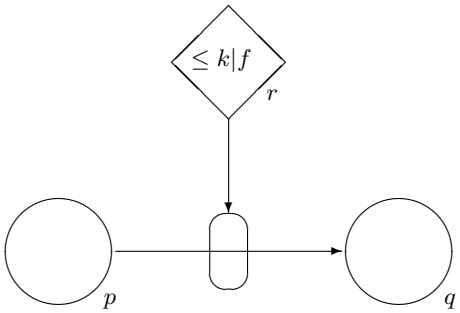


Fig. 3. Inhibitory astrocyte.

3 Computing with ESNPA Systems

In this section we first exhibit that ESNPA systems working as generators are computationally complete, i.e., able to generate any recursively enumerable set of non-negative integers. Then we show how networks of logical gates can be simulated by using specific ESNPA systems; in fact we describe an ESNPA system representing a NAND-gate. Finally we describe an ESNPA system representing a discrete amplifier.

3.1 Computational Completeness

As already the original model of spiking neural P systems was shown to be computationally complete, i.e., able to generate any recursively enumerable set of non-negative integers, with only those features also allowed in the sub-network of neurons in ESNPA systems, we immediately obtain computational completeness for ESNPA systems, too, because just omitting astrocytes gives a sufficiently powerful submodel of spiking neural P system as defined in [8]. Moreover, the model of ESNPA systems as defined above with just omitting the astrocytes is a submodel of extended spiking neural P systems as defined in [1] that again is sufficiently powerful to cover the proofs given there for computational completeness. The additional use of astrocytes would allow for different constructions with the rules for the neurons being even more restricted than in the case of spiking neural P systems, yet we do not go into the technical details of such constructions in this paper.

3.2 Networks of Logical Gates

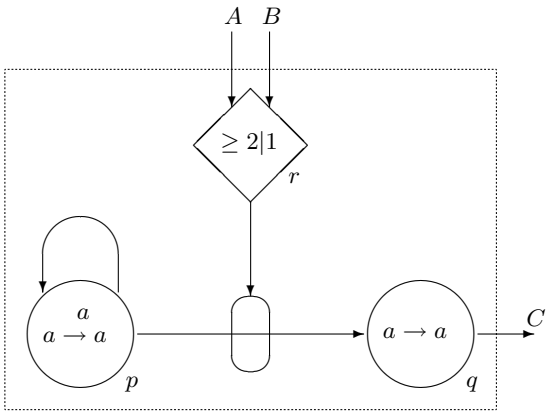


Fig. 4. AND-gate.

As is well known, any Boolean function can be obtained by networks only consisting of NAND-gates (and units representing the identity function). The identity function obviously can be obtained by using the same inputs (i.e., $A = B$) in an AND-gate as depicted in Figure 4.

The AND-gate is shown in Figure 4: A, B are the inputs, C is the output; the neuron p is a source sending out one spike in each time step which only reaches neuron q if the axon is *excited* by the astrocyte which reaches the excitatory threshold 2 if and only if both inputs A and B are 1. The notion $\geq 2|1$ in astrocyte r means that only if the sum of input spikes (A and B) is ≥ 2, then one (1) spike

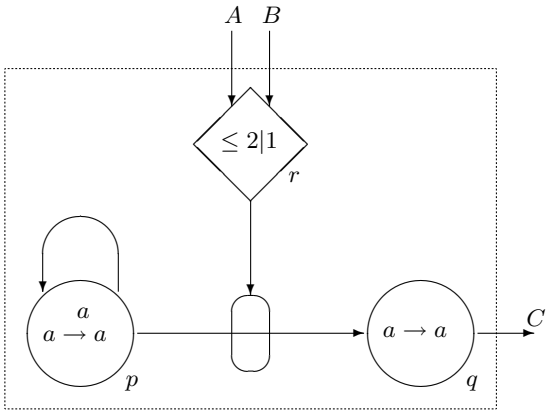


Fig. 5. NAND-gate.

is sent to neuron q, whereas if less than two input spikes arrive in astrocyte r, then no spike will reach the output neuron q. If both inputs (A and B) represent the same signal, i.e., if $A = B$, then neuron q will get a spike if and only if $A = 1$.

The NAND-gate is shown in Figure 5: again A, B are the inputs, C is the output; the neuron p is a source sending out one spike in each time step which only reaches neuron q if the axon is not inhibited by the astrocyte which reaches the inhibitory threshold 2 if and only if both inputs A and B are 1.

Any network of NAND-gates and AND-gates (only needed as identities keeping a signal as it is for one time step) of depth n yields the result of the computation with a delay of n, i.e., given the input at time t, the corresponding output appears at time $t + n$.

3.3 A Discrete Amplifier

The ESNPA system depicted in Figure 6 represents a discrete amplifier which, as soon as the input from B goes beyond the given threshold k, from the input x given at E computes the function $f(x) = nx$ at C. We have to remark that the rules $a^l \to a^l$ given in the neurons p and q represent the (theoretically infinite) set of rules $\left\{\{a\}^* / a^l \to a^l \mid l \in \mathbb{N}\right\}$ (for practical applications, an upper bound can be assumed).

4 Conclusion

The model we discussed in this paper is already very powerful from a theoretical point of view as elaborated in the preceding section. On the other hand, for specific applications, especially in the area of artificial neural networks and self-organizing feature maps, an extended version where we allow the dynamic evolution of new

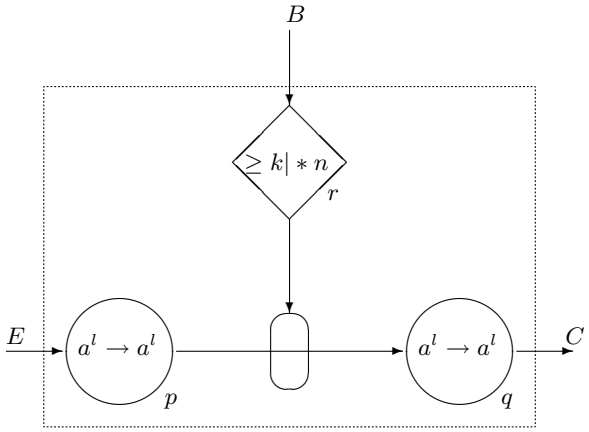


Fig. 6. An ESNPA amplifier.

connections between neurons, could be useful; the influence of the already existing astrocytes on these new axons plays an important role.

Another variant to be considered in the future are networks where part of the network may be destroyed which also has an interesting biological background. In this case, the ability of such a complex network to reorganize itself is the most challenging aspect of this variant.

Other variants may allow one astrocyte to influence more than one axon, eventually even in a different way and on the other hand, one axon may be influenced by several astrocytes, again eventually in a different way (in an inhibitory or excitatory way). For example, in this way more complex functions can be described by a single "unit" (circuit).

References

1. A. Alhazov, R. Freund, M. Oswald and M. Slavkovik, Extended Spiking Neural P Systems Generating Strings and Vectors of Non-Negative Integers, in: H. J. Hoogeboom, Gh. Paun, G. Rozenberg (eds), Pre-proceedings of the 7th Workshop on Membrane Computing WMC7, 2006, pp. 88-101.
2. F. Buarque de Lima Neto and P. de Wilde, Venn-like models of neo-cortex patches, *International Joint Conference on Neural Networks*, 2006, pp.89–96.
3. R. Freund and M. Oswald, Extended spiking neural P systems with inhibitory axons, in: M. Sugisaka, H. Tanaka, Proc. *International Symposium on Artificial Life and Robotics*, Beppu, Oita, Japan, 2007.
4. W. Gerstner, W. Kistler, *Spiking Neuron Models. Single Neurons, Populations, Plasticity.* Cambridge Univ. Press, 2002.
5. C. Giaume and K. Mc Carthy, Control of gap-junctional communication in astrocytic networks., *Trends Neuroscience 15*, 1995, pp. 5535–5550.
6. P. Guthrie, J. Knappenberger, M. Segal, M. Bennet, A. Charles and S. Kater., ATP released from astrocyte mediates glial calcium waves.,*J Neuroscience 19*, 1999, pp. 520–528.

7. F. He and Ye. Sun, Glial cells more than support cells?, *Int J Biochem Cell Biol. 39(4)*, 2007, pp. 661–665.
8. M. Ionescu, Gh. Păun and T. Yokomori, Spiking neural P systems, *Fundamenta Informaticae* 71, 2–3, 2006, pp. 279–308.
9. J. Kleijn and M. Koutny, Processes of nets with range arcs, CS-TR 985, School of Computing Science, Newcastle University, Oct. 2006.
10. T. Kohonen, *Self-Organizing Maps*, 3^{rd} ed., Springer-Verlag, Berlin, 2001.
11. W. Maass, Computing with spikes, Special Issue on *Foundations of Information Processing* of TELEMATIK 8, 1, 2002, pp. 32–36.
12. W. Maass and C. Bishop (eds), *Pulsed Neural Networks*. MIT Press, Cambridge, 2003.
13. C. Martín-Vide, J. Pazos, Gh. Păun and A. Rodríguez-Patón, A new class of symbolic abstract neural nets: Tissue P systems, in: Proceedings of COCOON 2002, Singapore, *Lecture Notes in Computer Science* 2387, Springer-Verlag, Berlin, 2002, pp. 290–299.
14. S. Mennerik and C. Zorumski, Glial contribution to excitatory neurotransmission in cultural hippocampal cells., *Nature 368*, 1994, pp. 59–62.
15. W. Nett, S. Oloff and K. McCarthy, Hippocampal astrocytes in situ exhibit calcium oscillations that occur independent of neuronal activity., *J Neurophysiol. 87(1)*, 2002, pp. 528–537.
16. R. Parri and V. Crunelli, An astrocyte bridge from synapse to blood flow., *Nat Neurosci 6(1)*, 2003, pp. 5-6.
17. Gh. Păun, *Computing with Membranes: An Introduction*. Springer, Berlin, 2002.
18. Gh. Păun, M.J. Pérez-Jiménez and G. Rozenberg, Spike trains in spiking neural P systems, *Intern J Found Computer Sci*, to appear (also available at [23]).
19. G. Perea and A. Araque, Communication between astrocytes and neurons: a complex language. *Journal of Physiology Paris* 96, 2002, 199–207.
20. J. Porter and Kd. Mc Carthy, Astrocytic neurotransmitter receptors in situ and in vivo., *Prog Neurobiol. (51)*, 1997, pp. 7817–7830.
21. X. Shen and P. de Wilde, Long-term neuronal behavior caused by two synaptic modification mechanisms. *Neurocomputing* 70, 7-9, 2007, 1482–1488.
22. G. Rozenberg and A. Salomaa (eds), *Handbook of Formal Languages* (3 volumes). Springer, Berlin, 1997.
23. The P Systems Web Page, `http://psystems.disco.unimib.it`

Information Theory over Multisets

Cosmin Bonchiş[1], Cornel Izbaşa[1], Gabriel Ciobanu[2]

[1] Research Institute "e-Austria" Timişoara, Romania
{cosmin, cornel}@ieat.ro
[2] "A.I. Cuza" University, Faculty of Computer Science and
Romanian Academy, Institute of Computer Science
gabriel@info.uaic.ro

"The words, the sad words,
Sometimes surround the time
As a pipe, the water which flows within."
Nichita Stănescu

Summary. Starting from Shannon theory of information, we present the case of producing information in the form of multisets, and encoding information using multisets. We compute the entropy of a multiset information source by constructing an equientropic string source (with interdependent symbols), and we compare this with a string information source with independent symbols. We then study the encoder and channel part of the system, obtaining some results about multiset encoding length and channel capacity.

1 Motivation

The attempt to study information sources which produce multisets instead of strings, and ways to encode information on multisets rather than strings, originates in observing new computational models like membrane systems which employ multisets [5]. Membrane systems have been studied extensively and there are plenty of results regarding their computing power, language hierarchies and complexity. However, while any researcher working with membrane systems (called also P systems) would agree that P systems process information, and that living cells and organisms do this too, we are unaware of any attempt to precisely describe natural ways to encode information on multisets or to study sources of information which produce multisets instead of strings. One could argue that, while some of the information in a living organism is encoded in a sequential manner, like in DNA for example, there might be important molecular information sources which involve multisets (of molecules) in a non-trivial way.

A simple question: given a P system with, say, 2 objects a and 3 objects b from a known vocabulary V (suppose there are no evolution rules), how much

information is present in that system? Also, many examples of P systems perform various computational tasks. Authors of such systems encode the input (usually numbers) in various ways, some by superimposing a string-like structure on the membrane system [1], some by using the natural encoding or the unary numeral system, that is, the natural number n is represented with n objects, for example, a^n. However, just imagine a gland which uses the bloodstream to send molecules to some tissue which, in turn, sends back some other molecules. There is for sure an energy and information exchange. How to describe it? Another, more general way to pose that question is: what are the natural ways to encode numbers, and more generally, information on multisets, and how to measure the encoded information?

If membrane systems, living cells and any other (abstract or concrete) multiset processing machines are understood as information processing machines, then we believe that such questions should be investigated. According to our knowledge, this is the first attempt of such an investigation. We start from the idea that a study of multiset information theory might produce interesting, useful results at least in systems biology; if we understand the *natural* ways to encode information on multisets, there is a chance that *Nature* might be using similar mechanisms.

Another way in which this investigation seems interesting to us is that there is more challenge in efficiently encoding information on multisets, because they constitute a poorer encoding media compared to strings. Encoding information on strings or even richer, more organized and complex structures are obviously possible and have been studied. Removing the symbol order, or their position in the representation as strings can lead to multisets carrying a certain penalty, which deserves a precise description. Order or position do *not* represent essential aspects for information encoding; symbol multiplicity, a native quality of multisets, is *enough* for many valid purposes. We focus mainly on such "natural" approaches to information encoding over multisets, and present some advantages they have over approaches that superimpose a string structure on the multiset. Then we encode information using multisets in a similar way as it is done using strings.

There is also a connection between this work and the theory of numeral systems. The study of number encodings using multisets can be seen as a study of a class of purely non-positional numeral systems.

2 Entropy of an Information Source

Shannon's information theory represents one of the great intellectual achievements of the twentieth century. Information theory has had an important and significant influence on probability theory and ergodic theory, and Shannon's mathematics is a considerable and profound contribution to pure mathematics.

Shannon's important contribution comes from the invention of the source-encoder-channel-decoder-destination model, and from the elegant and general solution of the fundamental problems which he was able to pose in terms of this model. Shannon has provided significant demonstration of the power of coding with delay

in a communication system, the separation of the source and channel coding problems, and he has established the fundamental natural limits on communication. As time goes on, the information theoretic concepts introduced by Shannon become more relevant to day-to-day more complex process of communication.

2.1 Short Review of Shannon Information Theory

We use the notions defined in the classical paper [6] where Shannon has formulated a general model of a communication system which is tractable to a mathematical treatment.

Definition 1. *The quantity H is a reasonable measure of choice or information.*

Consider an information source modeled by a discrete Markov process. For each possible state i of the source there is a set of probabilities $p_i(j)$ associated to the transitions to state j. Each state transition produces a symbol corresponding to the destination state, e.g., if there is a transition from state i to state j, the symbol x_j is produced. Each symbol x_i has an initial probability $p_{i\in\overline{1..n}}$ corresponding to the transition probability from the initial state to each state i.

We can also view this as a random variable X with x_i as events with probabilities p_i, $X = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ p_1 & p_2 & \cdots & p_n \end{pmatrix}$.

There is an entropy H_i for each state. The entropy of the source is defined as the average of these H_i weighted in accordance with the probability of occurrence of the states:

$$H(X) = \sum_i P_i H_i = -\sum_{i,j} P_i p_i(j) \log p_i(j) \tag{1}$$

Suppose there are two symbols x_i, x_j and $p(i,j)$ is the probability of the successive occurrence of x_i and then x_j. The entropy of the joint event is

$$H(i,j) = -\sum_{i,j} p(i,j) \log p(i,j)$$

The probability of symbol x_j to appear after the symbol x_i is the conditional probability $p_i(j)$.

String Entropy

Consider an information source which produces sequences of symbols selected from a set of n independent symbols x_i with probabilities p_i. The entropy formula for such a source is given in [6]:

$$H(X) = \sum_{i=1}^{n} p_i \log_b \frac{1}{p_i}$$

2.2 Multiset Entropy

We consider a discrete information source modeled by a discrete-time first-order Markov process (or Markov chain) which produces multiset messages (as opposed to string messages). A message is a multiset of symbols. To compute the entropy of such a source, we construct an equientropic source which produces strings with mutually dependent symbols. Each string produced by this equientropic source is an exponent of a multiset produced by the multiset source, because a multiset is a string equivalence class.

The entropy of such a source is computed by Shannon's formula 1, where P_i is the probability of state i, and $p_i(j)$ is the transition probability from state i to state j. To compute the probability of the state i we must first observe what is specific for the multisets. The corresponding state trees are presented in the next figures.

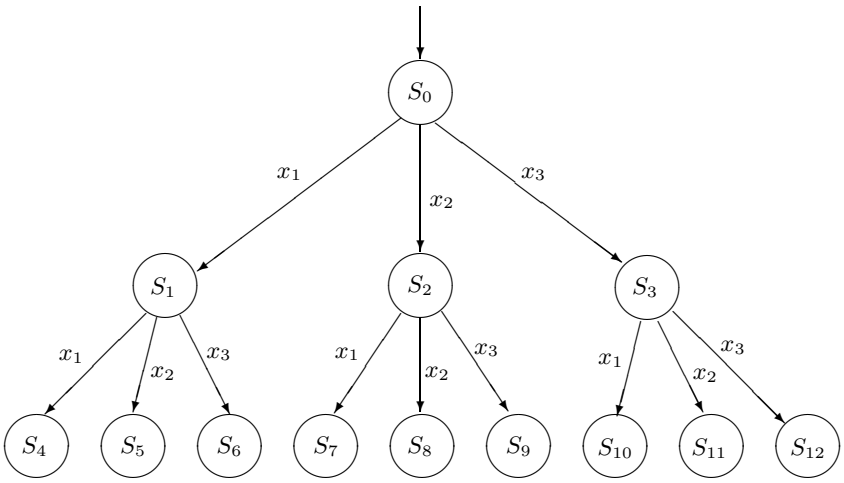


Fig. 1. String source states tree

We take the P_i for the first level of the tree, and because $P_0 = 1$ we get:

$$P_i = P_0 p_0(i) = p_i \tag{2}$$

To compute the transition probability $p_i(j)$ we know that for multisets $p(i, j) = 0$ for $i > j$.

Let N be the number of all symbols (with repetition allowed). Then the most probable number of symbols x_j is $p_j N$. For $i \leq j$, in order to obtain j after i, we observe that the symbols $x_{i>j}$ cannot be produced. Therefore, the probability to obtain j after i is given by the number of favorable cases over all possible cases

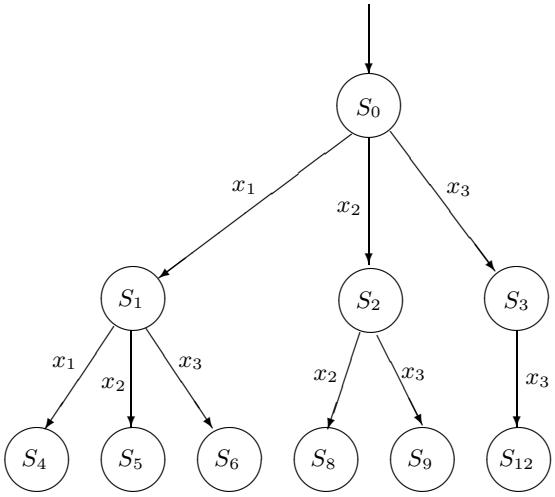


Fig. 2. Multiset source states tree

$$p_i(j) = \frac{p_j N}{N - \sum_{j=1}^{i-1} p_j N} = \frac{p_j}{\sum_{j=i}^{n} p_j}$$

namely

$$p_i(j) = \begin{cases} 0, & i > j \\ \frac{p_j}{\sum_{j=i}^{n} p_j}, & i \leq j \end{cases} \tag{3}$$

Theorem 1. *The entropy formula of a multiset generating information source is:*

$$H(X) = -\sum_{i=1}^{n} p_i \sum_{j=i}^{n} \frac{p_j}{\sum_{k=i}^{n} p_k} \log\left(\frac{p_j}{\sum_{k=i}^{n} p_k}\right). \tag{4}$$

Proof. From 1, 2, and 3 we infer

$$\begin{aligned} H(X) &= -\sum_{i,j,i\leq j} p_i \frac{p_j}{\sum_{k=i}^{n} p_k} \log\left(\frac{p_j}{\sum_{k=i}^{n} p_k}\right) \\ &= -\sum_{i=1}^{n} p_i \sum_{j=i}^{n} \frac{p_j}{\sum_{k=i}^{n} p_k} \log\left(\frac{p_j}{\sum_{k=i}^{n} p_k}\right). \end{aligned}$$

Proposition 1. *When the events are equiprobable, i.e.,* $p_i = \frac{1}{n}$*, then*

$$H(X) = \frac{\log n!}{n}.$$

Proof. We substitute $\frac{1}{n}$ for p_i in equation (4), and get

$$\begin{aligned} H(X) &= -\sum_{i=1}^{n} \frac{1}{n} \sum_{j=i}^{n} \left(\frac{\frac{1}{n}}{\sum_{j=i}^{n} \frac{1}{n}} \log \left(\frac{\frac{1}{n}}{\sum_{j=i}^{n} \frac{1}{n}} \right) \right) \\ &= -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=i}^{n} \frac{1}{n-i+1} \log \frac{1}{n-i+1} \\ &= -\frac{1}{n} \sum_{i=1}^{n} \left(\frac{1}{n-i+1} \log \frac{1}{n-i+1} \right) \sum_{j=i}^{n} 1 \\ &= \frac{1}{n} \sum_{i=1}^{n} \log(n-i+1) \\ &= \frac{1}{n} \sum_{i=1}^{n} \log i = \frac{\log n!}{n}. \end{aligned}$$

String Source Entropy vs. Multiset Source Entropy

Theorem 2. *The entropy of a multiset-producing source* is lower than or equal to *the entropy of an equiprobable string-producing source:*

$$H_{multiset} \leq H_{string(x_i - equiprobable)}$$

Proof. We know that $\sum_{i=1}^{n} p_i = 1 \Rightarrow \sum_{k=i}^{n} p_k \leq 1 \Rightarrow$

$$\frac{p_j}{\sum_{k=i}^{n} p_k} \geq p_j \tag{5}$$

Gibbs inequality suppose that $P = \{p_1, p_2, \ldots, p_n\}$ is a probability distribution. Then for any other probability distribution $Q = \{q_1, q_2, \ldots, q_n\}$ the following inequality holds

$$-\sum_{i=1}^{n} p_i \log p_i \leq -\sum_{i=1}^{n} p_i \log q_i \tag{6}$$

Then

$$H_m(X) = -\sum_{i=1}^{n} p_i \sum_{j=i}^{n} \frac{p_j}{\sum_{k=i}^{n} p_k} \log\left(\frac{p_j}{\sum_{k=i}^{n} p_k}\right) \overset{(5)}{\leq}$$

$$\leq -\sum_{i=1}^{n} p_i \sum_{j=i}^{n} \frac{p_j}{\sum_{k=i}^{n} p_k} \log p_j = -\sum_{i=1}^{n} \frac{p_i}{\sum_{k=i}^{n} p_k} \sum_{j=i}^{n} p_j \log p_j \overset{(6)}{\leq}$$

$$\overset{(6)}{\leq} -\sum_{i=1}^{n} \frac{p_i}{\sum_{k=i}^{n} p_k} \sum_{j=i}^{n} p_j \log q_j$$

with $Q_i = \{q_j | \text{where, } j = \overline{i,n} \text{ and } \sum_{j=i}^{n} q_j = 1\}$, and $q_j = \dfrac{1}{n-i+1}$:

$$-\sum_{i=1}^{n} \frac{p_i}{\sum_{k=i}^{n} p_k} \sum_{j=i}^{n} p_j \log q_j = -\sum_{i=1}^{n} \frac{p_i}{\sum_{k=i}^{n} p_k} \sum_{j=i}^{n} p_j \log \frac{1}{n-i+1}$$

$$= -\sum_{i=1}^{n} \frac{p_i}{\sum_{k=i}^{n} p_k} \left(\log \frac{1}{n-i+1}\right) \sum_{j=i}^{n} p_j = \sum_{i=1}^{n} p_i \log(n-i+1) \leq$$

$$\leq \sum_{i=1}^{n} p_i \log n = \log n = H_{string}(X)_{x_i - equiprobable}$$

Corollary 1. *When X is equiprobable, $H_m \leq H_s$.*

Proof. For $p_i = \dfrac{1}{n}$ we have

$$H_{multiset} = \frac{1}{n} \sum_{i=1}^{n} \log i = \frac{\log n!}{n} \leq \log n = H_{string}$$

Maximum Entropy for a Multiset Source

For a multiset source, equiprobable events do not generate the maximum entropy. This is obtained by maximizing expression 4, which seems difficult in the general case, but we give an example for the simplest case - with two events (a binary multiset source):

$$X = \begin{pmatrix} x_1 & x_2 \\ p_1 & p_2 \end{pmatrix}$$

The multiset entropy for these events is: $H_{multiset}(X) = -p_1(p_1 \log p_1 + p_2 \log p_2)$. Let $p = p_1 \Rightarrow H_{multiset}(X) = -p[p \log p + (1-p)\log(1-p)]$. Since this function has only one maximum in $[0,1]$, we need to solve:

$$H'_{multiset}(X) = 2p[\log(1-p) - \log p] - \log(1-p) = 0.$$

A numerical solution is $p \approx 0.703506$. The maximizing probability distribution is $X \approx \begin{pmatrix} x_1 & x_2 \\ 0.703506 & 0.296494 \end{pmatrix}$ and the maximum entropy is

$$H_{multiset}(X) \approx 0.427636 < H_{string}(X_{equiprobable}) = \log 2 \approx 0.6931472.$$

3 Multiset Encoding and Channel Capacity

After exploring the characteristics of a multiset generating information source, we move to the channel part of the communication system. Properties of previously developed multiset encodings are analyzed in [2, 3]. A formula for the capacity of multiset communication channel is derived based on the Shannon's general formula. Please note that one can have a multiset information source and a usual sequence-based encoder and channel. All the following combinations are possible:

Source/Encoder	Sequential	Multiset
Sequential	[6]	this paper
Multiset	this paper	this paper

Table 1. Source/Encoder types

3.1 String Encoding

We shortly review the results concerning the string encoding.

Encoding Length

We have a set of symbols X to be encoded, and an alphabet A. We consider the uniform encoding. Considering the length l of the encoding, then $X = \{x_i = a_1 a_2 \dots a_l \mid a_j \in A\}$.

If $p_i = P(x_i) = \frac{1}{n}$, then we have

$$H(X) = \sum_{i=1}^{n} \frac{1}{n} \log_b(n) = \log_b(n) \leq l$$

It follows that $n \leq b^l$. For $n \in \mathbb{N}$, $n - b^x = 0$ implies $x_0 = \log_b n$ and so $l = \lceil x_0 \rceil = \lceil \log_b n \rceil$.

Channel Capacity

Definition 2. *[6] The* capacity C *of a discrete channel is given by*

$$C = \lim_{T \to \infty} \frac{\log N(T)}{T}$$

where $N(T)$ is the number of allowed signals of duration T.

Theorem 3. *[6] Let $b_{ij}^{(s)}$ be the duration of the s^{th} symbol which is allowable in state i and leads to state j. Then the channel capacity C is equal to* $\log W$ *where W is the largest real root of the determinant equation:*

$$\left| \sum_s W^{-b_{ij}^{(s)}} - \delta_{ij} \right| = 0$$

where $\delta_{ij} = 1$ if $i = j$, and zero otherwise.

3.2 Multiset Encoding

We present some results related to the multiset encoding.

Encoding Length

We consider a set X of N symbols, an alphabet A, and the length of encoding l, therefore:

$X = \{x_i = a_1^{n_1} a_2^{n_2} \dots a_b^{n_b} \mid \sum_{j=1}^{b} n_j = l,\ a_j \in A\}$.

Proposition 2. *Non-uniform encodings over multisets are shorter than uniform encodings over multisets.*

Proof. Over multisets we have

1. for an uniform encoding: $N \leq N(b,l) = \left\langle {b \atop l} \right\rangle = \binom{b+l-1}{l} = \frac{(b+l-1)!}{l!(b-1)!} = \frac{\prod_{i=1}^{b-1}(l+i)}{(b-1)!}$. If x_0 is the real root of $n - \frac{\prod_{i=1}^{b-1}(x+i)}{(b-1)!} = 0$ then $l = \lceil x_0 \rceil$.
2. for non-uniform encoding: $N \leq N(b+1, l-1) = \left\langle {b+1 \atop l-1} \right\rangle = \binom{b+l-1}{l-1}$ $= \frac{(b+l-1)!}{(l-1)!b!} = \frac{\prod_{i=0}^{b-1}(l+i)}{b!} = \frac{l}{b}\frac{\prod_{i=1}^{b-1}(l+i)}{(b-1)!} = \frac{l}{b}N(b,l)$. Let $x_0^{'}$ be the real root of $n - \frac{\prod_{i=0}^{b-1}(x+i)}{b!} = 0$. Then $l' = \left\lceil x_0^{'} \right\rceil$.

From $n - N(b, x_0) = 0$ and $n - \frac{x_0^{'}}{b}N(b, x_0^{'}) = 0$ we get $N(b, x_0) = \frac{x_0^{'}}{b}N(b, x_0^{'})$. In order to prove $l > l' \iff x_0 > x_0'$, let suppose that $x_0 \leq x_0^{'}$. We have $x_0^{'} > b$ (for sufficiently large numbers), and this implies that $N(b, x_0) \leq N(b, x_0^{'}) < \frac{x_0^{'}}{b}N(b, x_0^{'})$. Since this is false, it follows that $x_0 > x_0'$ implies $l \geq l'$.

Channel Capacity

We consider that a sequence of multisets is transmitted along the channel. The capacity of such a channel is computed for base 4, then some properties of it for any base are presented.

Multiset channel capacity in base 4

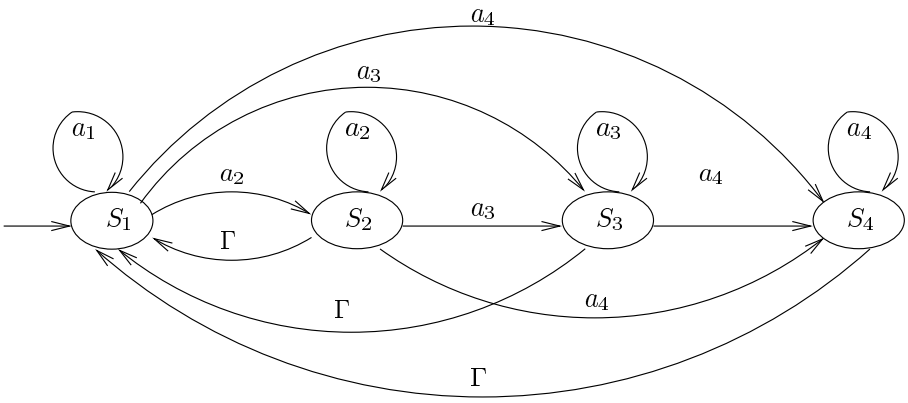


Fig. 3. Multiset channel capacity

In Figure 3 we have a graph $G(V, E)$ with 4 vertices $V = \{S_1, S_2, S_3, S_4\}$ and $E = \{(i, j) \mid i, j = \overline{1..4},\ i \leq j\} \cup \{(i, j) \mid i = 4,\ j = \overline{1..3}\}$

In Theorem 3 we get $b_{ij}^{(a_k)} = t_k$ because we consider that the duration to produce a_k is the same for each $(i, j) \in E$. The determinant equation is

$$\begin{vmatrix} W^{-t_1}-1 & W^{-t_2} & W^{-t_3} & W^{-t_4} \\ W^{-t_\Gamma} & W^{-t_2}-1 & W^{-t_3} & W^{-t_4} \\ W^{-t_\Gamma} & 0 & W^{-t_3}-1 & W^{-t_4} \\ W^{-t_\Gamma} & 0 & 0 & W^{-t_4}-1 \end{vmatrix} = 0$$

If we consider $t_\Gamma = t_k = t$, then the equation becomes

$1 - \dfrac{4}{W^t} + \dfrac{3}{W^{2t}} - \dfrac{1}{W^{3t}} = 0$, and $W_{real} = \sqrt[t]{\dfrac{4 + \sqrt[3]{\frac{47-3\sqrt{93}}{2}} + \sqrt[3]{\frac{47+3\sqrt{93}}{2}}}{3}} \approx \sqrt[t]{3.147899}$.
Therefore $C = \log_4 \sqrt[t]{3.147899}$ for $t = 1$, and so $C \approx 0.827194$.

Multiset channel capacity in base b

The determinant equation is

$$\begin{vmatrix} W^{-t_1}-1 & W^{-t_2} & W^{-t_3} & \cdots & W^{-t_b} \\ W^{-t_\Gamma} & W^{-t_2}-1 & W^{-t_3} & \cdots & W^{-t_b} \\ W^{-t_\Gamma} & 0 & W^{-t_3}-1 & \cdots & W^{-t_b} \\ \vdots & \vdots & \vdots & & \vdots \\ W^{-t_\Gamma} & \cdots & 0 & W^{-t_{b-1}}-1 & W^{-t_b} \\ W^{-t_\Gamma} & 0 & 0 & \cdots & W^{-t_b}-1 \end{vmatrix} = 0$$

Proposition 3. *If $t_\Gamma = t_k = t$, then the determinant equation becomes*

$$(1-\frac{1}{W^t})^{b-1} - \frac{1}{W^t} = 0. \tag{7}$$

The capacity C is given by $C = \log_b W$, where W is the largest real root of the equation (7). Considering $x = W^{-t}$, then we have

$$W = \frac{1}{\sqrt[t]{x}} \Rightarrow C = -\frac{1}{t}\log_b x. \tag{8}$$

Since we need the largest real root W then we should find the smallest positive root x of the equation

$$(1-x)^{b-1} - x = 0 \tag{9}$$

Let $f_b(x) = (1-x)^{b-1} - x$.

Lemma 1. *For all b there is a unique $x_b \in (0,1)$ such that $f_b(x_b) = 0$.*

Proof. We have $f_b'(x) = -(b-1)(1-x)^{b-2} - 1$.

- b is odd $\Rightarrow f_b'(x) = 0$ has the real root $x = 1 + \frac{1}{\sqrt[k-1]{k}} > 1$ and so $f_b'(x) < 0$ for all $x \in (-\infty, 1]$;
- b is even $\Rightarrow f_b'(x) \leq 0$ for all $x \in \mathbb{R}$.

Therefore $f_b(x)$ is decreasing for $x \in (0,1)$, $f_b(0) = 1$ and $f_b(1) = -1$. Then there exists a unique $x_b \in (0,1)$ such that $f_b(x_b) = 0$.

Lemma 2. *The smallest positive root of Equation (9) is decreasing with respect to b. More exactly, for all b we have $x_b \geq x_{b+1}$, where x_b is the smallest positive root of $f_b(x) = 0$.*

Proof. $f_{b+1}(x) - f_b(x) = (1-x)^b - x - ((1-x)^{b-1} - x) = -x(1-x)^{b-1}$. Then $f_{b+1}(x) - f_b(x) \leq 0$ for all $x \in (0,1)$. Since $f_{b+1}(x_b) \leq 0$ and $f_{b+1}(0) = 1$, then we have $x_{b+1} \in (0, x_b)$ according to Lemma 1.

Theorem 4. *Channel capacity is an increasing function with respect to b.*

Proof. This follows by Lemma 2 and Equation (8).

Remark 1. When $n = 2$, the capacity is $C = \frac{1}{t}$.

Proof. From $1 - \frac{2}{W^t} = 0$ we get $C = \log_2 \sqrt[t]{2} = \frac{1}{t}$.

4 Conclusion

Based on Shannon's classical work, we present a multiset entropy formula of an information source. We also present some relationships between this entropy and the string entropy. For a binary multiset source, we compute an approximate maximal value for the entropy. Using the determinant capacity formula, we compute the multiset channel capacity in base 4, and we describe some properties of the multiset channel capacity in base b. As future work we plan to further explore the properties of multiset based communication systems, and to develop some methods for computing the maximal multiset entropy in the general case.

A poetic vision of communication

Nichita Stănescu (1933-1983) was a Romanian poet proposed for Nobel Prize for literature. Here is his view of words and communication, first in Romanian and then in English (translation is ours).

> "Cuvintele / nu au loc decât în centrul lucrurilor, / numai înconjurate de lucruri. // Numele lucrurilor / nu e niciodata afară.
> Şi totuşi / cuvintele, tristele, / înconjoară câteodata timpul / ca o ţeavă, apa care curge prin ea. // ... ca şi cum ar fi lucruri..., / oho, ca şi cum ar fi lucruri..."
>
> "Words / do not belong but are in the center of things, / only surrounded by things. // The names of the things are never outside.
> But still / the words, the sad words, / sometimes surround the time / as a pipe, the water which flows within.// ... as they would be things..., / oh, as they would be things."
>
> *Nichita Stănescu*
>
> "For Nichita Stănescu, the pipe of words is for time what the communication channel is for a message; time flows through the pipe made of words as a message passes as a fluid through that which we call a communication channel."
>
> *Solomon Marcus [4]*

References

1. A. Atanasiu: Arithmetic with Membranes. *Pre-Proceedings of the Workshop on Multiset Processing*, Curtea de Argeş, 2000, 1–17.
2. C. Bonchiş, G. Ciobanu, C. Izbaşa: Encodings and Arithmetic Operations in Membrane Computing. *Theory and Applications of Models of Computation*, Lecture Notes in Computer Science vol. 3959, Springer, Berlin, 2006, 618–627.
3. C. Bonchiş, G. Ciobanu, C. Izbaşa: Number Encodings and Arithmetics over Multisets. *SYNASC'06: 8th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. Timişoara, IEEE Computer Society, 2006, 354–361.

4. S. Marcus: *Intâlnirea Extremelor*. Paralela 45, Bucharest, 2005, 166.
5. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
6. C.E. Shannon: A Mathematical Theory of Communication. *Bell System Technical Journal*, 27 (1948), 379–423, and 623–656.

VisualTissue: A Friendly Tool to Study Tissue P Systems Solutions for Graph Problems

Rafael Borrego-Ropero, Daniel Díaz-Pernil, Juan A. Nepomuceno

Escuela Técnica Superior de Ingenería Informática
Universidad de Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
{rborrego,sbdani,janepo}@us.es

Summary. P systems can be classified in two main groups: P systems with the membrane structure described by a tree, and tissue P systems with the membranes placed in the nodes of an arbitrary graph. **NP**-complete problems have been solved in linear time by trading space for time in the framework of recognizing tissue P systems with cell division. The design of this kind of systems is not an easy task to understand. In this paper we present a software application to help the design of solutions to **NP**-complete problems in the framework of recognizing tissue P systems with cell division.

VisualTissue application can be downloaded from the web:
www.visualtissue.es.kz.

1 Introduction

Membrane Computing is an emergent branch of Natural Computing introduced by Gh. Păun in [9], which, considering as computations the processes that take place into living cells, constructs a new non-deterministic model of computation. In membrane computing there are two types of frameworks: P systems with the membrane structure described by a tree, inspired from the cell, and tissue P systems with the membranes placed in the nodes of an arbitrary graph. The second type corresponds to the idea of forming a network of membranes linked in a specific manner and working together, [10]. In both types, the main idea is having multisets of objects placed in compartments and evolving according to given rules in a synchronous non-deterministic maximally parallel manner.

In the last years, this new field has been addressed in different ways: the study of computational properties such as computational power or complexity classes, definition of new variants of membrane systems closer to biological reality, using Membrane Computing as a new framework for performing biological simulations, etc. In the P systems web page [18] several softwares can be found. Most of them are

thought in order to run an experiment which is built using some kind of membrane system as a simulation framework, for example, for simulating biological systems, as in [14]. On the other hand, another group of software applications are thought as an easy way of visualizing the computations of a P system, [8], that is to say, as a learning tool.

In this paper we present a new visual tool called *VisualTissue* which have been developed in order to help users to understand the computations of tissue P systems with cell division. We have considered the reference [4], which solves 3-col problem in the framework of tissue P systems with cell division. Polynomial solutions to **NP**-complete problems in Membrane Computing are done by trading time for space, in a theoretical way. Although real implementations of such systems are no possible, because it should be necessary to implement the maximal parallelism in some way, there are very interesting problems to threat in this framework, for example, the $\mathbf{P} \neq \mathbf{NP}$ conjecture [13]. Our tool helps the user to understand the performance of one class of such systems.

This paper is organized as follows: in Section 2 tissue P systems with cell division and the 3-col problem are explained, in Section 3 the *VisualTissue* application software is presented and some of the most important implementation aspects are commented. Finally, ideas for future work are formulated.

2 Tissue P Systems with Cell Division

We will give a definition of this model, but first we briefly recall some of the concepts used later.

2.1 Preliminaries

An *alphabet*, Σ, is a non empty set, whose elements are called *symbols*. An ordered sequence of symbols is a *string*. The number of symbols in a string u is the *length* of the string, and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of strings of length n built with symbols from the alphabet Σ is denoted by Σ^n and $\Sigma^* = \cup_{n\geq 0}\Sigma^n$. A *language* over Σ is a subset from Σ^*.

A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset, then its *support* is defined as $supp(m) = \{x \in A \,|\, f(x) > 0\}$ and its *size* is defined as $\sum_{x\in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite).

If $m = (A, f)$ is a finite multiset over A, then it will be denoted by $m = \{\{a_1, \ldots, a_k\}\}$, where each element a_i occurs $f(a_i)$ times, or by a string containing the symbols $a_1, \ldots, a_k$.

A *graph* G is a pair $G = (V, E)$ where V is the set of vertices and E is the set of edges, each one of which is a (unordered) pair of (different) vertices. If $\{u, v\} \in E$, we say that u is *adjacent* to v (and also v is *adjacent* to u). The *degree* of $v \in V$ is the number of adjacent vertices to v.

In what follows, we assume that the reader is already familiar with the basic notions and terminology underlying P systems. For details, see [10].

2.2 Tissue P Systems with Cell Division

In the first definition of the model of tissue P systems [6, 7] the membrane structure does not change along the computation. Based on the cell-like model of P systems with active membranes, Gh. Păun et al. presented in [12] a new model of tissue P systems *with cell division*. The biological inspiration is clear: alive tissues are not *static* network of cells, since cells are duplicated via mitosis in a natural way.

The main features of this model, from the computational point of view, are that cells are not polarized (the opposite holds in the cell-like model of P systems with active membranes, see [10]); the cells obtained by division have the same labels as the original cell and if a cell is divided, its interaction with other cells or with the environment is blocked during the mitosis process. In some sense, this means that while a cell is dividing it closes the communication channels with other cells and with the environment.

Formally, a *tissue P system with cell division* of degree $q \geq 1$ is a tuple of the form

$$\Pi = (\Gamma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_0),$$

where:

1. Γ is a finite *alphabet*, whose symbols will be called *objects*.
2. $w_1, \ldots, w_q$ are strings over Γ.
3. $\mathcal{E} \subseteq \Gamma$.
4. $\mathcal{R}$ is a finite set of rules of the following forms:
   (a) *Communication rules*: $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$.
   (b) *Division rules*: $[a]_i \rightarrow [b]_i[c]_i$, where $i \in \{1, 2, \ldots, q\}$ and $a, b, c \in \Gamma$.
5. $i_0 \in \{0, 1, 2, \ldots, q\}$.

A tissue P system with cell division of degree $q \geq 1$ can be seen as a set of q cells (each one consisting of an elementary membrane) labeled by $1, 2, \ldots, q$. We shall use 0 to refer to the environment, and i_0 denotes the output region (which can be the region inside a cell or the environment).

The communication rules determine a virtual graph, where the nodes are the cells and the edges indicate if it is possible for pairs of cells to communicate directly. This is a dynamical graph, as new nodes can appear produced by the application of division rules.

The strings $w_1, \ldots, w_q$ describe the multisets of objects placed in the q cells of the system. We interpret that $\mathcal{E} \subseteq \Gamma$ is the set of objects placed in the environment, each one of them in an arbitrarily large amount of copies.

A communication rule $(i, u/v, j)$ can be applied over two cells i and j such that u is contained in cell i and v is contained in cell j. The application of this rule means that the objects of the multisets represented by u and v are interchanged between the two cells.

The division rule $[a]_i \rightarrow [b]_i[c]_i$ can be applied to a cell i containing object a. The application of this rule divides this cell into two new cells with the same label. All the objects in the original cell are replicated and copied in each of the new cells, with the exception of the object a, which is replaced by the object b in the first new cell and by c in the second one.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way. In one step, each object in a cell can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e., in each step we apply a maximal set of rules. This way of applying rules has only one restriction: when a cell is divided, the division rule is the only one which is applied for that cell in that step; the objects inside that cell do not evolve in that step.

2.3 Recognizing Tissue P Systems with Cell Division

NP-completeness has been usually studied in the framework of *decision problems*. Let us recall that a decision problem is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X.

In order to study the computing efficiency for solving **NP**-complete decision problems, a special class of tissue P systems with cell division is introduced in [12]: *recognizing tissue P systems*. The key idea of such recognizing system is the same one as from recognizing P systems with a cell-like structure.

Recognizing cell-like P systems were introduced in [15] and they are the natural framework to study and solve decision problems within Membrane Computing, 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, recognizing cell-like 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 cell*. The output of the computation (`yes` or `no`) is sent to the environment. In this way, cell-like P systems with input and external output are devices which can be seen as black boxes, in the sense that the user provides the data before the computation starts, and then waits *outside* the P system until it sends to the environment the output in the last step of the computation.

A recognizing tissue P system with cell division of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \Sigma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_{in}, i_0),$$

where:

- $(\Gamma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_0)$ is a tissue P system with cell division of degree $q \geq 1$ (as defined in the previous section).

- The working alphabet Γ has two distinguished objects `yes` and `no`, present in at least one copy in some initial multisets $w_1, \ldots, w_q$, but not present in $\mathcal{E}$.
- Σ is an (input) alphabet strictly contained in Γ.
- $i_{in} \in \{1, \ldots, q\}$ is the input cell.
- The output region i_0 is the environment.
- All computations halt.
- 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.

The computations of the system Π with input $w \in \Gamma^*$ start from a configuration of the form $(w_1, w_2, \ldots, w_{i_{in}} w, \ldots, w_q; \mathcal{E})$, that is, after adding the multiset w to the contents of the input cell i_{in}. The multiset w is *recognized* by Π if and only if the object `yes` is sent to the environment, in the last step of the corresponding computation. $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object `yes` (respectively, `no`) appears in the environment associated to the corresponding halting configuration of $\mathcal{C}$.

2.4 A Solution to the 3–coloring Problem

A k–coloring ($k \geq 1$) of an undirected graph $G = (V, E)$ is a function $f : V \to \{1, \ldots, k\}$, where the numbers are interpreted as colors. We say that G is k–colorable if there exists a k–coloring, f, such that $f(u) \neq f(v)$ for every edge $\{u, v\} \in E$ (such a k–coloring f is said to be *valid*).

The 3–coloring problem is the following: *given an undirected graph G, decide whether or not G is 3-colorable*; that is, if there exists a valid 3–coloring of G.

This problem is related to the famous Four Color Conjecture (solved by Appel and Haken [2, 3]). It is a particular case of the colorability problem: *Given an undirected graph G and a number k, decide whether G is k-coloreable.* The **NP**-completeness of the 3–coloring problem was proved by Stockmeyer [16] (see [5]).

First of all we define a polynomial encoding of the 3–coloring problem in a family $\mathbf{\Pi}$ of P systems constructed as in [4]. Let $u = (V, E)$ be an instance of the problem, with n vertices and m edges. Then we consider a *size* mapping on the set of instances defined as $s(u) = \langle n, m \rangle$. The codification of the instance will be the multiset $cod(u) = \{\{A_{ij} : \{A_i, A_j\} \in E \wedge 1 \leq i < j \leq n\}\} \cup \{\{w^q\}\}$.

The recognizing tissue P system with cell division that was used to solve the 3–coloring problem in [4] is defined as follows.

For each $n, m \in \mathbb{N}$, we consider the system

$$\Pi(\langle n, m \rangle) = (\Gamma(\langle n, m \rangle), \Sigma(n), w_1, w_2(n), \mathcal{R}(\langle n, m \rangle), \mathcal{E}(\langle n, m \rangle), i_{in}, i_0)$$

where:

- $\Gamma(\langle n, m \rangle)$ is the set

$\{A_i, R_i, T_i, B_i, G_i, \overline{R}_i, \overline{B}_i, \overline{G}_i : 1 \leq i \leq n\} \cup$
$\{a_i : 1 \leq i \leq 2n + m + \lceil log\ m \rceil + 11\} \cup \{c_i : 1 \leq i \leq 2n + 1\} \cup$
$\{d_i : 1 \leq i \leq \lceil log\ m \rceil + 1\} \cup \{f_i : 2 \leq i \leq m + \lceil log\ m \rceil + 6\} \cup$
$\{A_{ij}, P_{ij}, \overline{P}_{ij}, R_{ij}, B_{ij}, G_{ij} : 1 \leq i < j \leq n\} \cup \{b, D, \overline{D}, e, T, S, N, \flat, \texttt{yes}, \texttt{no}\}$

- $\Sigma(n) = \{A_{ij} : 1 \leq i < j \leq n\}$
- $w_1 = \{\{a_1, b, c_1, \texttt{yes}, \texttt{no}\}\}$
- $w_2(n) = \{\{D, A_1, \ldots, A_n\}\}$
- $\mathcal{R}(\langle n, m\rangle)$ is the set of rules:

1. **Division rules:**
   $r_{1,i} \equiv [A_i]_2 \rightarrow [R_i]_2[T_i]_2$ for $i = 1, \ldots, n$
   $r_{2,i} \equiv [T_i]_2 \rightarrow [B_i]_2[G_i]_2$ for $i = 1, \ldots, n$
2. **Communication rules:**
   $r_{3,i} \equiv (1, a_i/a_{i+1}, 0)$ for $i = 1, \ldots, 2n + m + \lceil log\ m \rceil + 10$
   $r_{4,i} \equiv (1, c_i/c_{i+1}^2, 0)$ for $i = 1, \ldots, 2n$
   $r_5 \equiv (1, c_{2n+1}/D, 2)$
   $r_6 \equiv (2, c_{2n+1}/d_1\overline{D}, 0)$
   $r_{7,i} \equiv (2, d_i/d_{i+1}^2, 0)$ for $i = 1, \ldots, \lceil log\ m \rceil$
   $r_8 \equiv (2, \overline{D}/e\, f_2, 0)$
   $r_{9,i} \equiv (2, f_i/f_{i+1}, 0)$ for $i = 2, \ldots, m + \lceil log\ m \rceil + 5$
   $r_{10,ij} \equiv (2, d_{\lceil log\ m \rceil + 1} A_{ij}/P_{ij}, 0)$ for $1 \leq i < j \leq n$
   $r_{11,ij} \equiv (2, P_{ij}/R_{ij}\overline{P}_{ij}, 0)$ for $1 \leq i < j \leq n$
   $r_{12,ij} \equiv (2, \overline{P}_{ij}/B_{ij}G_{ij}, 0)$ for $1 \leq i < j \leq n$
   $r_{13,ij} \equiv (2, R_iR_{ij}/R_i\overline{R}_j, 0)$ for $1 \leq i < j \leq n$
   $r_{14,ij} \equiv (2, B_iB_{ij}/B_i\overline{B}_j, 0)$ for $1 \leq i < j \leq n$
   $r_{15,ij} \equiv (2, G_iG_{ij}/G_i\overline{G}_j, 0)$ for $1 \leq i < j \leq n$
   $r_{16,j} \equiv (2, \overline{R}_jR_j/\flat, 0)$ for $1 \leq j \leq n$
   $r_{17,j} \equiv (2, \overline{B}_jB_j/\flat, 0)$ for $1 \leq j \leq n$
   $r_{18,j} \equiv (2, \overline{G}_jG_j/\flat, 0)$ for $1 \leq j \leq n$
   $r_{19} \equiv (2, e\,\flat/\lambda, 0)$
   $r_{20} \equiv (2, e\ f_{m+\lceil log\ m \rceil + 6}/T, 0)$
   $r_{21} \equiv (2, T/\lambda, 1)$
   $r_{22} \equiv (1, b\,T/S, 0)$
   $r_{23} \equiv (1, S\ \texttt{yes}/\lambda, 0)$
   $r_{24} \equiv (1, b\ a_{2n+m+\lceil log\ m \rceil + 11}/N, 0)$
   $r_{25} \equiv (1, N\ \texttt{no}/\lambda, 0)$

- $\mathcal{E}(\langle n, m\rangle) = \Gamma(\langle n, m\rangle) - \{\texttt{yes}, \texttt{no}\}$
- $i_{in} = 2$ is the *input cell.*
- $i_0 = 0$ is the *output region.*

3 A Look Inside VisualTissue

VisualTissue is a visual software application to understand the design of solutions for **NP**-complete problems, in the framework of recognizing tissue P system with cell division.

Some programming decisions have been taken in order to develop the application. We have chosen *C#* as programming language because it is a portable and a powerful object-oriented programming language. A great graphical package is available with the language so that *C#* is a good language for developing a visual tool. The software follows the Model-View-Controller (MVC), an architecture model of software development used in interactive systems. Three different parts or layers can be distinguished: data handling layer, algorithmic or business logic layer, and user interface or graphical layer. With this, it is easier to do maintenance of the code.

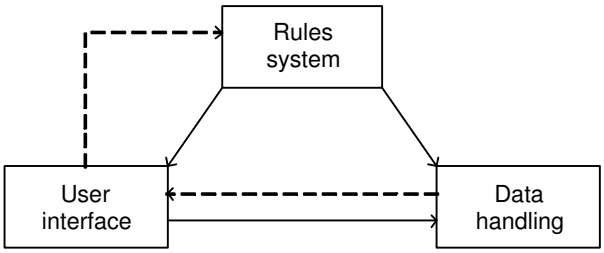


Fig. 1. Model-View-Controller architecture software

The application can handle data stored in XML and normal text files, which are compatible with *Algraf Project*, [19]. XML files can be easily generated and it is the best way to interact with data generated by other programs. Rules of the systems are fixed for the solution of each problem. In a future version, the user will be able to introduce his own tissue P system rules in order to study other possible solutions.

The algorithmic layer implements a recognizing tissue P system with cell division. All rules are applied in a non deterministic and maximal parallel way. The design of the tissue P system machine for solving the problem is non-deterministic until the $2n$ step. Every computation path reaches the same configuration in $2n$ steps, and after that the machine is deterministic and confluent. Consequently, we have chosen only a computation path in order to implement the tissue P system in the software, namely the one determined by the lexicographical election of the rules in non-deterministic steps. Other graph problems solutions can be easily added.

The graphical layer allows the user a visual and friendly interaction with the application. At the end of the simulation one can see the colored graph result of the problem if the answer is *yes*.

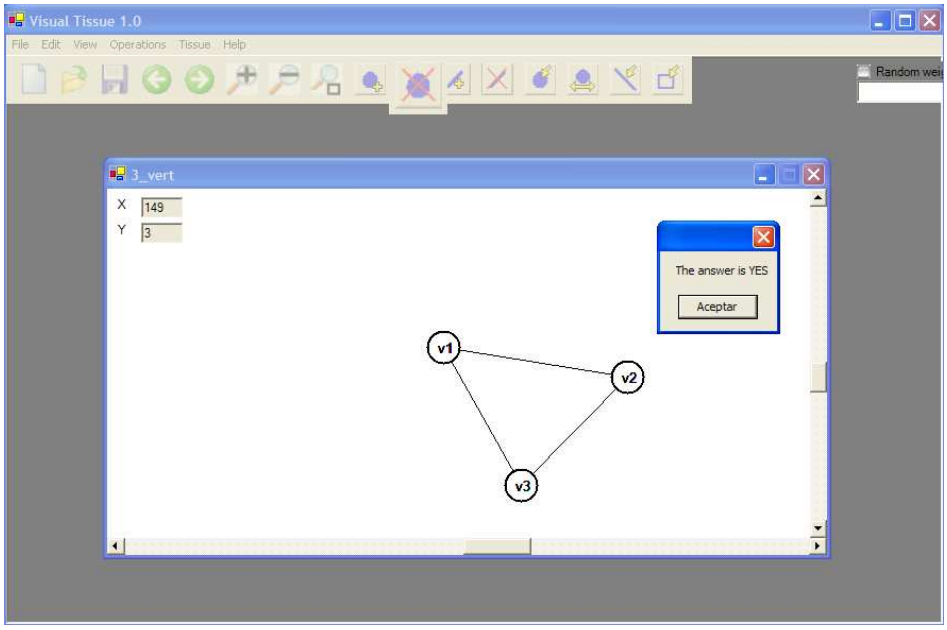


Fig. 2. Firstly, the graph to be studied must be drawn. A small screen show us wether it is 3-colorable or not.

3.1 An User Overview of the Application

This software tool allows us to follow step by step the execution of tissue P system with membrane division when solving the 3-col problem, [4]. The answers for a specific problem will be *yes* or *no*, and the colored graph is provided if it is possible. The user basically can do three different operations to run the system:

- Load or draw with the mouse the graph which is going to be studied. Several graphical options are available in the main window. Graphs can be handled in a easy way and images can be saved as image files or pdf files.
- Choose tissue 3-col algorithm in order to carry out the simulation. A second screen is showed in which each computation step can be observed. For each step, a graphical situation can be viewed and the different rules applied in each step are written in the bottom of the screen. The picture can be easily handled and each moment of the performance of the algorithm, and can be saved in different images format, or press the buttons with arrows to move between steps.
- Choose GO TO option to go directly to a specific step of the system.

VisualTissue software is available on the web, at: `www.visualtissue.es.kz`

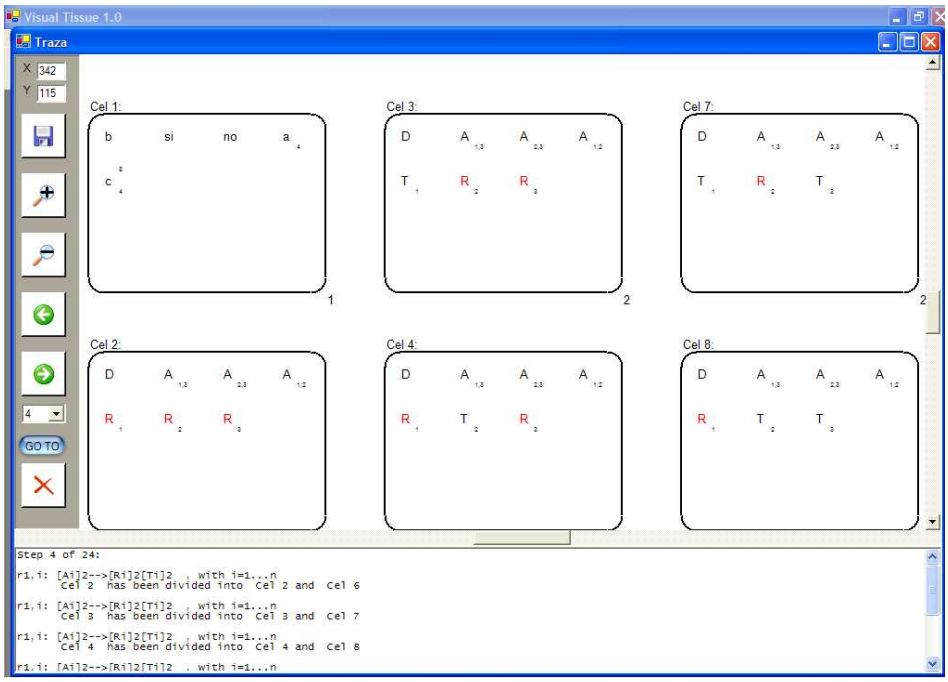


Fig. 3. The performance of the tissue P system can be observed step by step.

4 Future Works

Future works will be focused on different improvements as, for example: to handle input files of rules, work with other graph problems, consider other kinds of tissue P systems rules such as membrane creation, etc. One of the most interesting future tasks will be to build a software which simulates a Spiking Neural P system, extending this graphical tool. This future software task will be not only a way of visualizing the performance of the system, but also a framework to do simulations with Spiking Neural P systems too.

References

1. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *The Molecular Biology of the Cell*. Fourth Edition, Garland Publ. Inc., London, 2002.
2. K. Appel, W. Haken: Every planar map is 4-colorable - 1: Discharging. *Illinois Journal of Mathematics*, 21 (1977), 429–490.
3. K. Appel, W. Haken: Every planar map is 4-colorable - 2: Reducibility. *Illinois Journal of Mathematics*, 21 (1977), 491–567.
4. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear–time tissue P system based solution for the 3–coloring problem. *Theoretical Computer Science*, to appear.

5. M.R. Garey, D.S. Johnson: *Computers and Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
6. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón: A new class of symbolic abstract neural nets: Tissue P systems. *Lecture Notes in Computer Science* **2387** (2002), 290–299.
7. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón: Tissue P systems. *Theoretical Computer Science*, 296 (2003), 295–326.
8. I.A. Nepomuceno-Chamorro: A Java simulator for membrane computing. *Journal of Universal Computer Science*, 10 (2001), 620–629.
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, M.J. Pérez-Jiménez: Recent computing models inspired from biology: DNA and membrane computing. *Theoria*, 18 (2003), 72–84.
12. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez: Tissue P system with cell division. In Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez and F. Sancho-Caparrini (eds.), *Second Brainstorming Week on Membrane Computing*, Sevilla, Report RGNC 01/2004, 2004, 380–386.
13. M.J. Pérez-Jiménez: An approach to computational complexity in membrane computing. In G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa (eds.) *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 3365 (2005), 85–109.
14. M.J. Pérez-Jiménez, F.J. Romero-Campero: P systems, a new computational modelling tool for systems biology. *Transactions on Computational Systems Biology VI. Lecture Notes in Bioinformatics*, Springer-Verlag, Berlin, 4220 (2006), 176-197
15. 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 and Gy. Vaszil (eds.), *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, 2003, 284–294.
16. L.J. Stockmeyer: Planar 3-colorability is NP-complete. *SIGACT News*, 5, 3 (1973), 19–25.
17. ISI web page `http://esi-topics.com/erf/october2003.html`
18. P systems web page `http://psystems.disco.unimib.it/`
19. Algraf Project web page `http://www.algraf.es.kz/`

Towards a Causal Semantics for Brane Calculi

Nadia Busi

Dipartimento di Scienze dell'Informazione, Università di Bologna,
Mura A. Zamboni 7, I-40127 Bologna, Italy.
busi@cs.unibo.it

Summary. Brane Calculi are a family of biologically inspired process calculi, proposed in [6] to model the interactions of dynamically nested membranes. We propose a semantics that describes the causal dependencies occurring between the reactions of a system described in Brane Calculi. We investigate the basic properties that are satisfied by such a semantics. The notion of causality turns out to be quite relevant for biological systems, as it permits to point out which events occurring in a biological pathway are necessary for another event to happen.

1 Introduction

Brane calculi [6] are a family of process calculi proposed for modeling the behavior of biological membranes.

The formal investigation of biological membranes has been initiated by G. Păun [20], in the field of automata and formal language theory, with the definition of P systems. In a process algebraic setting, the notions of membranes and compartments are explicitly represented in BioAmbients [23], a variant of Mobile Ambients [8] based on a set of biologically inspired primitives of interaction.

Brane calculi represent an evolution of BioAmbients: the main difference w.r.t. previous approaches consists in the fact that the active entities reside on membranes, and not inside membranes. In [6] two basic instances of brane calculi have been proposed: the Phago/Exo/Pino (PEP) and the Mate/Bud/Drip (MBD) calculi.

In this paper we concentrate on the MBD calculus. The primitives of MBD are inspired by membrane fusion (mate) and fission (mito). Because membrane fission is an uncontrollable process that can split a membrane at an arbitrary place, it is replaced by two simpler operations: *budding*, that is splitting off one internal membrane, and *dripping*, that consists in splitting off zero internal membranes.

The aim of this work is to start an investigation of the causal dependencies arising in Brane Calculi, and more precisely in the MBD calculus. The main motivation for this work comes from system biology, as the understanding of the causal

relations occurring between the events of a complex biological pathway could be of precious help, e.g., for limiting the search space in the case some unpredicted event occurs.

The study of a causal semantics for process algebras dates back to the early nineties for CCS [17] (see, e.g., [10, 9, 15]), and to the mid nineties for the π-calculus [18] (see, e.g., [1, 3, 11, 12]).

To the best of our knowledge, the only other work that deals with causality in bio-inspired calculi is [14], where a causal semantics for Beta Binders [21, 22] – based on the π-calculus semantics and on the enhanced operational semantics approach of [12] – is defined. One of the main differences between Beta Binders and Brane Calculi is that the membrane structure in Beta Binders is flat, whereas in Brane Calculi the membranes are nested to form a hierarchical structure. As we will see, this difference has a deep impact on the complexity of the causal relation. The other differences between the two approaches will be discussed throughout the paper.

The paper is organized as follows. Section 2 introduces the syntax and the interleaving semantics for the MBD fragment of the Brane Calculus. Sections 3 and 4 are devoted to the definition of the causal semantics. Section 3 provides an informal description of the features of the causal semantics we are defining, and illustrates the problems that have arised through a list of examples. The formal definition of the causal semantics is in Section 4, followed by a discussion concerning the properties that are (not) satisfied by such a semantics. Finally, Section 5 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 [6].

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:[1]

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:*

$$
(\texttt{par}) \quad \frac{P \;\rightarrow\; Q}{P \circ R \;\rightarrow\; Q \circ R} \qquad\qquad (\texttt{brane}) \quad \frac{P \;\rightarrow\; Q}{\sigma(\!| P |\!) \;\rightarrow\; \sigma(\!| Q |\!)}
$$

$$
(\texttt{strucong}) \quad \frac{P' \equiv P \qquad P \;\rightarrow\; Q \qquad Q \equiv Q'}{P' \;\rightarrow\; Q'}
$$

[1] With abuse of notation we use $\equiv$ to denote both structural congruence on systems and structural congruence on processes.

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.

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 [6], are inspired by membrane fusion and splitting. To make membrane splitting more controllable, in [6] 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 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 synchronize 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) $mate_n.\sigma|\sigma_0(\!| P |\!) \circ mate_n^{\perp}.\tau|\tau_0(\!| Q |\!) \rightarrow \sigma|\sigma_0|\tau|\tau_0(\!| P \circ Q |\!)$

(bud) $bud_n^{\perp}(\rho).\tau|\tau_0(\!| bud_n.\sigma|\sigma_0(\!| P |\!) \circ Q |\!) \rightarrow \rho(\!| \sigma|\sigma_0(\!| P |\!) |\!) \circ \tau|\tau_0(\!| Q |\!)$

(drip) $drip(\rho).\sigma|\sigma_0(\!| P |\!) \rightarrow \rho(\!|\ |\!) \circ \sigma|\sigma_0(\!| P |\!)$

3 A Causal Semantics for MBD: an informal explanation

In this section we provide a causal semantics for MBD.

To define a causal semantics for process calculi, we follow the approach used in [15] for CCS, and in [1] for the π-calculus. The idea consists in decorating the reaction relation with two pieces of information:

- a fresh name k, that is associated to the reaction and it is taken from the set of causes $\mathcal{K}$;
- a set $H \subseteq \mathcal{K}$, containing all the names associated to the already occurred reactions, that represent a cause for the current reaction.

To keep track of the names of the already occurred reactions that may represent a cause for the reactions that may happen in the future, the syntax of the terms of the calculus is enriched with such an information on causal dependencies. As in [1], for the sake of clarity we only keep track of the so called immediate causes, as the set of general causes can be reconstructed by transitive closure of the immediate causal relation. We will provide more explanation on this point with an example in the following part of the paper.

Now we start with an informal introduction of causality in MBD. First we discuss how the standard kinds of causality arising in most process calculi – i.e., those due to the prefix structure of processes and to the synchronization of two complementary actions – scale to Brane Calculi. Then we perform a design choice concerned with the semantics of calculi for membranes, and finally we discuss other features peculiar of the MBD operations.

3.1 Classical causal dependencies: structural and synchronization causality

We start the kind of causal dependencies that arises in all process calculi, namely, structural causality and synchronization causality.

Structural causality arises from the prefix structure of terms. Consider for example the following system:

$$drip(\sigma).drip(\rho)(\!|\quad|\!)$$

Such a system can first create a new membrane with process σ, followed by the creation of a second new membrane with process ρ; i.e., it can perform the sequence of reactions

$$drip(\sigma).drip(\rho)(\!|\quad|\!) \;\rightarrow drip(\rho)(\!|\quad|\!) \circ \sigma(\!|\quad|\!) \rightarrow 0(\!|\quad|\!) \circ \sigma(\!|\quad|\!) \circ \rho(\!|\quad|\!)$$

The creation of the first membrane is a necessary condition for the creation of the second membrane, hence we say that the execution of the $drip(\rho)$ operation *is caused by* the execution of the $drip(\sigma)$ operation.

To remember the fact that the action $drip(\sigma)$ will be a cause for the actions performed by the continuation of the prefix, we replace the $drip(\sigma)$ prefix with a

causal operator containing the cause name associated to the $drip(\sigma)$ action. Thus, we obtain the following causal reactions:

$$drip(\sigma).drip(\rho)(\!|\ \ |\!) \xrightarrow{h;\emptyset} \{h\} :: drip(\rho)(\!|\ \ |\!) \circ \sigma(\!|\ \ |\!) \xrightarrow{k;\{h\}} \{k\} :: 0(\!|\ \ |\!) \circ \sigma(\!|\ \ |\!) \circ \rho(\!|\ \ |\!)$$

The decoration $h;\emptyset$ of the first reaction means that the first reaction is labeled with the causal name h and that its set of causes is empty. The decoration $k;\{h\}$ of the second reaction means that the second reaction has associated the causal name k, and it is caused by the reaction named h (i.e., the first reaction). The process $\{h\} :: drip(\rho)$ means that the first action performed by process $drip(\rho)$ is caused by the reaction named h. Note that – for the sake of brevity – process $\{k\} :: 0$ is decorated only with the immediate cause $\{k\}$, as the whole set of causes, i.e., $\{h, k\}$ can be easily constructed.

To lighten the notation, in the following we will drop the parentheses surrounding the set of causes, if this creates no confusion.

The other kind of causality, i.e., synchronization causality, arises when two processes synchronize on complementary actions. Consider the system

$$drip(\sigma_1).mate_n.drip(\tau_1)(\!|\ \ |\!) \circ drip(\sigma_2).mate_n^{\perp}.drip(\tau_2)(\!|\ \ |\!)$$

The mate reaction can be performed when both the actions $drip(\sigma_1)$ and $drip(\sigma_2)$ have been performed; hence, it is caused by both actions. We obtain the following:

$$\begin{array}{l} drip(\sigma_1).mate_n.drip(\tau_1)(\!|\ \ |\!) \circ drip(\sigma_2).mate_n^{\perp}.drip(\tau_2)(\!|\ \ |\!) \xrightarrow{h_1;\emptyset} \\ h_1 :: mate_n.drip(\tau_1)(\!|\ \ |\!) \circ drip(\sigma_2).mate_n^{\perp}.drip(\tau_2)(\!|\ \ |\!) \circ \sigma_1(\!|\ \ |\!) \xrightarrow{h_2;\emptyset} \\ h_1 :: mate_n.drip(\tau_1)(\!|\ \ |\!) \circ h_2 :: mate_n^{\perp}.drip(\tau_2)(\!|\ \ |\!) \circ \sigma_1(\!|\ \ |\!) \circ \sigma_2(\!|\ \ |\!) \xrightarrow{k;h_1,h_2} \\ k :: (drip(\tau_1) \mid drip(\tau_2)(\!|\ \ |\!) \circ \sigma_1(\!|\ \ |\!) \circ \sigma_2(\!|\ \ |\!) \end{array}$$

Hence, the (label k of the) mate reaction will be an immediate cause for both $drip(\tau_1)$ and $drip(\tau_2)$, and the global set of causes of these two drip actions will be $\{h_1, h_2, k\}$.

3.2 How does the causes distribute over the parallel components of a membrane process?

When moving to consider the features of the causal relation peculiar of membrane calculi, a first question arises: if a process on a membrane performs an action, this action will be a cause only for its continuation (and eventually for the continuation of its synchronizing action), or for the whole process on the membrane? In other words, consider the system

$$mate_n \mid drip(\sigma)(\!|\ \ |\!) \circ mate_n^{\perp}(\!|\ \ |\!)$$

If the system performs the mate synchronization, then the drip action will be caused or not by the mate action? The assumption that the drip will be caused

by the mate may have the following biological interpretation: when a membrane interaction operation is performed, all the membrane is involved, and at the end of the operation the structure of all the membrane has been affected. This assumption is considered in [14] in the definition of a causal semantics for Beta-binders [21, 22] , a bio-inspired process calculus roughly consisting of unnested compartments enclosing π-calculus processes. It is also used in [2] for the definition of a maximal parallelism semantics (i.e., step semantics with maximal progress: if an action can be performed in the current step, then it must be performed) for MBD. It is also the common approach used in the definition of the maximal parallelism semantics for Membrane Systems with evolving membranes (see, e.g., [19, 20]). In the present paper, we consider the opposite approach: in the above system, we consider the drip operation independent from the mate operation, as the drip operation can be executed regardless of the fact that the mate synchronization has been performed or not. The biological interpretation may be the following: the membrane proteins and the part of the lipid bilayer involved in the mate synchronization are different from the membrane proteins and the part of the bilayer that is performing the drip operation, and they lie in different parts of the membrane surface.

Thus, we consider the following causal reactions:

$$\begin{array}{l} mate_n \mid drip(\sigma)(\!|\quad|\!) \circ mate_n^{\perp}(\!|\quad|\!) \xrightarrow{h;\emptyset} \\ h :: 0 \mid drip(\sigma) \mid h :: 0(\!|\quad|\!) \xrightarrow{k;\emptyset} \\ h :: 0 \mid k :: 0 \mid h :: 0(\!|\quad|\!) \end{array}$$

Note that the information on the causes of the empty process 0 is completely irrelevant, hence in the following we will replace $H :: 0$ with 0.

3.3 Causal dependencies generated by the mate operation

Now we analyze the features peculiar of the MBD operations. According to the informal explanation above, a mate action turns out to be a cause for the continuations of the mate and the co-mate prefixes that synchronize to perform the operation. However, when considering the mate operation, a more subtle kind of causality, we call environment causality, is originated, e.g., between the mate action and the processes on the child membranes of the two membranes performing the mate and co-mate actions. This causality is due to the fact that the environment of such child membranes, i.e., the set of membranes with which they can interact, is increased by the execution of the mate action.

Mate followed by mate

Consider the following process:

$$\begin{array}{l} mate_n(\!|\ (mate_m \mid mate_o)(\!|\quad|\!) \circ\ mate_o^{\perp}(\!|\quad|\!)\ |\!) \circ \\ mate_n^{\perp}(\!|\ mate_m^{\perp}(\!|\quad|\!)\ |\!) \end{array}$$

Now the mate synchronization on m cannot be performed, as the two membranes whose processes can synchronize on such an operation belong to different membranes. On the other hand, the mate synchronization on o can take place, as both membranes whose processes can synchronize on such an operation belong to the same membrane.

However, if the mate synchronization on n takes place, the two external membranes are fused; this results in a change of the environment of the child membrane; now the mate on m can take place, as the two child membranes now belong to the same father membrane and can get in contact. Hence, the mate on m causally depends on the mate on n.

To this aim, we decorate the processes of the child membranes of the external membrane performing a mate with label k in the following way: the child membranes on the left are decorated with the enriched label k_i^+, whereas the child membrane on the right with the enriched complementary label k_i^-. [2] Note that we cannot simply decorate both groups of child membranes with label k, otherwise we are no longer able to distinguish between the synchronization on m, that is caused by k, and the synchronization on o which has no causes.

The enriched labels are used in the following way: when two processes preceded by enriched labels synchronize on a mate operation, the label k will be a cause for such a synchronization if one process in decorated with an enriched label and the synchronizing process is decorated with the complementary label.

We obtain the following causal reductions:

$$\begin{array}{l} mate_n(\!|\ (mate_m \mid mate_o)(\!|\ \ |\!)\ \circ\ mate_o^{\perp}(\!|\ \ |\!)\ |\!)\ \circ \\ mate_n^{\perp}(\!|\ mate_m^{\perp}(\!|\ \ |\!)\ |\!) \\ \xrightarrow{h;\emptyset} \\ (0 \mid 0)(\!|\ (h_i^+ :: mate_m \mid h_i^+ :: mate_o)(\!|\ \ |\!)\ \circ\ h_i^+ :: mate_o^{\perp}(\!|\ \ |\!)\ \circ \\ h_i^- :: mate_m^{\perp}(\!|\ \ |\!)\ |\!) \end{array}$$

Now, if the mate on m is executed, then it will be caused by h, as the *mate* and *comate* processes are labeled with h_i^+ and h_i^-, respectively:

$$\begin{array}{l} (0 \mid 0)(\!|\ (h_i^+ :: mate_m \mid h_i^+ :: mate_o)(\!|\ \ |\!)\ \circ\ h_i^+ :: mate_o^{\perp}(\!|\ \ |\!)\ \circ \\ h_i^- :: mate_m^{\perp}(\!|\ \ |\!)\ |\!) \\ \xrightarrow{k;h} \\ (0 \mid 0)(\!|\ (0 \mid h_i^+ :: mate_o)(\!|\ \ |\!)\ \circ\ h_i^+ :: mate_o^{\perp}(\!|\ \ |\!)\ \circ \\ 0(\!|\ \ |\!)\ |\!) \end{array}$$

On the other hand, if the mate on o is executed, then it is not caused by h_n, because the *mate* and *comate* processes are labeled with the same label h_i^+.

[2] The i in the labels k_i^+ and k_i^- stands for "internal", and means that the action with label k has been performed by the father membrane. The need for such a label will be made clear in the following.

Mate followed by bud

A similar problem arises between the father and the child membrane when a bud operation is performed. Consider the following process:

$$(mate_n \mid bud_m^{\perp}(\rho_1))(\!|\ bud_m(\!|\ \ |\!)\ \circ\ bud_o(\!|\ \ |\!)\ |\!)\circ$$
$$(mate_n^{\perp} \mid bud_o^{\perp}(\rho_2))(\!|\ \ |\!)$$

Now only the bud on m can be performed, as the membrane performing the bud on o is not a child of the membrane performing the corresponding cobud. However, the bud on o can be performed after the mate on n is performed, hence the bud causally depends on the mate. Thus, besides decorating the children of a membrane performing a mate (resp comate) with complementary labels k_i^+ (resp. k_i^-), we also decorate the subprocesses in parallel with the subprocess performing the mate (resp. the comate) with k_e^+ (resp. k_e^-)[3]. When a bud is performed, it is caused by k if the process performing the cobud on the father membrane is decorated, e.g., with k_e^+ and the process performing the bud on the child membrane is decorated with k_e^-.

Note that, in case of a mate followed by a drip, the decorated causes will give rise to no causal dependency: the drip is caused by the mate only if the mate (or the comate) is a prefix of the drip.

3.4 Causal dependencies generated by the bud and the drip operations

The bud (resp. drip) operation create a new membrane – whose membrane process is specified in the cobud (resp. drip) action – surrounding the child membrane that performs the synchronizing bud action (resp. with no children). As the new membrane does not exist before the bud (resp. drip) operation is performed, all the actions that such a membrane will perform are caused by the bud (resp. drip) operation.

Consider the following system:

$$bud_n^{\perp}(drip(\sigma))(\!|\ bud_n(\!|\ \ |\!)\ |\!)$$

This system can perform the following causal reactions:

$$bud_n^{\perp}(drip(\sigma))(\!|\ bud_n(\!|\ \ |\!)\ |\!) \xrightarrow{h;\emptyset}$$
$$0(\!|\ \ |\!)\ \circ\ h :: drip(\sigma)(\!|\ 0(\!|\ \ |\!)\ |\!) \xrightarrow{k;h}$$
$$0(\!|\ \ |\!)\ \circ\ 0(\!|\ 0(\!|\ \ |\!)\ |\!)\circ\ k :: \sigma(\!|\ \ |\!)$$

We note that the bud operation generates no environmental cause. Regarding the child membranes, they are essentially divided into two sets, thus possibly preventing some mate (or bud) operation that was possible before to happen. On the other hand, the other processes in the father membrane are left unchanged.

[3] Here the e means "external"

4 A causal semantics for MBD: the formal definition

In this section we provide a formal definition of the notions introduced in the previous section.

Definition 6. *Let $\mathcal{K}$ be a denumerable set of cause names, disjoint from the set Names. Let Deco($\mathcal{K}$) be the set*

$$Deco(\mathcal{K}) = \mathcal{K} \cup \{k_x^y \mid k \in \mathcal{K} \wedge x \in \{i,e\} \wedge y \in \{+,-\}\}$$

The set of membrane processes with causes are defined by the following grammar:

$$\tilde{\sigma}, \tilde{\tau} \quad ::= \quad 0 \mid \tilde{\sigma}|\tilde{\tau} \mid !\tilde{\sigma} \mid K :: a.\sigma$$

Variables a, b range over MBD actions specified in Definition 4, $K \subseteq Deco(\mathcal{K})$, and $a.\sigma$ is a sequential process as defined in Definition 1.

The set of systems with causes is defined as in Definition 1, but using processes with causes instead of processes to decorate membranes.

The set of causes preceding the process 0 is useless, hence it has been omitted. We omit the ˜ over processes if it is clear from the context that they are processes with causes.

To define the causal semantics, we need an auxiliary operator on processes (and on systems) permitting to add a set of causes in front of each sequential subprocess of the process (resp. of the processes associated to the most external membranes of the system.

Definition 7. *Given $K \subseteq Deco(\mathcal{K})$, the operator $K\Mapsto$ is inductively defined on processes with causes as follows:*

$$\begin{aligned} K\Mapsto 0 &= 0 \\ K\Mapsto (\sigma \mid \tau) &= K\Mapsto \sigma \mid K\Mapsto \tau \\ K\Mapsto !\sigma &= !K\Mapsto \sigma \\ K\Mapsto H :: a.\sigma &= H \cup K :: a.\sigma \end{aligned}$$

The operator $K\Mapsto$ is inductively defined on systems as follows:

$$\begin{aligned} K\Mapsto \diamond &= \diamond \\ K\Mapsto (P \circ Q) &= K\Mapsto P \ \circ \ K\Mapsto Q \\ K\Mapsto (!P) &= !K\Mapsto P \\ K\Mapsto (\sigma(\!| \ P \ |\!)) &= (K\Mapsto \sigma)(\!| \ P \ |\!) \end{aligned}$$

If the set K is a singleton, often we omit the surrounding parenthesis in the operator $K\Mapsto$; thus we write, e.g., $k\Mapsto P$ instead of $\{k\}\Mapsto P$.

Now we are ready to define the causal semantics for MBD. We write $P \xrightarrow{k;H} P'$ to denote the fact that system P performs an action – to which we associate the cause name k – that is caused by the (previously occurred) actions whose action

names form the set H. The cause name k is a fresh name: this means that it does not occur in P and that it has not used yet in the current computation.

The structural congruence relation is the that in Definition 2. The causal rules are obtained by decorating the rules in Definition 3 with the causal information.

Definition 8. *The causal reaction rules are the following:*

$$(\texttt{par}) \quad \frac{P \xrightarrow{k;H} Q}{P \circ R \xrightarrow{k;H} Q \circ R} \qquad (\texttt{brane}) \quad \frac{P \xrightarrow{k;H} Q}{\sigma(\!|\, P \,|\!) \xrightarrow{k;H} \sigma(\!|\, Q \,|\!)}$$

$$(\texttt{strucong}) \quad \frac{P' \equiv P \qquad P \xrightarrow{k;H} Q \qquad Q \equiv Q'}{P' \xrightarrow{k;H} Q'}$$

Now we are ready to define the causal reaction relation for MBD.

Definition 9. *The causal reaction relation for MBD is the least relation containing the following axioms, and satisfying the rules in Definition 8:*

$$\begin{array}{ll} (\texttt{kmate}) & (H_1 :: mate_n.\sigma)|\sigma_0(\!|\, P \,|\!) \;\circ\; (H_2 :: mate_n^{\perp}.\tau)|\tau_0(\!|\, Q \,|\!) \xrightarrow{k;H_1 \oplus_m H_2} \\ & ((k \cup H_1 \ominus_m H_2) \Vdash \sigma \;\mid\; k_e^+ \Vdash \sigma_0 \;\mid \\ & \;(k \cup H_2 \ominus_m H_1) \Vdash \tau \;\mid\; k_e^- \Vdash \tau_0) \quad (\!|\, k_i^+ \Vdash (P) \;\circ\; k_i^- \Vdash (Q) \,|\!) \end{array}$$

$$\begin{array}{ll} (\texttt{kbud}) & (H_1 :: bud_n^{\perp}(\rho).\tau)|\tau_0(\!|\, (H_2 :: bud_n.\sigma)|\sigma_0(\!|\, P \,|\!) \circ Q \,|\!) \xrightarrow{k;H_1 \oplus_b H_2} \\ & (k \cup H_1 \odot_b H_2) :: \rho(\!|\, ((k \cup H_2 \ominus_b H_1) \Vdash \sigma)|\sigma_0(\!|\, P \,|\!) \,|\!) \circ \\ & ((k \cup H_1 \ominus_m H_2) \Vdash \tau) \mid \tau_0(\!|\, Q \,|\!) \end{array}$$

$$\begin{array}{ll} (\texttt{kdrip}) & (H :: drip(\rho).\sigma|\sigma_0)(\!|\, P \,|\!) \xrightarrow{k;f_d(H)} \\ & k \cup f'_d(H) \Vdash \rho(\!|\;|\!) \;\circ\; (k' \cup f''_d(H) \Vdash \sigma) \mid \sigma_0(\!|\, P \,|\!) \end{array}$$

The auxiliary functions are defined as follows:

$$\begin{array}{l} H_1 \oplus_m H_2 = \{k \mid k \in (H_1 \cup H_2) \cap \mathcal{K}\} \cup \\ \qquad\qquad \{k \mid \{k_i^+, k_i^-\} \subseteq (H_1 \cup H_2)\} \\ H_1 \ominus_m H_2 = \{k_x^y \in H_1 \mid k \notin (H_1 \oplus_m H_2)\} \end{array}$$

$$\begin{array}{l} H_1 \oplus_b H_2 = \{k \mid k \in (H_1 \cup H_2) \cap \mathcal{K}\} \cup \\ \qquad\qquad \{k \mid k_e^x \in H_1 \wedge k_e^y \in H_2 \wedge x \neq y\} \\ H_1 \odot_b H_2 = \{k_i^y \in H_1 \mid k \notin (H_1 \oplus_b H_2)\} \\ H_1 \ominus_b H_2 = \{k_x^y \in H_1 \mid k \notin (H_1 \oplus_b H_2)\} \end{array}$$

$$\begin{array}{l} f_d(H) = \{k \mid k \in H \cap \mathcal{K}\} \\ f'_d(H) = \{k_i^x \mid k_i^x \in H \wedge x \in \{+,-\}\} \\ f''_d(H) = \{k_y^x \mid k_y^x \in H \wedge x \in \{+,-\} \wedge y \in \{i,e\}\} \end{array}$$

The auxiliary functions describe the way in which the (decorated) causes propagate when a reduction is performed.

In the merge operation, the set of causes of the merge action, denoted by $H_1 \oplus_m H_2$, contains both the causes of the mate and the comate operation, as well as those causes h such that h_i^+ decorates one of the merging membranes and h_i^- decorates the other (this means that the two membranes have become sibling membranes by the execution of a mate operation with label h). The external decorated causes are not taken into account because they are concerned with bud operations between a father and a child membrane, and not with sibling membranes. To avoid redundancy, the set of causes $H_1 \ominus H_2$ of the continuation of the mate action is obtained by removing the decorated causes whose name appears as a cause of the current mate synchronization (and analogously for comate, bud and cobud actions).

In the bud operation, the set of causes of the bud actions, denoted by $H_1 \oplus_b H_2$, contains both the causes of the bud and the cobud operation, as well as those causes h such that, e.g., h_e^+ decorates the father membrane and h_i^- decorates the child membrane. The set of causes of the newly created membrane is denoted by $H_1 \odot H_2$, and contains only internal causes; they are needed because in the system

$$mate_n(\!|\ bud_o^{\perp}(mate_m)(\!|\ bud_o(\!|\ \ |\!)\ |\!)\ |\!)\ \circ mate_n^{\perp}(\!|\ mate_m^{\perp}\ |\!)$$

the mate synchronization on m can happen only if the mate synchronization on n has been performed.

Regarding the drip operation, here there is no synchronization; hence, the set of causes labeling the reduction relation, represented by $f_d(H)$ is exactly the set of nondecorated causes. The newly created membrane is decorated with function $f'_d(H)$ containing only internal causes; they are needed because in the system

$$mate_n(\!|\ drip(mate_m)(\!|\ \ |\!)\ |\!)\ \circ mate_n^{\perp}(\!|\ mate_m^{\perp}\ |\!)$$

the mate synchronization on m can happen only if the mate synchronization on n has been performed. The external causes are not needed because they are used for bud synchronization between father and child, and the newly created membrane has no child taken from the old membrane.

4.1 Properties of the causal semantics

The only interesting property enjoyed by the causal semantics is the retrievability of the interleaving semantics. We start defining the function *DropCause* which removes the causes from processes and systems.

Definition 10. *The function DropCause is defined inductively on processes with causes in the following way:*

$$\begin{aligned} DropCause(0) &= 0 \\ DropCause(\tilde{\sigma}|\tilde{\tau}) &= DropCause(\tilde{\sigma}) \mid DropCause(\tilde{\tau}) \\ DropCause(!\tilde{\sigma}) &= !DropCause(\tilde{\sigma}) \\ DropCause(K :: a.\sigma) &= a.\sigma \end{aligned}$$

The function DropCause is defined inductively on systems with causes in the following way:

$$\begin{aligned} DropCause(\diamond) &= \diamond \\ DropCause(P \circ Q) &= DropCause(P) \ \circ \ DropCause(Q) \\ DropCause(!P) &= !DropCause(P) \\ DropCause(\tilde{\sigma}(\!|P|\!)) &= DropCause(\tilde{\sigma})(\!|DropCause(P)|\!) \end{aligned}$$

Theorem 1. *Let P be a system with causes. The following properties hold:*

- *if $P \xrightarrow{k;H} P'$ then $DropCause(P) \rightarrow DropCause(P')$;*
- *if $DropCause(P) \rightarrow Q$ then there exist a system with causes P', a cause name k and a set of cause names H such that $P \xrightarrow{k;H} P'$ and $Q = DropCause(P')$.*

The so-called diamond property, stating that if two non-causally related actions can happen one after the other, then they can happen also in the other order, and at the end they reach the same system, does not hold. In our setting, the diamond property can be formally defined as follows: given a system P, if $P \xrightarrow{h;H} P' \xrightarrow{k;K} P''$ and $h \notin K$, then there exists a system q such that $P \xrightarrow{k;K} Q \xrightarrow{h;H} P''$.

Consider e.g. the following system:

$$bud_m^{\perp}(0)(\!|\ mate_n(\!|\ \ |\!)\ \circ\ (bud_m \mid mate_n^{\perp})(\!|\ \ |\!)\ |\!)$$

This system can perform the mate action, followed by the bud action. Moreover, the two actions are independent, i.e., causally unrelated. However, if we first perform the bud action, then the submembrane $mate_n^{\perp})(\!|\ \ |\!)$ is isolated from the other submembrane, and the merge can no longer take place. However, there is no reason to consider the bud action as causally dependent on the mate action, as the bud action can actually independently occur at the beginning of the computation. Nevertheless, there is a form of asymmetric conflict between the two actions: the occurrence of the bud action prevents the mate action to happen, but the vice versa does not hold. A similar phenomenon takes place, e.g., in Petri nets with read and inhibitor arcs (see [4] for a discussion on this topic). A possible way to capture this kind of asymmetric conflict is based on the following idea: if a mate synchronization with causal label k is performed, we decorate the process of the father membrane of the two membrane that are fusing with a label, say k_f (where the f stands for "father"). When the bud operation is performed, the cobud prefix is decorated with k_f, whereas the bud prefix is decorate with k_e^-. As a but synchronization is performed in this situation, we get the information that – even if the bud does not causally depend from the mate – the bud synchronization cannot be swapped with the mate.

Even if there is no asymmetric conflict, there are situations where the diamond property does not hold. Even if the two actions can be performed in either order, the final states that are reached are different. Take the system

$$bud_n^{\perp}(0).\sigma(\!|\ (bud_n.\tau \mid drip(\rho))(\!|\ \ |\!)\ |\!)$$

If the bud action is performed first, then the membrane with process ρ will be dripped inside the newly created membrane, labeled with process 0. On the other hand, if the drip is performed first then the membrane with process ρ remains inside the older membrane (with process $bud_n^{\perp}(0).\sigma$ or with the continuation σ.

5 Conclusion

In this paper we tackled the problem of defining a causal semantics for an instance of Brane Calculi, namely, the MBD calculus.

As already pointed out in [14], we think that the study of the causal dependencies that arise between the actions performed by a process is of primary importance for biologically inspired calculi, because of its possible application to the analysis of complex biological pathways.

This paper represents a first step in this direction, but a lot of work remains to be done. The next step is the study of the causal semantics for the PEP calculus, and its integration with the causal semantics for MBD. Then, we will move to the full Brane Calculus, that, besides the membrane-membrane interaction primitives of PEP and MBD, also contains objects representing free-floating molecules, and primitives for molecule-molecule and membrane-molecule interactions. When the definition of a causal semantics has been completed, we will start investigating the causal dependencies arising in biological pathways involving membranes, such as, e.g., the LDL Cholesterol Degradation Pathway [16], that has been modeled in the full Brane Calculus in [5].

We also plan to perform a thorough investigation of the properties that are enjoyed by the causal semantics, and possibly to refine the definition of the causal semantics in order to fulfill some of the properties. For example, a possible solution in order to obtain a causal semantics that partially enjoys the diamond property has been sketched in the previous section, and deserves further investigation.

We also plan to extend our investigation to other calculi/systems whose membranes are organized in a dynamically evolving hierarchical structure, such as, e.g., the Projective Brane Calculus [13] or Membrane Systems with active, evolving membranes.

References

1. M. Boreale, D. Sangiorgi: A fully abstract semantics for causality in the π-Cclculus. *Acta Informatica*, 35, 5 (1998), 353–400. An extended abstract appeared in *Proc. STACS* 1995, 243–254.
2. N. Busi: On the computational power of the Mate/Bud/Drip Brane Calculus: interleaving vs. maximal parallelism. In *Proc 6th International Workshop on Membrane Computing* (WMC6), LNCS 3850, Springer, 2006.
3. N. Busi, R. Gorrieri: A Petri net semantics for π-calculus. In *Proc. Concur'95*, LNCS 962, Springer, 1995, 145–159.

4. N. Busi, G.M. Pinna: Comparing truly concurrent semantics for contextual place/transition nets with inhibitor and read arcs. *Fundam. Inform.*, 44, 3 (2000), 209–244.
5. N. Busi, C. Zandron: Modeling and analysis of biological processes by mem(brane) calculi and systems. In *Proceedings of the Winter Simulation Conference* (WSC 2006), ACM, 2006.
6. L. Cardelli: Brane Calculi - Interactions of biological membranes. In *Proc. Computational Methods in System Biology* 2004 (CMSB 2004), LNCS 3082, Springer, 2005.
7. L. Cardelli: *Abstract Machines for System Biology*. Draft, 2005.
8. L. Cardelli, A.D. Gordon: Mobile ambients. *Theoretical Computer Science*, 240, 1 (2000), 177–213.
9. Ph. Darondeau, P. Degano: Causal trees. In *Proc. ICALP'89*, LNCS 372, Springer, 1989, 234–248.
10. P. Degano, R. De Nicola, U. Montanari: Partial ordering descriptions and observations of nondeterministic concurrent processes. In *Proc. REX School/Workshop on Linear Time, Branching Time and Partial Order in Logic and Models of Concurrency*, LNCS 354, Springer, 1989, 438–466.
11. P. Degano, C. Priami: Causality for mobile processes. In *Proc. ICALP'95*, LNCS 944, Springer, 1995, 660–671.
12. P. Degano, C. Priami: Non interleaving semantics for mobile processes. *Theoretical Computer Science*, 216, 1-2 (1999), 237–270.
13. V. Danos, S. Pradalier: Projective brane calculus. In *Proc. Computational Methods in System Biology* 2004 (CMSB 2004), LNCS 3082, Springer, 2005.
14. M.L. Guerriero, C. Priami: *Causality and Concurrency in Beta-binders*. TR-01-2006 The Microsoft Research - University of Trento Centre for Computational and Systems Biology, 2006.
15. A. Kiehn: Proof systems for cause based equivalences. In *Proc. MFCS'93*, LNCS 711, Springer, 1993.
16. H. Lodish, A. Berk, P. Matsudaira, C.A. Kaiser, M. Krieger, M.P. Scott, S.L. Zipursky, J. Darnell: *Molecular Cell Biology*. W.H. Freeman and Company, 4th edition, 1999.
17. R. Milner: *Communication and Concurrency*. Prentice-Hall, 1989.
18. R. Milner, J. Parrow, D. Walker: A calculus of mobile processes. *Information and Computation*, 100 (1992), 1–77.
19. G. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
20. G. Păun: *Membrane Computing. An Introduction*. Springer, 2002.
21. C. Priami, P. Quaglia: Beta binders for biological interactions. In *Proc. of Computational Methods in Systems Biology*, LNCS 3082, Springer, 2005, 20–33.
22. C. Priami, P. Quaglia: Operational patterns in beta-binders. *T. Comp. Sys. Biology*, 1 (2005), 50–65.
23. 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.

A Linear Solution for Subset Sum Problem with Tissue P Systems with Cell Division

Daniel Díaz-Pernil, Miguel A. Gutiérrez-Naranjo,
Mario J. Pérez-Jiménez, Agustín Riscos-Núñez

Research Group on Natural Computing
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
{sbdani,magutier,marper,ariscosn}@us.es

Summary. Tissue P systems are a computing model in the framework of Membrane Computing where the tree-like membrane structure is replaced by a general graph. Recently, it has been shown that endowing these P systems with cell division, **NP**-complete problems can be solved in polynomial time. In this paper we present a solution to the Subset Sum problem via a family of such devices, and we also include the formal verification of such solution. This is the first solution to a numerical **NP**-complete problem by using tissue P systems with cell division.

1 Introduction

Membrane Computing is a bio-inspired computing model based on 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 generically called *P Systems*.

In the initial definition of the cell-like model of P systems [6], membranes are hierarchically arranged in a tree-like structure. Its biological inspiration comes from the morphology of cells, where small vesicles are surrounded by larger ones. This biological structure can be abstracted into a tree-like graph, where the root represents the skin of the cell (i.e. the outermost membrane) and the leaves represent membranes that do not contain any other membrane (elementary membranes). Besides, two nodes in the graph are connected if they represent two membranes such that one of them contains the other one.

Recently, new models of P systems have been explored. One of them is the model of *tissue P systems* where the tree-like membrane structure is replaced by a general graph. This model has two biological inspirations (see [3, 4]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and

communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules, which were introduced as communication rules for P systems in [5]. In symport rules, objects cooperate to traverse a membrane together in the same direction, whereas in the case of antiport rules, objects residing at both sides of the membrane cross it simultaneously but in opposite directions.

This paper is devoted to the study of the computational efficiency of tissue P systems with cell division. In literature, different models of cell-like P systems have been successfully used in order to design efficient solutions to **NP**-complete problems (see, for example, [2] and the references therein). These solutions are obtained by generating an exponential amount of workspace in polynomial time and using parallelism to check simultaneously all the candidate solutions.

From the seminal definition of tissue P systems [3, 4], several research lines have been developed and other variants have arisen (see [1] and references therein). One of the most interesting variants of tissue P systems was presented in [8], where the definition of tissue P systems is combined with the one of P systems with active membranes, yielding *tissue P systems with cell division*. The biological inspiration is clear: alive tissues are not *static* networks of cells, since cells are duplicated via mitosis in a natural way. One of the main features of such tissue P systems with cell division is related to their computational efficiency. In [8], a polynomial-time solution to the **NP**-complete problem SAT is shown, and in [1] a linear-time solution for the 3-COL problem was presented. In this paper we go on with the research in this model and present a linear-time solution to another well-known numerical **NP**-complete problem: the Subset Sum problem.

The paper is organized as follows: first we recall some preliminary concepts and the definition of tissue P systems with cell division. Next, recognizing tissue P systems are briefly described. A linear–time solution to the Subset Sum problem is presented in the following section, including a short overview of the computation and the formal verification of the solution. Finally, some conclusions and lines for future research are presented.

2 Preliminaries

In this section we briefly recall some of the concepts used later on in the paper.

An *alphabet*, Σ, is a non empty set, whose elements are called *symbols*. An ordered sequence of symbols is a *string*. The number of symbols in a string u is the *length* of the string, and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. The set of strings of length n built with symbols from the alphabet Σ is denoted by Σ^n and $\Sigma^* = \cup_{n\geq 0}\Sigma^n$. A *language* over Σ is a subset from Σ^*.

A *multiset* m over a set A is a pair (A, f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A, f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A \,|\, f(x) > 0\}$ and its *size* is defined as $\sum_{x\in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite).

A finite multiset $m = (A, f)$ will be denoted as $m = \{\{x_1^{f(x_1)}, \ldots, x_k^{f(x_k)}\}\}$, where $supp(m) = \{x_1, \ldots, x_k\}$, or alternatively as the string $x_1^{f(x_1)} \cdots x_k^{f(x_k)}$. The union of multisets will be denoted as concatenation when using the string notation.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For details, see the handbook [7].

3 Tissue P Systems with Cell Division

In the first definition of the model of tissue P systems [3, 4] the membrane structure did not change along the computation. The main features of tissue P systems with cell division, from the computational point of view, are that cells obtained by division have the same labels as the original cell, and if a cell is divided, then its interaction with other cells or with the environment is blocked during the mitosis process. In some sense, this means that while a cell is dividing it closes all its communication channels. This features imply that the underlying graph is dynamic, as nodes can be added during the computation by division and the edges can be deleted/re-established for dividing cells.

Actually, the underlying graph of connections between cells will not be handled explicitly: the initial structure is implicitly given by the number of initial cells (nodes) and the communication rules (marking edges that connect nodes); the opening/closing edges will be controlled by the semantics.

Formally, a *tissue P system with cell division* of initial degree $q \geq 1$ is a tuple of the form $\Pi = (\Gamma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_0)$, where:

1. Γ is a finite *alphabet*, whose symbols will be called *objects*.
2. $w_1, \ldots, w_q$ are strings over Γ.
3. $\mathcal{E} \subseteq \Gamma$.
4. $\mathcal{R}$ is a finite set of rules of the following form:
   (a) *Communication rules*: $(i, u/v, j)$, for $i, j \in \{0, 1, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$.
   (b) *Division rules*: $[a]_i \rightarrow [b]_i[c]_i$, where $i \in \{1, 2, \ldots, q\}$ and $a, b, c \in \Gamma$.
5. $i_0 \in \{0, 1, 2, \ldots, q\}$.

A tissue P system with cell division of degree $q \geq 1$ can be seen as a set of q cells labelled by $1, 2, \ldots, q$. We shall use 0 as the label of the environment, and i_0 for the output region (which can be the region inside a cell or the environment). As we said before, the underlying graph expressing connections between cells is implicit, being determined by the communication rules: the nodes are the cells and the edges indicate if it is possible for pairs of cells to communicate directly. The communication rule $(i, u/v, j)$ can be applied over two cells i and j such that u is contained in cell i and v is contained in cell j, and neither i nor j are being divided. The application of this rule means that the objects of the multisets represented by u and v are interchanged between the two cells.

The strings $w_1, \ldots, w_q$ describe the multisets of objects placed initially in the q cells of the system. We interpret that $\mathcal{E} \subseteq \Gamma$ is the set of objects placed in the environment, each one of them in an arbitrarily large amount of copies.

The division rule $[a]_i \rightarrow [b]_i[c]_i$ can be applied over a cell i containing object a. The application of this rule divides this cell into two new cells with the same label. All the objects in the original cell are replicated and copied in each of the new cells, with the exception of the object a, which is replaced by the object b in the first new cell and by c in the second one. Since both new cells keep the same label as their father cell, they keep the same connections too. There is no connection between both new cells.

Rules are used as usual in the framework of membrane computing, that is, in a maximally parallel way (a universal clock is considered). In one step, each object in a cell can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e, in each step we apply a maximal set of rules. This way of applying rules has only one restriction: when a cell is divided, the division rule is the only one which is applied for that cell in that step; the objects inside that cell do not move in that step.

4 Recognizing Tissue P Systems with Cell Division

NP-completeness has been usually studied in the framework of *decision problems*. Let us recall that a decision problem is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X.

In order to study the computational efficiency, a special class of tissue P systems is introduced in [8]: *recognizing*[1] *tissue P systems*.

A recognizing tissue P system with cell division of degree $q \geq 1$ is a tuple $\Pi = (\Gamma, \Sigma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_{in}, i_0)$, where

- $(\Gamma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_0)$ is a tissue P system with cell division of degree $q \geq 1$ (as defined in the previous section).
- The working alphabet Γ has two distinguished objects `yes` and `no`, present in at least one copy in an initial multiset $w_1, \ldots, w_q$, but not present in $\mathcal{E}$.
- Σ is an (input) alphabet strictly contained in Γ.
- $i_{in} \in \{1, \ldots, q\}$ is the input cell.
- The output region i_0 is the environment.
- All computations halt.
- 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.

The computations of the system Π with input $w \in \Gamma^*$ start from a configuration of the form $(w_1, w_2, \ldots, w_{i_{in}} w, \ldots, w_q; \mathcal{E})$, that is, after adding the multiset w to the contents of the input cell i_{in}. We say that the multiset w is *recognized* by Π if and only if the object `yes` is sent to the environment, in the last step

[1] In [8] they were called *recognizer* tissue P systems.

of all its associated computations. We say that $\mathcal{C}$ is an accepting (resp. rejecting) computation if the object `yes` (resp. `no`) appears in the environment associated with the corresponding halting configuration of $\mathcal{C}$.

Definition 1. *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 recognizing tissue P systems with cell division if the following holds:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$.*
- *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 tissue P system $\Pi(n)$ a *confluent* condition, in the following sense: every computation of a system with the *same* input multiset must always give the *same* answer. The pair of functions (cod, s) are called a *polynomial encoding* of the problem in the family of P systems.

We denote by $\mathbf{PMC}_{TD}$ the set of all decision problems which can be solved by means of recognizing tissue P systems with cell division in polynomial time.

5 A Solution for the Subset Sum Problem

The Subset Sum problem is the following one: *Given a finite set A, a weight function, $w : A \to \mathbb{N}$, and a constant $k \in \mathbb{N}$, determine whether or not there exists a subset $B \subseteq A$ such that $w(B) = k$.*

Next, we shall prove that the Subset Sum problem can be solved in a linear time by a family of recognizing tissue P systems with cell division. We shall address the resolution via a brute force algorithm.

We shall use a tuple $(n, (w_1, \ldots, w_n), k)$ 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.

Theorem 1. `Subset Sum` $\in \mathbf{PMC}_{TD}$.

Proof. Let $A = \{a_1, \ldots, a_n\}$ be a finite set, $w : A \to \mathbb{N}$ a weight function, and $k \in \mathbb{N}$. Let $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a function defined by

$$g(n,k) = \frac{(n+k)(n+k+1)}{2} + n$$

This function is primitive recursive and bijective between $\mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$ and computable in polynomial time. Let us denote by $u = (n, (w_1, \ldots, w_n), k)$, where $w_i = w(a_i)$, $1 \leq i \leq n$, the given instance of the problem. We define the polynomially computable function $s(u) = g(n,k)$.

We shall provide a family of tissue P systems where each P system solves all the instances of the Subset Sum problem with the same size. The weight function w of the concrete instance will be provided via an input multiset determined via the function $cod(u) = \{\{v_i^{w_i} : 1 \leq i \leq n\}\} \cup \{\{q^k\}\}$.

Next, we shall provide a family $\mathbf{\Pi} = \{\Pi(g(n,k)) : n, k \in \mathbb{N}\}$ of recognizing tissue P systems with cell division which solve the Subset Sum problem in a linear time. For each $(n,k) \in \mathbb{N} \times \mathbb{N}$ we shall consider the system $\Pi(g(n,k)) = (\Gamma, \Sigma, \omega_1, \omega_2, \mathcal{R}, \mathcal{E}, i_{in}, i_0)$, where

- $\Gamma = \Sigma(n) \cup \{A_i, B_i : 1 \leq i \leq n\}$
  $\cup \{z_i : 1 \leq i \leq n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 11\}$
  $\cup \{c_i : 1 \leq i \leq n+1\}$
  $\cup \{d_i : 1 \leq i \leq \lceil \log n \rceil + \lceil \log(k+1) \rceil + 4\}$
  $\cup \{e_i : 1 \leq i \leq \lceil \log n \rceil + 1\}$
  $\cup \{B_{ij} : 1 \leq i \leq n \wedge 1 \leq j \leq \lceil \log(k+1) \rceil + 1\}$
  $\cup \{b, f_1, g_1, g_2, p, D, T, S, N, \mathtt{yes}, \mathtt{no}\}$
- $\Sigma = \{q\} \cup \{v_i : 1 \leq i \leq n\}$
- $\omega_1 = z_1\, b\, c_1\, \mathtt{yes}\, \mathtt{no}$
- $\omega_2 = DA_1 \cdots A_n$
- $\mathcal{R}$ is the following set of rules:
  1. *Division rules:*
     $r_{1,i} \equiv [A_i]_2 \to [B_i]_2[\lambda]_2$ for $i = 1, \ldots, n$
  2. *Communication rules:*
     $r_{2,i} \equiv (1, z_i / z_{i+1}, 0)$ for $i = 1, \ldots, n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 10$
     $r_{3,i} \equiv (1, c_i / c_{i+1}^2, 0)$ for $i = 1, \ldots, n$
     $r_4 \equiv (1, c_{n+1} / D, 2)$
     $r_5 \equiv (2, c_{n+1} / d_1 e_1, 0)$
     $r_{6,i} \equiv (2, e_i / e_{i+1}^2, 0)$ for $i = 1, \ldots, \lceil \log n \rceil$
     $r_{7,i} \equiv (2, d_i / d_{i+1}, 0)$ for $i = 1, \ldots, \lceil \log n \rceil + \lceil \log(k+1) \rceil + 3$
     $r_{8,i} \equiv (2, e_{\lceil \log n \rceil + 1} B_i / B_{i1}, 0)$ for $i = 1, \ldots, n$
     $r_{9,i,j} \equiv (2, B_{ij} / B_{ij+1}^2, 0)$ for $i = 1, \ldots, n$, $j = 1, \ldots, \lceil \log(k+1) \rceil$
     $r_{10,i} \equiv (2, B_{i \lceil \log(k+1) \rceil + 1} v_i / p, 0)$ for $i = 1, \ldots, n$
     $r_{11} \equiv (2, pq / \lambda, 0)$
     $r_{12} \equiv (2, d_{\lceil \log n \rceil + \lceil \log(k+1) \rceil + 4} / g_1 f_1, 0)$

$r_{13} \equiv (2, f_1 p/\lambda, 0)$
$r_{14} \equiv (2, f_1 q/\lambda, 0)$
$r_{15} \equiv (2, g_1/g_2, 0)$
$r_{16} \equiv (2, g_2 f_1/T, 0)$
$r_{17} \equiv (2, T/\lambda, 1)$
$r_{18} \equiv (1, bT/S, 0)$
$r_{19} \equiv (1, S\texttt{yes}/\lambda, 0)$
$r_{20} \equiv (1, z_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 11} b/N, 0)$
$r_{21} \equiv (1, N\texttt{no}/\lambda, 0)$

- $\mathcal{E} = \Gamma - \{\texttt{yes}, \texttt{no}\}$
- $i_{in} = 2$, is the input cell
- $i_0 = env$, is the output cell

The design is structured in the following stages:

- *Generation Stage*: The initial cell labelled by 2 is divided into two new cells; and the divisions are iterated n times until a cell has been produced for each possible candidate solution. Simultaneously to this process, two counters (c_i and z_i) evolve in the cell labelled by 1: the first one controls the step in which the communication between cells 2 and cell 1 starts and the second one will be useful in the output stage.
- *Pre–checking Stage*: When this stage starts, we have 2^n cells labelled by 2, each of them encoding a subset of the set A. In each such a cell, as many objects p as the weight of the corresponding subset will be produced. Recall that there are k copies of the object q in every cell labelled by 2 (since they were introduced as part of the input multiset).
- *Checking Stage*: In each cell labelled by 2, the number of copies of objects p and q are compared. The way to do that is removing from the cell in one step all possible pairs (p, q). After doing so, if some objects p or q remain in the cell, then the cell was not encoding a solution of the problem; otherwise, the weight of the subset of A encoded on the cell equals to k and hence it encodes a solution to the problem.
- *Output Stage*: The system sends to the environment the right answer according to the results of the previous stage:
  - *Answer yes:* After the checking stage, there is a cell labelled by 2 without objects p nor q. In this case, such a cell sends an object T to the cell 1. This object T causes the cell 1 to expel an object `yes` to the environment (see rules $r_{17} - r_{19}$).
  - *Answer no:* Every cell labelled by 2 contains some objects p or q. In this case, no object T arrives to the cell labelled by 1 and an object `no` is sent to the environment.

The proof will be concluded in Subsection 5.2. Before going on, let us informally present an overview of the computation.

5.1 An overview of the computation

First of all, we recall the polynomial encoding of the Subset Sum problem in the family $\mathbf{\Pi}$ constructed above. Let $u = (n, (w_1, \ldots, w_n), k)$ be an instance of the problem, $s(u) = g(n, k)$ and $cod(u) = \{\{v_i^{w_i} : 1 \leq i \leq n\}\} \cup \{\{q^k\}\}$.

Next, we describe informally how the recognizing tissue P system with cell division $\Pi(s(u))$ with input $cod(u)$ works. Let us start with the *generation stage*. Recall that if a division rule is triggered in a cell, then communication rules cannot be simultaneously applied to the contents of such cell. In this stage we have two parallel processes:

- On the one hand, in the cell labelled by 1 we have two counters: z_i, which will be used in the answer stage, and c_i, which will be multiplied until getting 2^n copies in exactly n steps.
- On the other hand, in the cells labelled by 2, the division rules are applied. For each object A_i (which codifies a member of the set A) we obtain two cells labelled by 2: one of them has an element B_i and the other does not.

When all divisions have been done, after n steps, we shall have 2^n cells with label 2 and each of them will contain the encoding of a subset of A. At this moment, the generation stage ends and the pre-checking stage begins.

For each cell 2, an object D is changed by a copy of the counter c_i. In this way, 2^n copies of D will appear in the cell 1, and in each cell labelled by 2 there will be an object c_{n+1}. The occurrence of such object c_{n+1} in the cells 2 will produce the apparition of two counters:

(a) The counter d_i lets the checking stage start, since it produces the apparition of the objects g_1 and f_1 after $\lceil \log n \rceil + \lceil \log(k+1) \rceil + 4$ steps.
(b) The counter e_i will be multiplied until obtaining $2^{\lceil \log n \rceil}$ copies, ensuring that at least n copies of $e_{\lceil \log n \rceil + 1}$ will be available in the step $n + \lceil \log n \rceil + 2$. Then, we trade objects $e_{\lceil \log n \rceil + 1}$ and B_i against B_{i1} for each element A_i in the subset associated with the cell.
After that, for each $1 \leq i \leq n$ we get $2^{\lceil \log(k+1) \rceil}$ copies of $B_{i\lceil \log(k+1) \rceil + 1}$, ensuring that at least $k+1$ copies will be available. Then for each element A_i in the subset associated with the cell we get $\min\{2^{\lceil \log(k+1) \rceil}, w(a_i)\}$ copies of object p, in the step $n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 5$.

The checking takes place in the step $n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 6$, when all pairs of objects p and q present in any cell labelled by 2 are sent to the environment. In this way, if the weight of the subset associated with a cell is equal to k, then no object p or q remains in this cell in the next step. Otherwise, if the encoding is not exactly of weight k, then at least one object p or q will remain in the cell. In the next step the answer stage starts. Two cases must be considered for each cell:

- If no object p or q remain in the cell, the object f_1 does not evolve, g_1 evolves to g_2, and in the step $n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 8$ the objects f_1 and g_2 are traded against T from the environment. In the next step T is sent to the cell 1,

and in the step $n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 10$, the objects T and b are sent to the environment traded by S. Finally, in the step $n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 11$ the objects S and `yes` are sent to the environment.

- If any object p or q remains in the cell, such object is sent to the environment together with the object f_1. This causes that the object b still remains in the cell 2 after the step $n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 10$. In this way, the objects b and $z_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 11}$ are traded by the object N with the environment, and in the step $n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 12$ the objects N and `no` are sent to the environment.

5.2 Verification

Next, we prove that the family of recognizing tissue P systems with cell division described above solves the Subset Sum problem in a linear time, according to Definition 1.

Before going on, let us remark that the defined family is *consistent*, i.e., all systems of the family are recognizing tissue P systems with cell division. By construction (type of rules and working alphabet) we can check that it is a family of tissue P systems with cell division. Moreover, we shall prove next that all computations of all systems in the family always halt and in the last step of computations either an object `yes` or `no` is sent to the environment.

Polynomial uniformity of the family

Next, we show that the family $\mathbf{\Pi} = \{\Pi(g(n,k)) \ : \ n,k \in \mathbb{N}\}$ defined in Theorem 1 is polynomially uniform by Turing machines. To this aim we are going to show that it is possible to build $\Pi(g(n,k))$ in polynomial time with respect to the size of u.

It is easy to check that the rules of a system $\Pi(g(n,k))$, with $n,k \in \mathbb{N}$ of the family are defined recursively from the values n and k. Besides, the necessary resources to build an element of the family are of polynomial order with respect to the same:

- Size of the alphabet: $(n+2) \cdot \lceil \log(k+1) \rceil + 6n + 3\lceil \log n \rceil + 28 \in O(n \cdot \log k)$
- Initial number of cells: $2 \in \theta(1)$.
- Initial number of objects: $n + 6 \in \theta(n)$.
- Number of rules: $(n+2) \cdot \lceil \log(k+1) \rceil + 5n + 3\lceil \log n \rceil + 26 \in O(n \cdot \log k)$
- Maximal length of a rule: $3 \in \theta(1)$.

Therefore, a deterministic Turing machine can build $\Pi(g(n,k))$ in a polynomial time with respect to n and k.

Notice that every instance $u = (n, (w_1, \ldots, w_n), k)$ is introduced in the initial configuration of its associated cellular system via an input multiset (i.e. an 1-ary representation) and hence, $|u| \in O(n+k)$ holds.

We would like to recall that the functions cod and s have been defined above for an instance $u = (n, (w_1, \ldots, w_n), k)$ of the Subset Sum problem as follows: $cod(u) = \{\{v_i^{w_i} : 1 \leq i \leq n\}\} \cup \{\{q^k\}\}$, and $s(u) = g(n, k)$, respectively. Both functions are computable in polynomial time and the pair (cod, s) is a polynomial encoding of $I_{\texttt{SubsetSum}}$ in $\mathbf{\Pi}$, since for each instance u of the Subset Sum problem we have that $cod(u)$ is an input multiset of the system $\Pi(s(u))$.

In order to settle the formal verification of the family of tissue P systems we shall prove that all the systems of the family are polynomially bounded, and also that they are sound and complete with respect to $(\texttt{SubsetSum}, cod, s)$.

Polynomial boundness of the family

In order to ensure that the system $\Pi(s(u))$ with input $cod(u)$ is polynomially (indeed, linearly) bounded, it suffices to find the moment in which the computation halts, or at least, an upper bound for it. As we shall show, the number of steps of the computations of any system of the family can always be bounded by a linear function. Nonetheless, we would like to stress that the amount of pre-computed resources for each instance u is polynomial in the size of the instance, since $cod(u)$ needs to be computed and $\Pi(s(u))$ needs to be built.

Proposition 1. *The family* $\mathbf{\Pi} = \{\Pi(g(n, k)) : n, k \in \mathbb{N}\}$ *is polynomially bounded with respect to* $(\texttt{SubsetSum}, cod, s)$.

Proof. (Sketch). We shall informally go through the stages of the computation in order to estimate a bound for the number of steps. The computation will be studied in more detail when addressing the soundness and completeness proof.

Let $u = (n, (w_1, \ldots, w_n), k)$ be an instance of the Subset Sum problem. We shall study what happens during the computation of the system $\Pi(s(u))$ with input $cod(u)$ which processes such instance in order to find the halting step, or at least, an upper bound for it.

First, the generation stage lasts exactly n steps, where all the divisions of the cells of the system are performed.

After that, the pre-checking stage starts with the rule r_4. In the following step the object d_1 arrives to all cells 2 and the counter d_i starts. At the step $n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 5$, the last element of the counter d_i is reached and the checking stage ends. In this way, in the $(n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 6)$-th step of the computation the checking takes place. Recall that only one step is needed for the checking, as rule r_{11} takes out in parallel all pairs (p, q) from all cells 2.

The last one is the answer stage. The longest case is obtained when the answer is negative. In this case there is one step where only the counter z_i is working since no element T has reached the cell 1. In the next step an object N is brought from the environment and, finally, in the $(n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 12)$-th step, the object `no` is sent to the environment.

Therefore, there exists a linear bound with respect to $(n + \log k)$ on the number of steps of the computation.

Soundness and Completeness of the family

In order to prove the soundness and completeness of the family $\mathbf{\Pi}$ with respect to $(\texttt{SubsetSum}, cod, s)$, we shall prove that given an instance u of the Subset Sum problem, the system $\Pi(s(u))$ with input $cod(u)$ sends out an object `yes` if and only if the answer to the problem for the considered instance u is affirmative, and the object `no` is sent out otherwise.

Proposition 2. *The family* $\mathbf{\Pi} = \{\Pi(g(n,k)) : n,k \in \mathbb{N}\}$ *is sound and complete with respect to* $(\texttt{SubsetSum}, cod, s)$.

Proof. In order to complete the proof we shall proceed through a number of auxiliary results.

Remark 1. Given a computation $\mathcal{C}$ we denote the configuration at the i-th step as $\mathcal{C}_i$. Moreover, $\mathcal{C}_i(1)$ will denote the multiset associated to cell 1 in such configuration.

We start with the generation stage (i.e., the n first steps of the computation). It consists of two parallel processes, each of them in one cell.

Lemma 1. *If* $\mathcal{C}$ *is an arbitrary computation of the system, then for all* j *such that* $0 \leq j \leq n$, $\mathcal{C}_j(1) = \{\{z_{j+1}, c_{j+1}^{2^j}, b, \texttt{yes}, \texttt{no}\}\}$ *holds.*

Proof. We shall reason by induction on j.

Base Case. We have $\mathcal{C}_0(1) = \{z_1, c_1, b, \texttt{yes}, \texttt{no}\}$, and thus the lemma holds for $j = 0$.

Case $j < n \rightarrow j+1$**.** Let j be such that $1 \leq j < n$ and we have, by inductive hypothesis, $\mathcal{C}_j(1) = \{\{z_{j+1}, c_{j+1}^{2^j}, b, \texttt{yes}, \texttt{no}\}\}$. In this configuration, only the rules $r_{3,j+1}$ and $r_{4,j+1}$ can be applied to cell 1, and therefore $\mathcal{C}_{j+1}(1) = \{\{z_{j+2}, c_{j+2}^{2^{(j+1)}}, b, \texttt{yes}, \texttt{no}\}\}$.

Lemma 2. *If* $\mathcal{C}$ *is an arbitrary computation of the system, then:*

1. *For each subset* $V \subseteq \{1, \ldots, n\}$ *there exists only one cell* 2 *in* $\mathcal{C}_n$ *whose multiset is* $cod(u) \cup \{\{D\}\} \cup \{\{B_i : i \in V\}\}$
2. *There exist exactly* 2^n *cells labelled by* 2 *in configuration* $\mathcal{C}_n$

Proof. It is clear that division rules cannot be applied in parallel over the same cell. At this point there is an intrinsic non-determinism of the system. We have to apply all the division rules, but the order is non-deterministically chosen.

At time 0, there are n division rules that can be applied. When we apply one of them to the cell labelled by 2, for example, $r_{1,i}$ $(1 \leq i \leq n)$ we eliminate the object A_i and will obtain two new cells, in the first one an object B_i will appear, but not in the second one. The remaining contents of the original cell will be in the two new cells as well.

In the following step, another division rule can be applied to the two cells labelled by 2. As described above, the object A_j that triggers the rule disappears,

and a new object B_j appear in one of the new cells (note that the two cells may chose different objects A_j). This process is repeated in all cells 2 for each division rule.

It is clear that by triggering a division rule $r_{1,j}$ we get two different cells. Only one of them containing B_j. On the other hand, each object A_i appears exactly once in their initial configuration. Therefore, after applying n division rules we obtain 2^n cells labelled by 2.

Moreover, let V be a subset of $\{1,\ldots,n\}$, if for each division rule $[A_i]_2 \rightarrow [B_i]_2[\lambda]_2$ we focus on the cell containing B_i only for $i \in V$ (we select the other cell otherwise), then after n division steps we will have a cell labelled by 2 such that B_j belongs to the cell if and only if $j \in V$, irrespectively of the order in which the division rules are applied.

Lemma 3. *If $\mathcal{C}$ is an arbitrary computation of the system, then for all i such that $1 \leq i \leq \lceil \log n \rceil + \lceil \log(k+1) \rceil + 7$, $\mathcal{C}_{n+i}(1) = \{\{z_{n+i+1}, D^{2^n}, b, \texttt{yes}, \texttt{no}\}\}$ holds*

Proof. In order to prove the lemma it suffices to observe the following:

- $\mathcal{C}_n(1) = \{\{z_{n+1}, c_{n+1}^{2^n}, b, \texttt{yes}, \texttt{no}\}\}$ holds from Lemma 1.
- There exist exactly 2^n cells labelled by 2 in configuration $\mathcal{C}_n$, each of them containing an object D (this follows from Lemma 2).
- In the next step of the computation, only rules r_4 and $r_{2,n+1}$ are applicable on cell 1, yielding $\mathcal{C}_{n+1}(1) = \{\{z_{n+1+1}, D^{2^n}, b, \texttt{yes}, \texttt{no}\}\}$
- During the rest of the checking stage, only rules of type $r_{2,i}$ are applicable on cell 1, and the result follows.

Lemma 4. *Let $\mathcal{C}$ be an arbitrary computation of the system. Then:*

- *For each subset $V \subseteq \{1,\ldots,n\}$ there exists only one cell 2 in $\mathcal{C}_{n+1}$ whose associated multiset is*

$$cod(u) \cup \{\{c_{n+1}\}\} \cup \{\{B_i : i \in V\}\}$$

- *There exist exactly 2^n cells labelled by 2 in configurations $\mathcal{C}_{n+i}$, for $1 \leq i \leq \lceil \log n \rceil + \lceil \log(k+1) \rceil + 12$*

Proof. $\mathcal{C}_{n+1}$ is obtained from $\mathcal{C}_n$ by the application of the rules r_4 and $r_{2,n+1}$ and hence, 2^n objects c_{n+1} in the cell 1 are traded against 2^n objects D from the cells 2 (one from each cell). Then $\mathcal{C}_{n+1}(1) = \{\{z_{n+2}, D^{2^n}, b, \texttt{yes}, \texttt{no}\}\}$ and for every $V \subseteq \{1,\ldots,n\}$ there exists only one cell 2 whose associated multiset is $cod(u) \cup \{\{c_{n+1}\}\} \cup \{\{B_i : i \in V\}\}$

Since no division rule has been applied in this step (actually, they will not be applied anymore along the computation), the number of cells 2 remains the same as in the previous configuration.

Lemma 5. *Let $\mathcal{C}$ be an arbitrary computation of the system. For each i ($1 \leq i \leq \lceil \log n \rceil + 1$) and for each $V \subseteq \{1,\ldots,n\}$ there exists only one cell 2 in $\mathcal{C}_{n+i+1}$ whose associated multiset is*

$$cod(u) \cup \{\{d_i, e_i^{2^{(i-1)}}\}\} \cup \{\{B_j : j \in V\}\}$$

Proof. We shall reason by induction on i.

Case $i = 1$. It suffices to note that r_5 is the only rule that can be applied on cells 2 in configuration $\mathcal{C}_{n+1}$, and the result follows from the previous Lemma.

Case $1 \leq i < \lceil \log n \rceil + 1 \rightarrow i + 1$.

Let i be such that $1 \leq i < \lceil \log n \rceil + 1$ and let us suppose that the result holds for i. Let V be an arbitrary subset of $\{1, \ldots, n\}$, then let us consider its associated cell whose contents are indicated by inductive hypothesis. The only rules applicable to this cell are $r_{6,i}$ and $r_{7,i}$, and therefore the multiset of such cell 2 in $\mathcal{C}_{n+i+2}$ will be

$$cod(u) \cup \{\{d_{i+1}, e_{i+1}^{2^i}\}\} \cup \{\{B_j \,:\, j \in V\}\},$$

as we wanted to prove.

Lemma 6. *Let $\mathcal{C}$ be an arbitrary computation of the system. For each l $(1 \leq l \leq \lceil \log(k+1) \rceil + 2)$ and for each $V \subseteq \{1, \ldots, n\}$ there exists only one cell 2 in $\mathcal{C}_{n+\lceil \log n \rceil + l + 2}$ whose associated multiset is*

$$cod(u) \cup \{\{d_{\lceil \log n \rceil + l + 1}, e_{\lceil \log n \rceil + 1}^{(2^{\lceil \log n \rceil} - |V|)}\}\} \cup \{\{B_{jl}^{2^{(l-1)}} \,:\, j \in V\}\}$$

Proof. We shall reason by induction on l.

Case $l = 1$. Let V be an arbitrary subset of $\{1, \ldots, n\}$, then from the previous Lemma it follows that there exists a cell whose associated multiset in $\mathcal{C}_{n+\lceil \log n \rceil + 2}$ is $cod(u) \cup \{\{d_{\lceil \log n \rceil + 1}, e_{\lceil \log n \rceil + 1}^{2^{\lceil \log n \rceil}}\}\} \cup \{\{B_j \,:\, j \in V\}\}$. The next configuration for this cell is obtained by applying rules $r_{7,\lceil \log n \rceil + 1}$ and $r_{8,j}$ for every $j \in V$:

- $r_{7,\lceil \log n \rceil + 1}$ allows the evolution of the counter d to $d_{\lceil \log n \rceil + 2}$ in each cell 2.
- Each $r_{8,j}$ allows the replacement of the objects B_j (together with a copy of $e_{\lceil \log n \rceil + 1}$) by B_{j1}.

and thus the result holds for $l = 1$.

Case $1 \leq l < \lceil \log(k+1) \rceil \rightarrow l + 1$.

Let l be such that $1 \leq l < \lceil \log n \rceil + 1$ and let us suppose that the result holds for l. That is, there exists only one cell 2 in $\mathcal{C}_{n+\lceil \log n \rceil + l + 3}$ with the multiset

$$cod(u) \cup \{\{d_{\lceil \log n \rceil + l + 1}, e_{\lceil \log n \rceil + 1}^{2^{\lceil \log n \rceil} - |V|}\}\} \cup \{\{B_{jl}^{2^{(l-1)}} \,:\, j \in V\}\}$$

In the next step, rules $r_{7,n+\lceil \log n \rceil + l + 3}$ and $r_{9,j,l}$, for every $j \in V$, will be applied on this cell, and thus the multiset of such cell 2 in $\mathcal{C}_{n+\lceil \log n \rceil + l + 1 + 3}$ will be

$$cod(u) \cup \{\{d_{\lceil \log n \rceil + l + 2}, e_{\lceil \log n \rceil + 1}^{2^{\lceil \log n \rceil} - |V|}\}\} \cup \{\{B_{j(l+1)}^{2^l} \,:\, j \in V\}\}$$

Lemma 7. *Let $\mathcal{C}$ be an arbitrary computation of the system. For each $V \subseteq \{1, \ldots, n\}$ there exists only one cell 2 in $\mathcal{C}_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 5}$ whose associated multiset is*

$$\{\{p^{\sum_{j \in V} w_j}, q^k\}\} \cup \{\{d_{\lceil \log n \rceil + \lceil \log(k+1) \rceil + 4}, e_{\lceil \log n \rceil + 1}^{2^{\lceil \log n \rceil} - |V|}\}\} \cup \{\{B_{j(\lceil \log(k+1) \rceil + 1)}^{2^{\lceil \log(k+1) \rceil} - w_j} \,:\, j \in V\}\}$$

Proof. Let V be an arbitrary subset of $\{1,\ldots,n\}$. From the previous Lemma (taking $l = \lceil \log(k+1) \rceil + 2$), it follows that in $\mathcal{C}_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 4}$ there is a cell 2 associated with this subset whose content is indicated above. In the next step, the following rules are applied:

- $r_{7,\lceil \log n \rceil + \lceil \log(k+1) \rceil + 3}$ allows the evolution of the counter $d_{\lceil \log n \rceil + \lceil \log(k+1) \rceil + 4}$.
- $r_{10,j}$, for every $j \in V$, allow the replacement of all the existing objects v_j (recall that there are $w(a_j)$ copies of v_j in $cod(u)$), together with objects $B_{j(\lceil \log(k+1) \rceil + 1)}$ against $\sum_{j \in V} w_j$ objects p.

Remark 2. From this moment on, for the sake of simplicity, given $V \subseteq \{1,\ldots,n\}$ we shall note by $w_V = \sum_{j \in V} w_j$, and we shall also note by $trash_V$ the following multiset:

$$\{\{e_{\lceil \log n \rceil + 1}^{2^{\lceil \log n \rceil} - |V|}\}\} \cup \{\{B_{j(\lceil \log(k+1) \rceil + 1)}^{2^{\lceil \log(k+1) \rceil} - w_j} : j \in V\}\}$$

Lemma 8. *Let $\mathcal{C}$ be an arbitrary computation of the system and let $V \subseteq \{1,\ldots,n\}$. We have the following:*

- *If $k = w_V$ then there exists only one cell 2 in $\mathcal{C}_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 6}$ whose associated multiset is*
$$\{\{f_1, g_1\}\} \cup trash_V$$
- *If $k < w_V$ then there exists only one cell 2 in $\mathcal{C}_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 6}$ whose associated multiset is*
$$\{\{p^{w_V - k}, f_1, g_1\}\} \cup trash_V$$
- *If $k > w_V$ then there exists only one cell 2 in $\mathcal{C}_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 6}$ whose associated multiset is*
$$\{\{q^{k - w_V}, f_1, g_1\}\} \cup trash_V$$

Proof. Let V be an arbitrary subset of $\{1,\ldots,n\}$. From the previous Lemma, it follows that in $\mathcal{C}_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 5}$ there is a cell 2 associated with this subset whose content is indicated above. In the next step, the following rules are applied:

- r_{11} sends all existing pairs (p, q) to the environment. So, if $k = w_V$ then all objects p and q will disappear of the cell. However, if $k < w_V$ then $w_V - k$ objects p will remain in the cell, and conversely if $k > w_V$ then $k - w_V$ objects q will remain in the cell.
- r_{12} trades the object $d_{\lceil \log n \rceil + \lceil \log(k+1) \rceil + 4}$ against the objects f_1, g_1.

Lemma 9. *Let $\mathcal{C}$ be an arbitrary computation of the system, and let V be a subset of $\{1,\ldots,n\}$.*

1. *For each V such that $w_V = k$, there exists one cell 2 such that:*
   a) its associated multiset in $\mathcal{C}_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 7}$ is $\{\{f_1, g_2\}\} \cup trash_V$
   b) its associated multiset in $\mathcal{C}_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 8}$ is $\{\{T\}\} \cup trash_V$

c) For each $i\,(1 \leq i \leq 4)$*, its associated multiset in* $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+i+8}$ *is* $trash_V$

2. *For each* V *such that* $w_V \neq k$*, there exists one cell* 2 *such that for each* $i\,(1 \leq i \leq 6)$ *its associated multiset in* $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+i+6}$ *is* $\{\{g_2, p^{w_V-k-1}\}\} \cup trash_V$ *or* $\{\{g_2, q^{k-w_V-1}\}\} \cup trash_V$

Proof. 1. From the previous Lemma, if $k = w_V$ then there exists only one cell 2 in $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+6}$ whose associated multiset is

$$\{\{f_1, g_1\}\} \cup trash_V$$

Then, by applying the rule r_{15} (no other rule can be applied in this cell) we obtain in the next step $\{\{f_1, g_2\}\} \cup trash_V$. After that, by applying the rule r_{16} we obtain $\{\{T\}\} \cup trash_V$. Finally, by applying the rule r_{17} the element T is sent to the cell labelled by 1. No more rules can be applied after this moment in the cells labelled by 2. Therefore, for $1 \leq i \leq 4$, the contents of the cell in $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+i+8}$ is the multiset $trash_V$.

2. From the previous Lemma, if $k < w_V$ then there exists only one cell 2 in $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+6}$ whose associated multiset is

$$\{\{p^{w_V-k}, f_1, g_1\}\} \cup trash_V$$

Then, by applying the rule r_{13} an object p is sent to the environment together with object f_1. In parallel, rule r_{15} trades g_1 against g_2. No more rules can be applied in the cell after this moment. Therefore, for $1 \leq i \leq 5$, the contents of the cell in $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+i+7}$ is the multiset $\{\{g_2, p^{w_V-k-1}\}\} \cup trash_V)$. Analogously for the case when $k > w_V$ (applying the rule r_{14} instead of r_{13}).

Lemma 10. *Let* $\mathcal{C}$ *be an arbitrary computation of the system. Let us suppose that there exists* $V \subseteq \{1, \ldots, n\}$ *such that* $w_V = k$*. Then*

(a) $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+9}(1) = \{\{z_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+10}, D^{2^n}, b, \texttt{yes}, \texttt{no}, T\}\}$
(b) $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+10}(1) = \{\{z_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+11}, D^{2^n}, S, \texttt{yes}, \texttt{no}\}\}$
(c) $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+11}(1) = \{\{z_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+11}, D^{2^n}, \texttt{no}\}\}$

Proof. The configuration of item (a) is obtained by the application of rules r_{17} and $r_{2,n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+9}$ to the previous configuration. Analogously, the configurations of items (b) and (c) are obtained by the application of rules r_{18} and r_{19} respectively.

Lemma 11. *Let* $\mathcal{C}$ *be an arbitrary computation of the system. Let us suppose that there does not exist any subset* $V \subseteq \{1, \ldots, n\}$ *such that* $w_V = k$*. Then*

(a) $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+9}(1) = \{\{z_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+10}, D^{2^n}, b, \texttt{yes}, \texttt{no}\}\}$
(b) $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+10}(1) = \{\{z_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+11}, D^{2^n}, b, \texttt{yes}, \texttt{no}\}\}$
(c) $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+11}(1) = \{\{D^{2^n}, N, \texttt{yes}, \texttt{no}\}\}$
(d) $\mathcal{C}_{n+\lceil \log n\rceil+\lceil \log(k+1)\rceil+12}(1) = \{\{D^{2^n}, \texttt{yes}\}\}$

Proof. The configuration of item (a) and (b) are obtained by the application of rules $r_{2,n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 9}$ and $r_{2,n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 10}$ to the previous configuration. Analogously, the configurations of items (c) and (d) are obtained by the application of rules r_{20} and r_{21} respectively.

5.3 Main Results

From the discussion in the previous sections and according to the definition of solvability given in Definition 1, we deduce the following result:

Theorem 1. `Subset Sum` $\in \mathbf{PMC}_{TD}$.

As a consequence of this result we have:

Theorem 2. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{TD}$.

Proof It suffices to make the following observations: the Subset Sum problem is **NP**-complete, `SubsetSum` $\in \mathbf{PMC}_{TD}$ and the class $\mathbf{PMC}_{TD}$ is stable under polynomial-time reduction, and also closed under complement.

6 Conclusions and Future Work

The physical limitations of current silicon-based hardware have been one of the triggers for the development of alternative models of computation (also known as *unconventional*). In particular, the scientific community is getting increasingly interested on computing models inspired by Nature. These new models abstract features of living entities and use them as inspiration for designing algorithms within new computing paradigms.

Membrane Computing is a new cross-disciplinary field of Natural Computing which has reached an important success in its short life. In these years many results have been presented related to the computational power of membrane devices, but up to now no implementation *in vivo* or *in vitro* has been carried out.

As the classical complexity classes **P** and **NP** are very likely to be different, the design of efficient solutions (in time) to **NP**-complete problems consequently needs to handle an exponential amount of resources. Cellular Computing with Membranes provides a framework where this trade-off between time and space is formalized in a natural way, getting inspiration from the way new cells are created (are *born*) from existing ones. Indeed, the mitosis process (cell division) is the motivation for the model of tissue P systems with cell division used in this paper. As the model allows all existing cells to be divided in parallel at every step, it follows directly that one can produce 2^n membranes in n steps. Using this ability, we have presented in this paper a solution to a numerical **NP**-complete problem using a family of recognizing tissue P systems with cell division.

More precisely, this paper deals with the design and formal verification of an *algorithm* to solve a well-known problem in an efficient and uniform way, and in

this sense it is a theoretical result, mainly related to computational complexity classes. We would like to stress that the result presented in this paper improves previous designs (for other problems) in two senses. On the one hand, the size of the rules is bounded by 3 and, on the other hand, the number of steps and the initial resources are of $O(\log k)$ order instead of being linearly dependent on k.

This is the first design of a solution to a numerical problem in this framework (up to our knowledge), and thus it may be useful as a template or guidance when addressing other numerical problems. Besides, the strategies that have been applied in the design presented in this paper can be also used when working on similar models. For instance, there is a promising new paradigm within Membrane Computing, namely Spiking Neural P systems, that is based on communication between neurons (recall that the inspiration of tissue P systems comes from communication and cooperation between cells in a tissue). Efficient resolution of hard problems has not yet been addressed in this new model, but it may in a near future. We would also like to mention as future work the development of software tools to simulate such computational processes, as the existing simulators for other membrane computing models have proved to be very useful as assistants for designing P systems and for understanding the way they work.

Acknowledgment

The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

1. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: A linear–time tissue P system based solution for the 3–coloring problem. *Theoretical Computer Science*, to appear.
2. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, F.J. Romero-Campero: A linear solution for QSAT with Membrane Creation. In *Membrane Computing. International Workshop WMC6, Vienna, Austria, 2005*, LNCS 3850, Springer, 2006, 241–252.
3. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón: A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. In *Computing and Combinatorics: 8th Annual International Conference, COCOON 2002, Singapore, 2002*, LNCS 2387, Springer, 2002, 290–299.
4. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón: Tissue P systems. *Theoretical Computer Science*, 296 (2003), 295–326.
5. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3 (2002), 295–305.
6. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
7. Gh. Păun: *Membrane Computing – An Introduction*. Springer, Berlin, 2002.

8. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez: Tissue P System with cell division. In *Second Brainstorming Week on Membrane Computing*, Sevilla, Report RGNC 01/2004, (2004), 380–386.
9. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: A polynomial complexity class in P systems using membrane division. In *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, (2003), 284–294.

Polarizationless P Systems with Active Membranes Working in the Minimally Parallel Mode

Rudolf Freund[1], Gheorghe Păun[2,3], Mario J. Pérez-Jiménez[3]

[1] Institute of Computer Languages
Vienna University of Technology
Favoritenstr. 9, Wien, Austria
rudi@emcc.at
[2] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania, and
george.paun@imar.ro, gpaun@us.es
[3] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

Summary. We investigate the computing power and the efficiency of P systems with active membranes without polarizations, working in the minimally parallel mode. We prove that such systems are computationally complete and able to solve **NP**-complete problems even when the rules are of a restricted form, e.g., for establishing computational completeness we only need rules handling single objects and no division of non-elementary membranes is used.

1 Introduction

P systems with active membranes basically use five types of rules: (a) evolution rules, by which a single object evolves to a multiset of objects, (b) send-in, and (c) send-out rules, by which an object is introduced in or expelled from a membrane, maybe modified during this operation into another object, (d) dissolution rules, by which a membrane is dissolved, under the influence of an object, which may be modified into another object by this operation, and (e) membrane division rules; this last type of rules can be used both for elementary and non-elementary membranes, or only for elementary membranes. As introduced in [10], all these types of rules also use polarizations for membranes, "electrical charges" $+, -, 0$, controlling the application of the rules.

Systems with rules of types (a), (b), (c) were shown to be equivalent in computational power with Turing machines [11], even when using only two polarizations

[4], while P systems using all types of rules were shown to be universal even without using polarizations [1]. Another important class of results concerns the possibility of using P systems with active membranes to provide polynomial solutions to computationally hard problems. Several papers have shown that both decision and numerical **NP**-complete problems can be solved in a polynomial time (often, even linear time) by means of P systems with three polarizations [11], [13], then the number of polarizations was decreased to two [3], [4]. The systems constructed in these solutions use only division of elementary membranes. At the price of using division also for non-elementary membranes, the polarizations can be completely removed, [2].

All papers mentioned above apply the rules in the maximally parallel mode: in each step, the assignment of objects to the rules in the chosen multiset of rules to be applied in parallel is maximal, i.e., no further rule could be added to this chosen multiset of rules in such a way that the rules in the resulting extended multiset still could be applied in parallel. Recently, [5] a more relaxed strategy of using the rules was introduced, the so-called minimal parallelism: in each step, the assignment of objects to the rules in the chosen multiset of rules to be applied in parallel does not allow for extending it by any rule out of a set of rules from which no rule has been chosen so far for this multiset of rules. This introduces an additional degree of non-determinism in the system evolution, but still computational completeness and polynomial solutions to `SAT` were obtained in the new framework by using P systems with active membranes, with three polarizations and division of only elementary membranes.

In this paper we continue the study of P systems working in the minimally parallel way, and we prove that the polarizations can be avoided, at the price of using all five types of rules for computational completeness and the division of non-elementary membranes for computational efficiency. Moreover, in the proof for establishing computational completeness we restrict the form of the rules to handling only single objects in all types of rules (we call this the one-normal form for P systems with active membranes).

2 Prerequisites

We suppose that the reader is familiar with the basic elements of Turing computability [6], and of membrane computing [11]. We here, in a rather informal way, introduce only the necessary notions and notation.

For an alphabet A, by A^* we denote the set of all strings of symbols from A including the empty string λ. A *multiset* over an alphabet A is a mapping from A to the set of natural numbers; we represent a multiset by a string from A^*, where the number of occurrences of a symbol $a \in A$ in a string w represents the multiplicity of a in the multiset represented by w (hence, all strings obtained by permuting symbols in the string w represent the same multiset). The family of Turing-computable sets of natural numbers is denoted by NRE (with RE coming from "recursively enumerable").

In our proofs showing computational completeness we use the characterization of NRE by means of *register machines*. Such a device consists of a given number of registers, each of which can hold an arbitrarily large natural number, and a set of labeled instructions which specify how the numbers stored in registers can change, and which instruction should follow after the instruction just carried out. There are three types of instructions:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ add 1 to register r, and then go to one of the instructions labeled by l_j and l_k, non-deterministically chosen;
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ if register r is non-empty (non-zero), then subtract 1 from it and go to the instruction labeled by l_j, otherwise go to the instruction labeled by l_k;
- $l_h : \texttt{HALT}$ the halt instruction.

A *register machine* is a construct $M = (n, B, l_0, l_h, I)$, where n is the number of registers, B is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to `HALT` only), and I is the set of instructions. Each label of B labels only one instruction from I, thus precisely identifying it. A register machine M generates a set $N(M)$ of numbers in the following way: having initially all registers empty (i.e., storing the number zero), start with the instruction labeled by l_0, and proceed to apply instructions as indicated by the labels and by the contents of registers. If we reach the halt instruction, then the number stored at that time in register 1 is said to be computed by M, and therefore it is introduced in $N(M)$. Since we have a non-deterministic choice in the continuation of the computation in the case of ADD instructions, $N(M)$ can be an infinite set. It is known (see [8]) that in this way we can compute all the sets of numbers which are Turing-computable, even using register machines with only three registers as well as registers two and three being empty whenever the register machine halts.

A register machine can also work in the accepting mode. The number to be accepted is introduced in register 1, with all other registers being empty. We start computing with the instruction labeled by l_0; if the computation halts, then the number is accepted (the contents of the registers in the halting configuration do not matter). We still denote by $N(M)$ the set of numbers accepted by a register machine M. In the accepting case, we can request the register machines to be deterministic, namely with all instructions $l_i : (\texttt{ADD}, l_j, l_k)$ having $l_j = l_k$. It is known that deterministic accepting register machines characterize NRE even with only three registers and all registers being empty whenever the register machine halts.

3 P Systems with Active Membranes

We first introduce the P systems with active membranes in the general form, and then we describe the restricted version we investigate in this paper.

A *P system with active membranes*, of the initial degree $n \geq 1$, is a construct of the form

$$\Pi = (O, H, \mu, w_1, \ldots, w_n, R, h_o),$$

where:

1. O is the alphabet of *objects*;
2. H is a finite set of *labels* for membranes;
3. μ is a *membrane structure*, consisting of n membranes having initially neutral polarizations, labeled (not necessarily in a one-to-one manner) with elements of H;
4. $w_1, \ldots, w_n$ are strings over O, describing the *multisets of objects* placed in the n initial regions of μ;
5. R is a finite set of *developmental rules*, of the following forms:
   (a) $[_h a \to v]_h^e$,
   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);
   (b) $a[_h\,]_h^{e_1} \to [_h b]_h^{e_2}$,
   for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$
   (*in* communication rules; a possibly modified object is introduced in a membrane; the polarization of the membrane can also be modified, but not its label);
   (c) $[_h a\,]_h^{e_1} \to b[_h\,]_h^{e_2}$,
   for $h \in H, e_1, e_2 \in \{+, -, 0\}, a, b \in O$
   (*out* communication rules; a possibly modified object is sent out of the membrane; the polarization of the membrane can also be modified, but not its label);
   (d) $[_h a\,]_h^e \to b$,
   for $h \in H, e \in \{+, -, 0\}, a, b \in O$
   (*dissolving* rules; in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);
   (e) $[_h a\,]_h^{e_1} \to [_h b\,]_h^{e_2} [_h c\,]_h^{e_3}$,
   for $h \in H, e_1, e_2, e_3 \in \{+, -, 0\}, a, b, c \in O$
   (*division* rules for elementary or non-elementary membranes; in reaction with an object, the membrane with label h is divided into two membranes with the same label, possibly of different polarizations; the object specified in the rule is replaced in the two new membranes by possibly new objects; the remaining objects may evolve in the same step by rules of type (a) and the result of this evolution is duplicated in the two new membranes; if membrane h contains other membranes, then they may evolve at the same time by rules of any type and the result of their evolution is duplicated in the two new membranes);

6. $h_o \in H$ or $h_o = env$ indicates the output region.

In the maximally parallel mode, in each step (a global clock is assumed) we apply multisets of rules in such a way that no further rule can be applied to the remaining objects or membranes. In each step, each object and each membrane can be involved in only one rule. Because this way of using the rules is standard in membrane computing, we do not give further details, and we present only the minimally parallel mode, which here means the following:

All the rules of any type involving a membrane h constitute the set R_h (this means all the rules of type (a) of the form $[_h a \to v]_h^e$, all the rules of type (b) of the form $a[_h\,]_h^{e_1} \to [_h a]_h^{e_2}$, and all the rules of types (c) – (e) of the form $[_h a]_h^e \to z$, with the same h, constitute the set R_h). Moreover, if a membrane h appears several times in a given configuration of the system, then for each occurrence of the membrane we consider a different set R_h; this means that we identify the ith copy of membrane h with the pair (h, i), and we consider the set of rules $R_{h,i} = R_h$. Then, in each step, we choose a multiset of rules taken from the sets $R_{h,i}$, $h \in H$, in such a way that after having assigned objects to all the rules in this multiset, no rule from any of the sets $R_{h,i}$ from which no rule has been taken so far, could be used in addition.

Whenever relevant, in order to make visible the sets of rules, we write explicitly the sets R_h, $h \in H$, instead of the global set R.

Of course, as usual for P systems with active membranes, each membrane and each object can be involved in only one rule, and the choice of rules to be used and of objects and membranes to evolve is done in a non-deterministic way. We should note that for rules of type (a) the membrane is not considered to be involved: when applying $[_h a \to v]_h$, the object a cannot be used by other rules, but the membrane h can be used by any number of rules of type (a) as well as by one rule of types (b) – (e). In each step, the use of rules is done in the bottom-up manner (first the inner objects and membranes evolve, and the result is duplicated if any surrounding membrane is divided).

A halting computation provides a result given by the number of objects present in region h_o at the end of the computation; this is a region of the system if $h_o \in H$ (and in this case, for a computation to be successful, exactly one membrane with label h_o should be present in the halting configuration), or it is the environment if $h_o = env$.

We shall also consider the following normal form for P systems with active membranes: A system Π is said to be in the *one-normal form* if the membranes have no polarization (it is the same as saying that always all membranes have the same polarization, say 0, which therefore is irrelevant and thus omitted) and the rules are of the forms $[_h a \to b]_h$, $a[_h\,]_h \to [_h b]_h$, $[_h a\,]_h \to b[_h\,]_h$, $[_h a\,]_h \to b$, and $[_h a\,]_h \to [_h b\,]_h [_h c\,]_h$, for $a, b, c \in O$, such that $a \neq b, a \neq c, b \neq c$. Note that the labels of membranes are never changed.

The set of numbers generated in the minimally parallel way by a system Π is denoted by $N_{min}(\Pi)$. The family of sets $N_{min}(\Pi)$, generated by systems with

rules of the non-restricted form, having initially at most n_1 membranes and using configurations with at most n_2 membranes during any computation is denoted by $N_{min}OP_{n_1,n_2}((a),(b),(c),(d),(e))$; when a type of rules is not used, it is not mentioned in the notation. If any of the parameters n_1, n_2 is not bounded, then it is replaced by $*$. If the systems do not use polarizations for membranes, then we write $(a_0),(b_0),(c_0),(d_0),(e_0)$ instead of $(a),(b),(c),(d),(e)$. When the system Π is in the one-normal form, then we write $N_{min}OP_{n_1,n_2}((a_1),(b_1),(c_1),(d_1),(e_1))$.

When considering families of numbers generated by P systems with active membrane working in the maximally parallel mode, the subscript min is replaced by max in the previous notations. For precise definitions, we refer to [11] and to the papers mentioned below.

4 Computational Completeness

The following results are well known:

Theorem 1. (i) $N_{max}OP_{3,3}((a),(b),(c)) = NRE$, *[11].*
(ii) $N_{max}OP_{*,*}((a_0),(b_0),(c_0),(d_0),(e_0)) = NRE$, *[1].*
(iii) $N_{min}OP_{3,3}((a),(b),(c)) = NRE$, *[5].*

In turn, the following inclusions follow directly from the definitions:

Lemma 1. $N_{mode}OP_{n_1,n_2}((a_1),(b_1),(c_1),(d_1),(e_1)) \subseteq$
$N_{mode}OP_{n_1,n_2}((a_0),(b_0),(c_0),(d_0),(e_0)) \subseteq$
$N_{mode}OP_{n_1,n_2}((a),(b),(c),(d),(e))$,
for all $n_1, n_2 \geq 1$, *mode* $\in \{max, min\}$.

We now improve the equalities from Theorem 1 in certain respects, starting with proving the computational completeness of P systems with active membranes in the one-normal form when working in the maximally parallel mode, and then we extend this result to the minimal parallelism.

Theorem 2. $N_{max}OP_{n_1,*}((a_1),(b_1),(c_1),(d_1),(e_1)) = NRE$, *for all* $n_1 \geq 5$.

Proof. We only prove the inclusion

$$NRE \subseteq N_{max}OP_{5,*}((a_1),(b_1),(c_1),(d_1),(e_1)).$$

Let us consider a register machine $M = (3, B, l_0, l_h, I)$ generating an arbitrary set $N(M) \in NRE$. We then construct the P system

$$\Pi = (O, H, \mu, w_s, w_h, w_1, w_2, w_3, R, env),$$

of the initial degree 5, with

$$
\begin{aligned}
O &= \{d_i \mid 0 \le i \le 5\} \cup \{g, \#, \#', p, p', p'', c, c', c''\} \cup B \cup \{l' \mid l \in B\} \\
&\quad \cup \{l_{iu} \mid l_i \text{ is the label of an ADD instruction in } I, 1 \le u \le 4\} \\
&\quad \cup \{l_{iu0} \mid l_i \text{ is the label of a SUB instruction in } I, 1 \le u \le 4\} \\
&\quad \cup \{l_{iu+} \mid l_i \text{ is the label of a SUB instruction in } I, 1 \le u \le 6\}, \\
H &= \{s, h, 1, 2, 3\}, \\
\mu &= [_s[_1\,]_1\,[_2\,]_2\,[_3\,]_3[_h\,]_h]_s, \\
w_s &= l_0 d_0, w_\alpha = \lambda, \text{ for all } \alpha \in H - \{s\},
\end{aligned}
$$

and with the rules constructed as described below.

The value stored in a register $r = 1, 2, 3$ of M, in Π is represented by the number of copies of membranes with label r plus one (if the value of the register r is k, then we have $k + 1$ membranes with label r). The membrane with label h is auxiliary, it is used for controlling the correct simulation of instructions of M by computations in Π. Each step of a computation in M, i.e., using an ADD or a SUB instruction, corresponds to six steps of a computation in Π. We start with all membranes being empty, except for the skin region, which contains the initial label l_0 of M and the auxiliary object d_0. If the computation in M halts, that is, the object l_h appears in the skin region of Π, then we pass to producing one object c for each membrane with label 1 present in the system, except one; in this way, the number of copies of c sent to the environment represents the correct result of the computation in M.

We first indicate the rules used in each of the six steps of simulating instructions ADD and SUB, and then we present the rules for producing the result of the computation; in each configuration, only the membranes and the objects relevant for the simulation of the respective instruction are specified. In particular, we ignore the "garbage" object g, because once introduced it remains idle for the whole computation.

The **simulation of an instruction** $l_i : (\texttt{ADD}(r), l_j, l_k)$ uses the rules from Table 1.

Table 1. The simulation of an ADD instruction

Step	Main rules	Auxiliary rules	Configuration
–	–	–	$[_s l_i d_0\,[_r\,]_r \cdots\, [_h\,]_h]_s$
1	$l_i[_r\,]_r \to [_r l_i']_r$	$[_s d_0 \to d_1]_s$	$[_s d_1\,[_r l_i']_r \cdots\, [_h\,]_h]_s$
2	$[_r l_i']_r \to [_r l_{i1}]_r [_r g]_r$	$[_s d_1 \to d_2]_s$	$[_s d_2\,[_r l_{i1}]_r [_r g]_r \cdots\, [_h\,]_h]_s$
3	$[_r l_{i1}]_r \to l_{i2}[_r\,]_r$	$d_2[_h\,]_h \to [_h d_3]_h$	$[_s l_{i2}\,[_r\,]_r [_r\,]_r \cdots\, [_h d_3]_h]_s$
4	$[_s l_{i2} \to l_{i3}]_s$	$[_h d_3]_h \to d_4[_h\,]_h$	$[_s l_{i3} d_4[_r\,]_r [_r\,]_r \cdots\, [_h\,]_h]_s$
5	$[_s l_{i3} \to l_{i4}]_s$	$[_s d_4 \to d_5]_s$	$[_s l_{i4} d_5[_r\,]_r [_r\,]_r \cdots\, [_h\,]_h]_s$
6	$[_s l_{i4} \to l_t]_s, t \in \{j, k\}$	$[_s d_5 \to d_0]_s$	$[_s l_t d_0[_r\,]_r [_r\,]_r \cdots\, [_h\,]_h]_s$

The label object l_i enters into the correct membrane r (even if the register r is empty, there is at least one membrane with label r) and in the next step divides it. The object l_{i2} exits the newly produced membrane, but g remains inside; l_{i2} will evolve three further steps, just to synchronize with the evolution of the auxiliary objects $d_u, u \geq 0$, and the auxiliary membrane h, so that in the sixth step we end with the label l_j or l_k of the next instruction to be simulated present in the skin membrane, together with d_0; note that the number of copies of membrane r was increased by one.

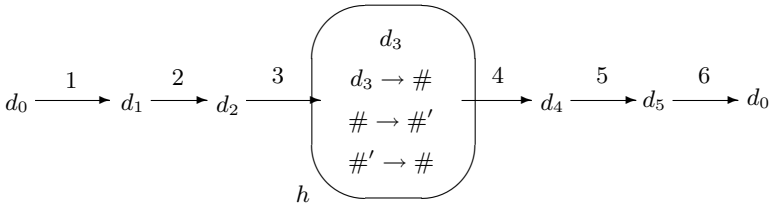


Fig. 1. The evolution of auxiliary objects

The auxiliary objects $d_u, u \geq 0$, and the auxiliary membrane h are used in the simulation of SUB instructions, as we will see immediately; in step 4, there also are other rules to be used in membrane h, introducing the trap-object #, but using such rules will make the computation never halt, hence, they will not produce an unwanted result.

The evolution of objects $d_u, u \geq 0$, is represented graphically in Figure 1; membrane h is only used in step 3, to send an object inside, and in the next step, for sending an object out of it. On each arrow we indicate the step, according to Table 1; the steps made by using rules of types (b), (c) are pointed out by drawing the respective arrows crossing the membranes, thus suggesting the *in/out* actions.

The way the "ADD module" works is suggested in Figure 2; the division operation from step 2 is indicated by a branching arrow.

The **simulation of an instruction** $l_i : (\texttt{SUB}(r), l_j, l_k)$ is done with the help of the auxiliary membrane h. The object l_i enters a membrane r (there is at least one copy of it) and divides it, and on this occasion makes a non-deterministic choice between trying to continue as having register r non-empty or as having it empty. If the guess has been correct, then the correct action is done (decrementing the register in the first case, doing nothing in the second case) and the correct next label is introduced, i.e., l_j or l_k, respectively. If the guess has not been correct, then the trap object # is introduced. For all membranes x of the system we consider the rules $[_x \# \to \#']_x$, $[_x \#' \to \#]_x$, hence, the appearance of # will make the computation last forever.

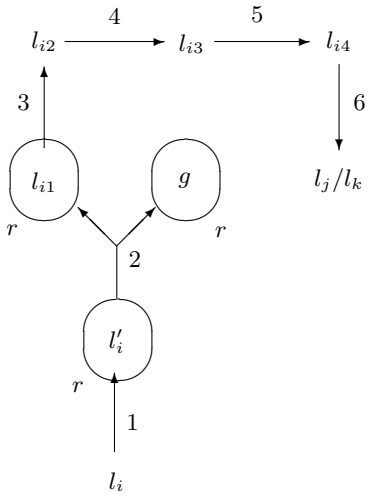


Fig. 2. The work of the ADD module

In Table 2 we present the rules used in the case of guessing that the register r is not empty. Like in the case of simulating an ADD instruction, the auxiliary objects and membranes do not play any role in this case.

Table 2. The simulation of a SUB instruction, guessing that register r is not empty

Step	Main rules	Auxiliary rules	Configuration
–	–	–	$[_s l_i d_0\ [_r\ ]_r \cdots [_h\ ]_h]_s$
1	$l_i[_r\ ]_r \to [_r l'_i]_r$	$[_s d_0 \to d_1]_s$	$[_s d_1\ [_r l'_i]_r \cdots [_h\ ]_h]_s$
2	$[_r l'_i]_r \to [_r l_{i1+}]_r [_r l_{i2+}]_r$	$[_s d_1 \to d_2]_s$	$[_s d_2 [_r l_{i1+}]_r [_r l_{i2+}]_r \cdots [_h\ ]_h]_s$
3	$[_r l_{i1+}]_r \to l_{i3+}$ $[_r l_{i2+}]_r \to l_{i4+}[_r\ ]_r$	$d_2[_h\ ]_h \to [_h d_3]_h$	$[_s l_{i3+} l_{i4+} [_r\ ]_r \cdots [_h d_3]_h]_s$
4	$l_{i3+}[_r\ ]_r \to [_r g]_r$ $l_{i4+}[_r\ ]_r \to [_r l_{i5+}]_r$ or $[_s l_{i3+} \to \#]_s, [_s l_{i4+} \to \#]_s$	$[_h d_3]_h \to d_4[_h\ ]_h$	$[_s d_4 [_r l_{i5+}]_r [_r g]_r \cdots [_h\ ]_h]_s$ $[_s d_4 \# [_r\ ]_r \cdots [_h\ ]_h]_s$
5	$[_r l_{i5+}]_r \to l_{i6+}$	$[_s d_4 \to d_5]_s$	$[_s d_5 l_{i6+} [_r\ ]_r \cdots [_h\ ]_h]_s$
7	$[_s l_{i6+} \to l_j]_s$	$[_s d_5 \to d_0]_s$	$[_s l_j d_0 [_r\ ]_r \cdots [_h\ ]_h]_s$

In step 2 we divide the membrane r containing the object l'_i and the objects l_{i1+}, l_{i2+} are introduced in the two new membranes. One of them is immediately dissolved, thus the number of copies of membrane r remains unchanged; the objects l_{i3+}, l_{i4+} are introduced in this step. In the next step (the fourth one of the simulation), objects l_{i3+}, l_{i4+} look for membranes r in the skin region. If both of them find such membranes – and this is the correct/desired continuation – then both of them enter such membranes; l_{i3+} becomes g and l_{i4+} becomes l_{i5+}. If only

one of them finds a membrane r, then the other one has to evolve to object # in the skin membrane and the computation never halts.

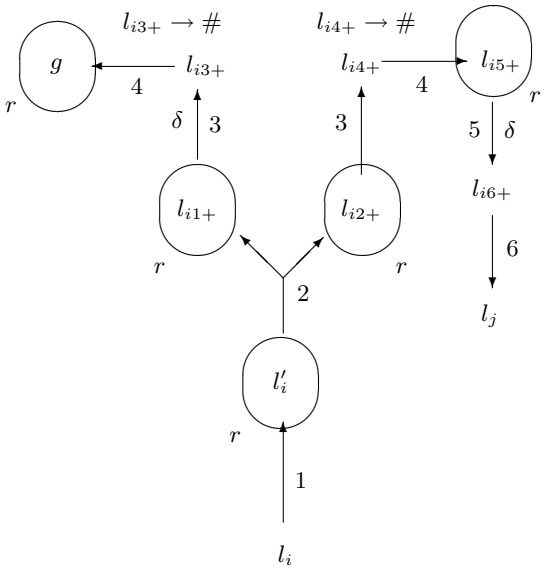


Fig. 3. The simulation of a SUB instruction when guessing that the register is non-empty

If we had enough membranes r and l_{i3+}, l_{i4+} went there, then in the next step l_{i5+} exits and is changed to l_{i6+}. In the sixth step, l_{i6+} becomes l_j (simultaneously with introducing d_0), and this completes the simulation. Thus, the computation continues without having # present in the system if and only if the guess made in step 2 has been correct, i.e., if register r has been non-empty. Because in step 5 a membrane r was dissolved, the number of these membranes was decreased by one, and this corresponds to subtracting one from register r.

Figure 3 indicates the evolution of objects l_{iu+} in the six steps mentioned in Table 2; when a dissolving operation is used, this is indicated by writing δ near the respective arrow.

However, in step 2, instead of the rule $[_r l'_i]_r \to [_r l_{i1+}]_r [_r l_{i2+}]_r$ we can use the rule $[_r l'_i]_r \to [_r l_{i10}]_r [_r l_{i20}]_r$, with the intention to simulate the subtract instruction in the case when the register r is empty – that is, only one membrane with label r is present in the system. The rules used in the six steps of the simulation are given in Table 3.

In this case, the auxiliary objects $d_u, u \geq 0$, and the auxiliary membrane h play an essential role.

Table 3. The simulation of a SUB instruction, guessing that register r is empty

Step	Main rules	Auxiliary rules	Configuration
–	–	–	$[_s l_i d_0\ [_r\ ]_r \cdots [_h\ ]_h]_s$
1	$l_i[_r\ ]_r \to [_r l_i']_r$	$[_s d_0 \to d_1]_s$	$[_s d_1\ [_r l_i']_r \cdots [_h\ ]_h]_s$
2	$[_r l_i']_r \to [_r l_{i10}]_r [_r l_{i20}]_r$	$[_s d_1 \to d_2]_s$	$[_s d_2\ [_r l_{i10}]_r [_r l_{i20}]_r \cdots [_h\ ]_h]_s$
3	$[_r l_{i10}]_r \to l_{i30}$ $[_r l_{i20}]_r \to l_{i40}[_r\ ]_r$	$d_2[_h\ ]_h \to [_h d_3]_h$	$[_s l_{i30} l_{i40}\ [_r\ ]_r \cdots [_h d_3]_h]_s$
4	$[_r l_{i30}]_r \to [_r g]_r$, l_{i40} waits or $l_{i40}[_r\ ]_r \to [_r \#]_r$ or $l_{i40}[_h\ ]_h \to [_h l_k']_h$	$[_h d_3]_h \to d_4[_h\ ]_h$ $[_h d_3 \to \#]_h$	$[_s l_{i40} d_4 [_r g]_r \cdots [_h\ ]_h]_s$ $[_s d_4 [_r \#]_r \cdots [_h\ ]_h]_s$ $[_s [_r g]_r \cdots [_h l_k' \#]_h]_s$
5	$l_{i40}[_h\ ]_h \to [_h l_k']_h$	$[_s d_4 \to d_5]_s$	$[_s d_5 [_r\ ]_r \cdots [_h l_k']_h]_s$
6	$[_h l_k']_h \to l_k[_h\ ]_h$	$[_s d_5 \to d_0]_s$	$[_s l_k d_0 [_r\ ]_r \cdots [_h\ ]_h]_s$

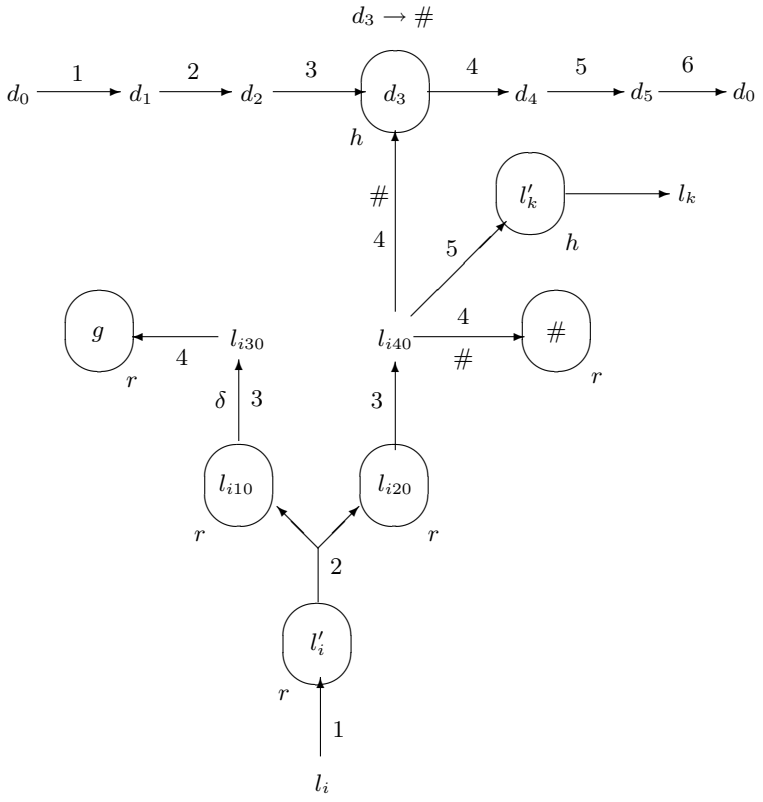


Fig. 4. The simulation of a SUB instruction when guessing that the register is empty

Until step 3 we proceed exactly as above, but now we introduce the objects l_{i30}, l_{i40}. The first one can enter the available membrane r, there evolving to g. If

there is a second membrane r, i.e., if the guess has been incorrect, then the rule $l_{i40}[_r\ ]_r \to [_r\#]_r$ should be used simultaneously (step 4), and the computation never ends. If there is no second membrane r, then in step 4 l_{i40}, can also enter membrane h, but then the trap object is produced here by the rule $[_h d_3 \to \#]_h$. The only way not to introduce the object # is (i) not to have a second membrane r, (ii) to use the rule $[_h d_3]_h \to d_4[_h\ ]_h$, thus preventing the use of the rule $l_{i40}[_h\ ]_h \to [_h l'_k]_h$, and (ii) not sending l_{i40} to the unique membrane r. This means that during step 4 the object l_{i40} should wait unchanged in the skin region.

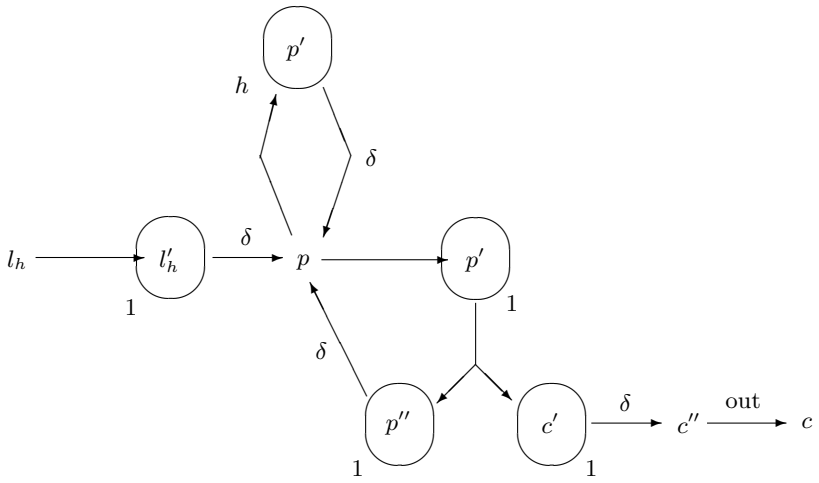


Fig. 5. The end of computations

In the next step, l_{i40} can enter membrane h (or the unique membrane r, but it becomes # there). In the next step, l_k is released from membrane h, at the same time with producing d_0, hence, the simulation is completed correctly and the system can pass to simulating another instruction.

The six steps of the computation are shown in Figure 4, this time with the evolution of the auxiliary objects being indicated, too. The arrows marked with # correspond to moves which are not desired.

For both guesses, the simulation of the SUB instruction works correctly, and the process can be iterated.

If the computation in M halts, i.e., if l_h is reached, i.e., if the object l_h is introduced in the skin region, we can start to use the following rules:

$$l_h[_1\ ]_1 \to [_1 l'_h]_1,$$
$$[_1 l'_h]_1 \to p,$$
$$p[_1\ ]_1 \to [_1 p']_1,$$
$$[_1 p']_1 \to [_1 p'']_1 [_1 c']_1,$$

$$[_1 p'']_1 \to p,$$
$$[_1 c']_1 \to c'',$$
$$[_s c'']_s \to c[_s\ ]_s,$$
$$p[_h\ ]_h \to [_h p']_h,$$
$$[_h p']_h \to p.$$

The object l_h dissolves one membrane with label 1 (thus, the remaining membranes with this label are now as many as the value of register 1 of M), and gets transformed into p. This object enters each membrane with label 1, divides it, reproduces itself (after passing through p' and p'') and also produces a copy of the object c''; this happens when dissolving the two membranes with label 1 obtained by division (rule $[_1 p']_1 \to [_1 p'']_1 [_1 c']_1$), hence, the number of copies of the membrane with label 1 has decreased by one. The object c'' immediately exits the system, thereby being changed into c.

At any time, the object p can also enter membrane h and then dissolves it. The computation can stop only after having dissolved all membranes with label 1 (i.e., a corresponding number of copies of c has been sent out) and after having dissolved the membrane h, hence, the evolution of objects $d_u, u \geq 0$, also stops.

The function of this final module is described in Figure 5.

Consequently, $N(M) = N_{max}(\Pi)$, and this concludes the proof. □

Theorem 3 gives the same result as Theorem 3 from [1], with the additional constraint to have evolution rules with exactly one object on the right-hand side (neither producing more objects, nor erasing objects, as it is the case in [1]).

The previous proof can easily be changed in order to obtain the computational completeness of P systems in the one-normal form also in the case of minimal parallelism. Specifically, we can introduce additional membranes in order to avoid having two or more evolution rules to be applied in the same region or one or more evolution rules and one rule of types (b) – (e) which involve the same membrane (these are the only situations where the minimal parallelism does not correspond to the maximal parallelism; note that all rules of types (b) – (e) are associated with membranes which are involved in the use of rules, hence, no two rules of these types can be applied for the same membrane).

In the previous construction, situations as above which must be avoided are the following ones:

- Rules $[_s l_{i3+} \to \#]_s$ and $[_s l_{i4+} \to \#]_s$, in step 4 of the simulation of SUB for the guess of a non-empty register. However, no other rule is applicable in the skin region at that time. Applying at least one rule of this type means introducing the trap object, hence, applying one of these rules is the same for the fate of the computation as applying all possible rules of that kind.
- Steps 5 and 6 of the auxiliary module (see again Figure 1) are performed in the skin region, in parallel with steps of modules from Figures 2 and 3. This can be avoided by introducing one further auxiliary membrane, h', in the skin

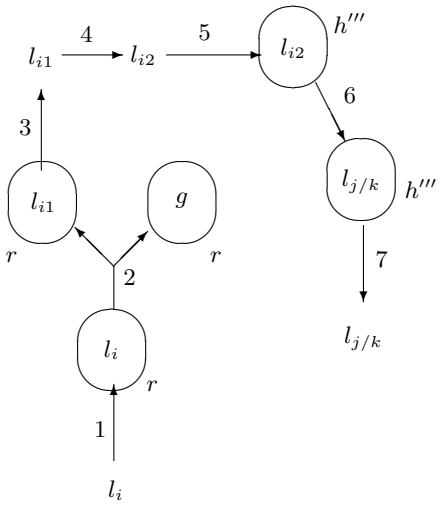


Fig. 6. The work of module ADD, with equal objects in the rules

region, and replacing the rules $[_s d_4 \to d_5]_s$, $[_s d_5 \to d_0]_s$ (both of them from R_s) by the rules $d_4[_{h'}\,]_{h'} \to [_{h'} d_5]_{h'}$ and $[_{h'} d_5]_{h'} \to d_0[_{h'}\,]_{h'}$. These rules are associated with membrane h', hence, they should be used in parallel with the rules from the skin region.

– A further case when the maximal parallelism is important in the previous construction is in step 4 of simulating a SUB instruction for the guess that the register is empty (Figure 4): if rule $l_{i40}[_h\,]_h \to [_h l'_k]_h$ is used in step 4, then also the rule $[_h d_3 \to \#]_h$ must be used. In the minimally parallel mode this can be ensured if we replace this latter rule by $d_3[_{h''}\,]_{h''} \to [_{h''}\#]_{h''}$, where h'' is one further membrane, provided in the initial configuration inside membrane h. In this way, the rule $d_3[_{h''}\,]_{h''} \to [_{h''}\#]_{h''}$ involves the new membrane h'', hence, it should be used in parallel with the rule $l_{i40}[_h\,]_h \to [_h l'_k]_h$. Of course, we also need the rules $[_{h''}\# \to \#']_{h''}$, $[_{h''}\#' \to \#]_{h''}$, instead of the corresponding rules from membrane h.

In this way, we obtain the following counterpart of Theorem 2 (note that we use two further membranes, h' and h''):

Theorem 3. *$N_{min}OP_{n_1,*}((a_1),(b_1),(c_1),(d_1),(e_1)) = NRE$, for all $n_1 \geq 7$.*

In the one-normal form we require that objects appearing on the left-hand side of a rule are different from those appearing on the right-hand side. This restriction can be reversed for rules of types (b) – (d): these rules can be of the forms $a[_h\,]_h \to [_h b]_h$, $[_h a\,]_h \to b[_h\,]_h$, $[_h a\,]_h \to b$, with $a = b$. The changes to be made in the previous proofs are as suggested in Figures 6, 7, 8, 9, which present the modules for simulating ADD and SUB instructions, in the two possible guesses,

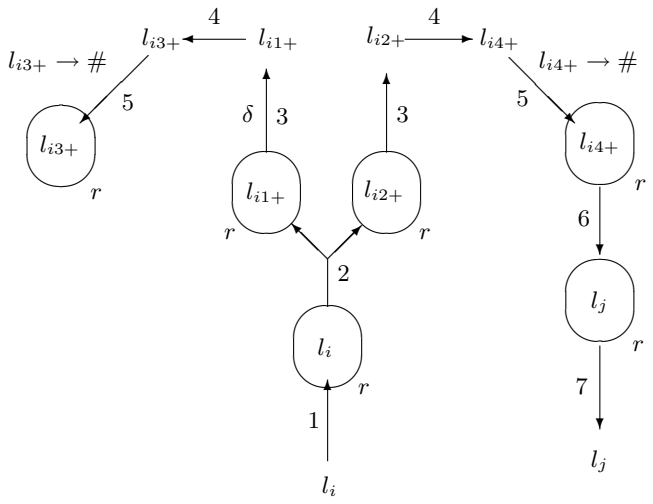


Fig. 7. The work of module SUB, when guessing that the register is non-empty, with equal objects in the rules

as well as the final module, directly for the case of the minimally parallel way of using the rules. This time we need seven steps for simulating an instruction of the register machine and additional membranes h, h', h'', h''', with several evolution steps done inside new membranes in order to change the objects (for instance, this is the case for step 6 from Figure 6, which is done by using the rule $[_{h'''} l_{i2} \to l_\alpha]_{h'''}$ for $\alpha \in \{j, k\}$. The technical details are left to the reader.

One further observation is that the previous systems can easily be modified in order to work in the accepting mode: we introduce the number to be analyzed in the form of the multiplicity of membranes with label 1 initially present in the skin membrane (for number n we start with $n+1$ membranes $[_1\,]_1$ in the system) and we work exactly as above, with the final module now having only the task of dissolving the auxiliary membrane with label h, thus halting the computation (the module can be changed, because it is no longer necessary to send out of the system objects c corresponding to the copies of membrane with label 1 present in the end of the computation, but this is not essential).

Consequently, all the previous results, for both minimally and maximally parallel use of rules, are valid also for accepting P systems in the one-normal form.

5 Efficiency Results

We now pass to proving the efficiency result mentioned in the Introduction – this time, the system is not in the one-normal form, and it remains as an interesting research topic to check whether this is possible or not.

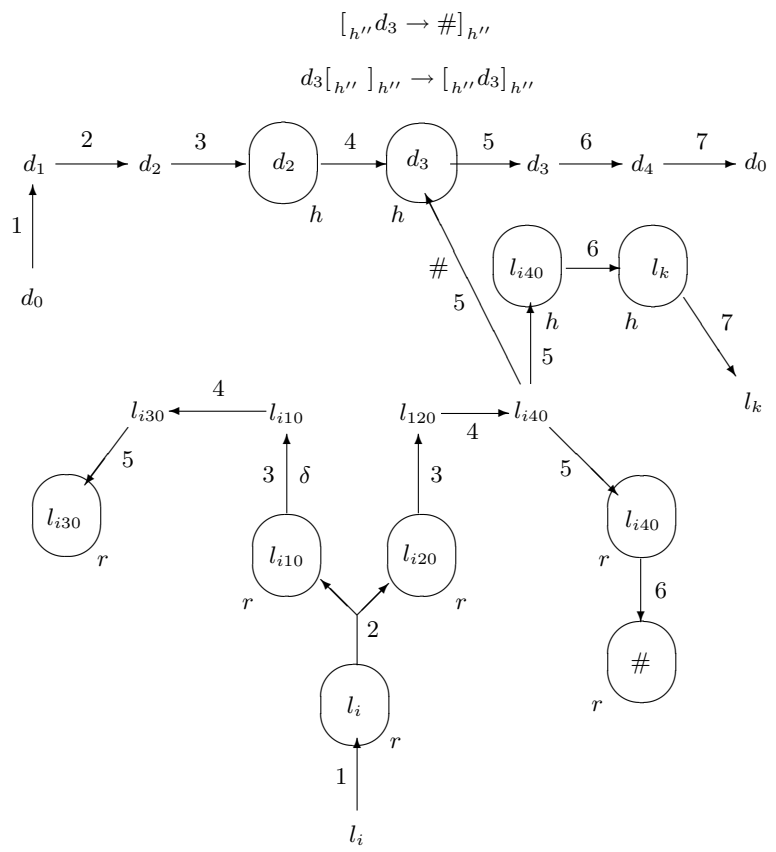


Fig. 8. The work of module SUB, when guessing that the register is empty, with equal objects in the rules

5.1 Solving SAT by Polarizationless P Systems

Below we give an efficient semi–uniform solution to the satisfiability problem by using P systems with polarizationless active membranes working in the minimally parallel mode.

Theorem 4. *The satisfiability of any propositional formula in the conjunctive normal form, using n variables and m clauses, can be decided in a linear time with respect to n by a polarizationless P system with active membranes, constructed in a semi-uniform way in linear time with respect to n and m, and working in the minimally parallel mode.*

Proof. Let us consider a propositional formula $\varphi = C_1 \wedge \ldots \wedge C_m$ such that each clause C_j, $1 \leq j \leq m$, is of the form $C_j = y_{j,1} \vee \ldots \vee y_{j,k_j}$, $k_j \geq 1$, for $y_{j,r} \in \{x_i, \neg x_i \mid 1 \leq i \leq n\}$. For each $i = 1, 2, \ldots, n$, let us denote

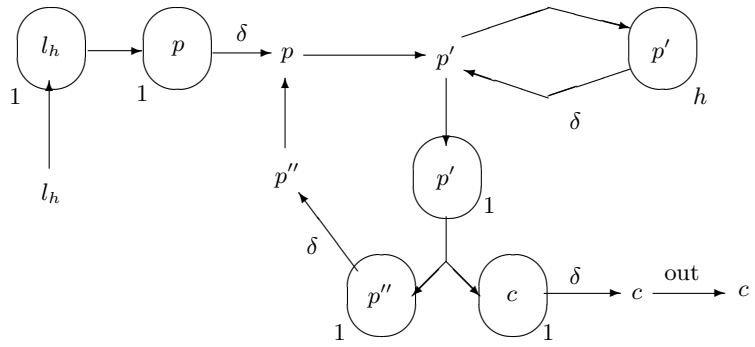


Fig. 9. The work of the final module, with equal objects in the rules

$$t(x_i) = \{c_j \mid \text{there is } r, 1 \le r \le k_j, \text{ such that } y_{j,r} = x_i\},$$
$$f(x_i) = \{c_j \mid \text{there is } r, 1 \le r \le k_j, \text{ such that } y_{j,r} = \neg x_i\}.$$

These are the sets of clauses which assume the value *true* when x_i is *true*, and *false* when x_i is *false*, respectively.

We construct the P system $\Pi(\varphi)$ with the following components (the output membrane is not necessary, because the result is obtained in the environment):

$$\begin{aligned}
O &= \{a_i, f_i, t_i \mid 1 \le i \le n\} \\
&\quad \cup \{c_j, d_j \mid 1 \le j \le m\} \\
&\quad \cup \{p_i \mid 1 \le i \le 2n+7\} \\
&\quad \cup \{q_i \mid 1 \le i \le 2n+1\} \\
&\quad \cup \{r_i \mid 1 \le i \le 2n+5\} \\
&\quad \cup \{b_1, b_2, y, \texttt{yes}, \texttt{no}\}, \\
H &= \{s, s', p, q, r, 0, 1, 2, \ldots, m\}, \\
\mu &= [_s[_{s'}[_p\,]_p[_0[_q\,]_q[_r]_r[_1\,]_1[_2\,]_2 \cdots [_m\,]_m]_0]_{s'}]_s, \\
w_p &= p_1,\ w_q = q_1,\ w_r = r_1,\ w_0 = a_1, \\
w_s &= w_{s'} = w_j = \lambda, \text{ for all } j = 1, 2, \ldots, m.
\end{aligned}$$

The set of evolution rules, R, consists of the following rules:

(1) $[p_i \to p_{i+1}]_p$, for all $1 \le i \le 2n+6$,
$[q_i \to q_{i+1}]_q$, for all $1 \le i \le 2n$,
$[r_i \to r_{i+1}]_r$, for all $1 \le i \le 2n+4$.
These rules are used for evolving counters p_i, q_i, and r_i in membranes with labels p, q, and r, respectively.

(2) $[a_i]_0 \to [f_i]_0[t_i]_0$, for all $1 \le i \le n$,
$[f_i \to f(x_i)a_{i+1}]_0$ and $[t_i \to t(x_i)a_{i+1}]_0$, for all $1 \le i \le n-1$,

$[\, f_n \to f(x_n)]_0$,
$[\, t_n \to t(x_n)]_0$.
The goal of these rules is to generate the truth assignments of the n variables $x_1, \ldots, x_n$, and to analyse the clauses satisfied by x_i and $\neg x_i$, respectively.

(3) $c_j[\ ]_j \to [\, c_j]_j$ and $[\, c_j]_j \to d_j$, for all $1 \leq j \leq m$.
In parallel with the division steps, if a clause C_j is satisfied by the previously expanded variable, then the corresponding object c_j enters membrane j in order to dissolve it and to send objects d_j to membrane 0.

(4) $[\, q_{2n+1}\ ]_q \to q_{2n+1}[\ ]_q$,
$[\, q_{2n+1}\ \to b_1]_0$.
By using these rules the counter q produces an object b_1 in each membrane 0.

(5) $b_1[\ ]_j \to [\, b_1]_j$ and $[\, b_1]_j \to b_2$, for all $1 \leq j \leq m$,
$[\, b_2\ ]_0 \to b_2$.
These rules allow to detect whether the truth assignment associated with a membrane 0 assigns the value *false* to the formula (in that case, the membrane 0 will be dissolved).

(6) $[\, p_{2n+7}\ ]_p \to p_{2n+7}[\ ]_p$,
$[\, p_{2n+7}\ ]_{s'} \to \texttt{no}[\ ]_{s'}$,
$[\, \texttt{no}\ ]_s \to \texttt{no}[\ ]_s$,
$[\, r_{2n+5}\ ]_r \to r_{2n+5}$,
$[\, r_{2n+5}\ ]_0 \to y[\ ]_0$,
$[\, y\ ]_{s'} \to \texttt{yes}$,
$[\, \texttt{yes}\ ]_s \to \texttt{yes}[\ ]_s$.
These rules produce the answer of the P system.

An overview of the computations in the P system

For the sake of readability, the initial configuration is given in Figure 10.

The membranes with labels p, q, and r, with the corresponding objects p_i, q_i, and r_i, respectively, are used as counters, which evolve simultaneously with the *main membrane* 0, where the truth assignments of the n variables $x_1, \ldots, x_n$ are generated; the use of separate membranes for counters makes possible the correct synchronization even for the case of the minimal parallelism. The evolution of counters is done by the rules of type (1).

In parallel with these rules, membrane 0 evolves by means of the rules of type (2). In odd steps (from step 1 to step $2n$), we divide the (non-elementary) membrane 0 (with f_i, t_i corresponding to the truth values *false, true*, respectively, for variable x_i); in even steps we introduce the clauses satisfied by $x_i, \neg x_i$, respectively. When we divide membrane 0, all inner objects and membranes are replicated; in particular, all membranes with labels $1, 2, \ldots, m$, as well as membranes q and r, are replicated, hence, they are present in all membranes with label 0.

This process lasts $2n$ steps. At the end of this phase, all 2^n truth assignments for the n variables have been generated and they are encoded in membranes labeled by 0.

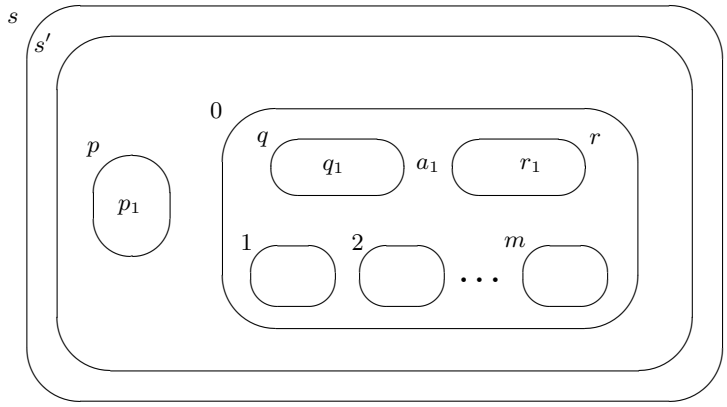


Fig. 10. The initial configuration of the system from the proof of Theorem 4

In parallel with the division steps, if a clause C_j is satisfied by the previously expanded variable, then the corresponding object c_j enters membrane j, by means of the first rule of type (3), permitting its dissolution by means of the second rule of that type and sending objects d_j to membrane 0.

This is done also in step $2n+1$, in parallel with using the rules of type (1) and (4) for evolving membranes p, q, and r.

In step $2n+2$, the counters p_i and r_i keep evolving and the second rule of type (4) produces an object b_1 in each membrane 0.

Thus, after $2n+2$ steps, the configuration $\mathcal{C}_{2n+2}$ of the system consists of 2^n copies of membrane 0, each of them containing the membrane p empty, the membrane r with the object r_{2n+3}, possible objects c_j and d_j, $1 \leq j \leq m$, as well as copies of those membranes with labels $1, 2, \ldots, m$ corresponding to clauses which were not satisfied by the truth assignment generated in that copy of membrane 0. The membranes associated with clauses satisfied by the truth assignments generated have been dissolved by the corresponding object c_j. Moreover, in that configuration the membrane p contains the object p_{2n+3}, and membranes s' and s are empty.

Therefore, formula φ is satisfied if and only if there is a membrane 0 where all membranes $1, 2, \ldots, m$ have been dissolved. In order to check this last condition, we proceed as follows.

In step $2n+3$ we use the first rule of type (5) which introduces the object b_1 in a membrane j which has not been dissolved (this is made in a non–deterministically manner). In parallel, the counters q and r follow with evolving. The object b_1 in membrane j (step $2n+4$) dissolves that membrane producing an object b_2 in membrane 0.

In step $2n+5$ the counter r_{2n+5} exits from membrane r and, simultaneously, each membrane 0 containing an object b_2 is dissolved by the third rule of type (5).

Then, formula φ is satisfied if and only if in the configuration $\mathcal{C}_{2n+5}$ there exists a membrane 0 that has not been dissolved (and thus containing the object r_{2n+5}).

In the next step, the counter q_i evolves to q_{2n+7} in membrane q, and if there is a membrane 0 that has not been dissolved, the object r_{2n+5} sends an object y to membrane s'. On the contrary, only the counter q_i evolves.

In step $2n+7$ the counter p_{2n+7} exits from membrane p to membrane s', by applying the first rule of type (6). If the formula φ is satisfiable then an object y dissolves the membrane s' by applying the sixth rule of type (6) producing an object `yes` in the skin. In the next step, this object is sent to the environment and the P system halts. On the contrary, if membrane s' has not been dissolved, the object p_{2n+7} in membrane s' produces an object `no` in the skin, by using the second rule of type (6); in the next step an object `no` is sent to the environment and the system halts.

Therefore, if the formula is satisfiable, then the object `yes` exits the system in step $2n+8$, and, if the formula is not satisfiable, then the object `no` exits the system in step $2n+9$. In both cases, this is the last step of the computation.

The system $\Pi(\varphi)$ uses $9n+2m+18$ objects, $m+6$ initial membranes, containing in total 4 objects, and $8n+4m+21$ rules. The length of any rule is bounded by $m+3$. Clearly, all computations stop (after at most $2n+9$ steps) and all give the same answer, `yes` or `no`, to the question whether formula φ is satisfiable, hence, the system is weakly confluent. These observations conclude the proof. □

Remark 1. 1. The system used in the previous proof is not in the one-normal form, because the rules of type (b) are of the form $[_h a \to u]_h$ with u being an arbitrary multiset. Because the maximal length of such strings u is known ($m+1$), we can replace each rule of this form by m rules, each of them introducing two objects; using rules of the form $[_h a \to b]_h$, we can also synchronize the applications of rules, hence, at the price of getting a computation m times longer, we can obtain a sort of two-normal form. Of course, the rules of types (b), (c), (d) can be also arranged to have either different objects or identical objects. We leave the details as a task for the reader, together with the more interesting issue of finding a polynomial solution to `SAT` by a system in the one-normal form.

2. Let us note that we can design a *deterministic* P system $\Pi(\varphi)$ working in minimally parallel mode which decides the satisfiability of φ. To this aim, it is enough to have m copies of the object b_1 in each membrane 0 of the configuration $\mathcal{C}_{2n+2}$. For that, the rule $[q_{2n+1} \to b_1]_0$ can be replaced by $[q_{2n+1} \to b_1^m]_0$.
3. As a consequence of Theorem 4 we obtain the inclusion of $\mathbf{NP} \cup \mathbf{co}-\mathbf{NP}$ in the class of all decision problems solvable in polynomial time in a semi–uniform way by a family of P systems with polarizationless active membranes working in the minimally parallel mode (let us recall that due to the confluence of the P systems solving a decision problem, every P system that decides an instance working in minimally parallel mode, also decides it working in the maximally parallel mode, but the reciprocal is not true).

5.2 A Formal Verification

In order to assure that the system $\Pi(\varphi)$ decides the instance φ, two main properties must to be proved: (a) if *there exists an* accepting computation of the P system processing φ, answering `yes`, then the problem also answers `yes` for that instance (*soundness*), and (b) if the problem answers *yes*, then *any* computation of the P system processing that instance answers *yes* (*completeness*). Hence, the system $\Pi(\varphi)$ must satisfy a condition of *confluence*: every computation of the system has the same output.

To prove that the system $\Pi(\varphi)$ is sound and complete with respect to the **SAT** problem, it is sufficient to show that a truth assignment makes true the formula φ if and only if in the configuration $\mathcal{C}_{2n+5}$ there is at least a membrane labeled by 0 that has not been dissolved.

If σ is a truth assignment on a set of variables $\{x_1, \ldots, x_n\}$, then we write $\sigma(x_i) = f$ or $\sigma(x_i) = t$.

Lemma 2. *For each i $(1 \leq i \leq n)$ the following conditions hold true:*

1. $\mathcal{C}_{2i-1}(p) = \{p_{2i}\}$, $\mathcal{C}_{2i-1}(s) = \mathcal{C}_{2i-1}(s') = \emptyset$.
2. *For each truth assignment σ on $\{x_1, \ldots, x_i\}$ there exists a unique membrane 0 such that its contents is*

$$\{(\sigma(x_i))_i\} \cup \{d_j \mid c_j \in \bigcup_{1 \leq l \leq i-2} (\sigma(x_l))(x_l) \wedge i \geq 3\} \cup \{c_j \mid c_j \in \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l) \wedge i \geq 3\}.$$

   Moreover, it inside contains a membrane labeled by q which contains the object q_{2i}, a membrane labeled by r which contains the object r_{2i}, and empty inner membranes j, where $1 \leq j \leq m$; moreover, if $i \geq 2$, then:

   (a) $c_j \notin \bigcap_{1 \leq l \leq i-2} (\sigma(x_l))(x_l)$,

   (b) if $c_j \in (\sigma(x_{i-1}))(x_{i-1}) - \bigcap_{1 \leq l \leq i-2} (\sigma(x_l))(x_l)$, *then* $\mathcal{C}_{2i-1}(j) = \{c_j\}$.
3. $\mathcal{C}_{2i}(p) = \{p_{2i+1}\}$, $\mathcal{C}_{2i}(s) = \mathcal{C}_{2i}(s') = \emptyset$.
4. *For each truth assignment σ on $\{x_1, \ldots, x_i\}$ there exists a unique membrane 0 such that its contents is*

$$\{(\sigma(x_i))(x_i), a_{i+1}\} \cup \{d_j \mid c_j \in \bigcup_{1 \leq l \leq i-1} (\sigma(x_l))(x_l) \wedge i \geq 2\} \cup \{c_j \mid c_j \in \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l) \wedge i \geq 3\},$$

   where $a_{n+1} = \lambda$.
   Furthermore, it contains inside a membrane labeled by q which contains the object q_{2i+1}, a membrane labeled by r which contains the object r_{2i+1}, and

empty inner membranes j, where $1 \leq j \leq m$; moreover, if $i \geq 2$, then $c_j \notin \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l)$.

Proof. By induction on n. It is easy to prove the base case. Let us suppose that the result holds for i $(1 \leq i < n)$.

From the induction hypothesis, we deduce that the configuration $\mathcal{C}_{2i+1}$ is obtained from $\mathcal{C}_{2i}$ by applying the rules $[p_{2i+1} \rightarrow p_{2i+2}]_p$, $[q_{2i+1} \rightarrow q_{2i+2}]_q$, $[r_{2i+1} \rightarrow r_{2i+2}]_r$, $[a_{i+1}]_0 \rightarrow [f_{i+1}]_0[t_{i+1}]_0$, and $c_j\,[\ ]_j \rightarrow [c_j]_j$ $(1 \leq j \leq m)$.

Hence, we have $\mathcal{C}_{2i+1}(p) = \{p_{2i+2}\}$ and $\mathcal{C}_{2i+1}(s) = \mathcal{C}_{2i+1}(s') = \emptyset$. Moreover, for each truth assignment σ on $\{x_1, \ldots, x_i, x_{i+1}\}$ there exists a unique membrane 0 which contains

$$\{(\sigma(x_{i+1}))_{i+1}\} \cup \{d_j \mid c_j \in \bigcup_{1 \leq l \leq i-1} (\sigma(x_l))(x_l) \wedge i \geq 2\} \cup \{c_j \mid c_j \in \bigcap_{1 \leq l \leq i} (\sigma(x_l))(x_l) \wedge i \geq 3\}.$$

This membrane 0 inside contains a membrane labeled by q which contains the object q_{2i+2}, a membrane labeled by r which contains the object r_{2i+2}, and inner membranes j such that:

- if $i \geq 2$ and $c_j \in \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l)$, then membrane j has been dissolved,
- if $i \geq 2$ and $c_j \in (\sigma(x_i))(x_i) - \bigcap_{1 \leq l \leq i-1} (\sigma(x_l))(x_l)$, then $\mathcal{C}_{2i+1}(j) = \{c_j\}$;
- the remaining membranes are empty.

The configuration $\mathcal{C}_{2i+2}$ is obtained from $\mathcal{C}_{2i+1}$ by applying the rules $[p_{2i+2} \rightarrow p_{2i+3}]_p$, $[q_{2i+2} \rightarrow q_{2i+3}]_q$; $[r_{2i+2} \rightarrow r_{2i+3}]_r$, $[f_{i+1} \rightarrow f(x_{i+1}a_{i+1}]_0$, $[t_{i+1} \rightarrow t(x_{i+1})a_{i+1}]_0$, where $a_{n+1} = \lambda$, and $c_j\,[\ ]_j \rightarrow [c_j]_j$ $(1 \leq j \leq m)$.

Hence, we have $\mathcal{C}_{2i+2}(p) = \{p_{2i+3}\}$ and $\mathcal{C}_{2i+2}(s) = \mathcal{C}_{2i+2}(s') = \emptyset$. Moreover, for each truth assignment σ on $\{x_1, \ldots, x_i, x_{i+1}\}$ there exists a unique membrane 0 which contains

$$\{(\sigma(x_{i+1}))(x_{i+1})\} \cup \{d_j \mid c_j \in \bigcup_{1 \leq l \leq i} (\sigma(x_l))(x_l) \wedge i \geq 2\} \cup \{c_j \mid c_j \in \bigcap_{1 \leq l \leq i} (\sigma(x_l))(x_l) \wedge i \geq 3\}$$

That membrane 0 inside contains a membrane labeled by q which contains the object q_{2i+3}, a membrane labeled by r which contains the object r_{2i+3}, and empty inner membranes j, where $1 \leq j \leq m$; if $i \geq 2$, then $c_j \notin \bigcap_{1 \leq l \leq i} (\sigma(x_l))(x_l)$. □

Lemma 3. *The configuration $\mathcal{C}_{2n+5}$ fulfills the following conditions:*

1. $\mathcal{C}_{2n+5}(p) = \{p_{2n+6}\}$, $\mathcal{C}_{2n+5}(s) = \emptyset$, *and the content of* $\mathcal{C}_{2n+5}(s')$ *is*

$$\{b_2^u, r_{2n+5}^u\} \cup \{d_j \mid c_j \in \bigcup_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 2\}$$
$$\cup \{c_j \mid c_j \in \bigcap_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 3\},$$

   where u *is the number of truth assignment* σ *such that* $\sigma(\varphi) = 0$.
2. *For each assignment* σ *which satisfies the formula* φ *there exists a unique membrane 0 whose contents is*

$$\{r_{2n+5}\} \cup \{d_j \mid c_j \in \bigcup_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge n \geq 2\}$$
$$\cup \{c_j \mid c_j \in \bigcap_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge n \geq 3\}.$$

Corollary 1. *The formula* φ *is satisfiable if and only if in the configuration* $\mathcal{C}_{2n+5}$ *there is at least a membrane labeled by 0 that has not been dissolved.*

Proof. Indeed, a truth assignment σ is a satisfying assignment if and only if in the configuration $\mathcal{C}_{2n+4}$ the object b_1 from the membrane 0 associated with the assignment σ has not entered any membrane j $(1 \leq j \leq m)$, because all such membranes have been dissolved in previous steps. But this condition is equivalent to the existence of some membrane labeled by 0 in the configuration $\mathcal{C}_{2n+5}$. □

Corollary 2. *Let* u *be the number of truth assignments* σ *such that* $\sigma(\varphi) = 0$, *and let* $\mathcal{C}$ *be a computation of* $\Pi(\varphi)$.

1. *If the formula* φ *is satisfiable, then* $\mathcal{C}_{2n+8}(p) = \emptyset$, *the contents of* $\mathcal{C}_{2n+8}(s)$ *is*

$$\{b_2^u, r_{2n+5}^u, y^{n-u-1}, p_{2n+7}\} \cup \{d_j \mid c_j \in \bigcup_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 2\}$$
$$\cup \{c_j \mid c_j \in \bigcap_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 3\},$$

   $\mathcal{C}_{2n+8}(env) =$ `yes`, *and the system halts giving an affirmative answer.*
2. *If the formula* φ *is not satisfiable, then* $\mathcal{C}_{2n+8}(p) = \emptyset$, *the contents of* $\mathcal{C}_{2n+8}(s')$ *is*

$$\{b_2^u, r_{2n+5}^u, y^{n-u}\} \cup \{d_j \mid c_j \in \bigcup_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 2\}$$
$$\cup \{c_j \mid c_j \in \bigcap_{1\leq l\leq n} (\sigma(x_l))(x_l) \wedge \sigma(\varphi) = 0 \wedge n \geq 3\},$$

   $\mathcal{C}_{2n+8}(s) = \{\texttt{no}\}$, $\mathcal{C}_{2n+9}(p) = \emptyset$, $\mathcal{C}_{2n+9}(s) = \emptyset$, $\mathcal{C}_{2n+9}(env) =$ `no`, *and the system halts giving a negative answer.*

6 Final Remarks

We have shown that P systems with polarizationless active membranes are computationally complete, even when working in the minimally parallel mode with very restricted forms of the rules, i.e., only evolving one object in or through a membrane (one-normal form). Moreover, we have also shown that `SAT` can be solved by P systems with polarizationless active membranes even when working in the minimally parallel mode, but in this case the question whether we can restrict the rules to the one-normal form remains as an open problem.

Acknowledgements

The work of Gh. Păun was partially supported by Project BioMAT 2-CEx06-11-97/19.09.06. The work of M.J. Pérez-Jiménez was supported by the project TIN2005-09345-C04-01 of the Ministerio de Educación y Ciencia of Spain, co–financed by FEDER funds, and of the project of Excellence TIC 581 of the Junta de Andalucía.

References

1. A. Alhazov: P systems without multiplicities of symbol-objects. *Information Processing Letters*, 100 (2006), 124–129.
2. A. Alhazov, M.J. Pérez-Jiménez: Uniform solution to QSAT using polarizationless active membranes, Vol. I of Proc. of the Fourth Brainstorming Week on Membrane Computing, Sevilla (Spain), Jan 30 - Feb 3, 2006, 29–40.
3. A. Alhazov, R. Freund, On efficiency of P systems with active membranes and two polarizations. In [7], 81–94.
4. A. Alhazov, R. Freund, Gh. Păun: Computational completeness of P systems with active membranes and two polarizations. In *Machines, Computations, and Universality. 4th Intern. Conf. Sankt Petersburg, Russia, 2004*, LNCS 3354 (M. Margenstern, ed.), Springer, 2005, 82–92.
5. G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez: P systems with minimal parallelism. *Theoretical Computer Sci.*, to appear.
6. J.E. Hopcroft, J.D. Ullman: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.
7. G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing, International Workshop, WMC5, Milano, Italy, 2004, Selected Papers*. Lecture Notes in Computer Science 3365, Springer-Verlag, Berlin, 2005.
8. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
9. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
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: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.

12. Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds.: *Proceedings of the Second Brainstorming Week on Membrane Computing, Sevilla, February 2004*. Technical Report 01/04 of Research Group on Natural Computing, Sevilla University, Spain, 2004.
13. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: *Teoria de la complejidad en modelos de computacion celular con membranas*. Kronos Editorial, Sevilla, 2002.

Spiking Neural P Systems: Stronger Normal Forms

Marc García-Arnau[1], David Peréz[1], Alfonso Rodríguez-Patón[1], Petr Sosík[1,2]

[1] Universidad Politécnica de Madrid - UPM, Facultad de Informática
Campus de Montegancedo s/n, Boadilla del Monte
28660 Madrid, Spain mgarnau@dia.fi.upm.es
[2] Institute of Computer Science, Silesian University
74601 Opava, Czech Republic petr.sosik@fpf.slu.cz

Summary. Spiking neural P systems are computing devices recently introduced as a bridge between spiking neural nets and membrane computing. Thanks to the rapid research in this field there exists already a series of both theoretical and application studies. In this paper we focus on normal forms of these systems while preserving their computational power. We study combinations of existing normal forms, showing that certain groups of them can be combined without loss of computational power, thus answering partially open problems stated in [8, 9]. We also extend some of the already known normal forms for spiking neural P systems considering determinism and strong acceptance condition. Normal forms can speed-up development and simplify future proofs in this area.

1 Introduction

Spiking neural P systems (SN P systems) are a rather new bio-inspired computational model that incorporates to membrane computing [6] some ideas from spiking neurons [3], [4].

Since they were first presented in [2], the number of publications dealing with this model is constantly growing. An interesting review on the current research topics in SN P systems can be found in [9].

Informally, an SN P system consists of a set of neurons placed in the nodes of a graph that are linked by synapses. These neurons send signals (*spikes*) along the arcs of the graph. To do so, the neurons contain firing or spiking rules which are of the form $E/a^r \rightarrow a;t$ with E being a regular expression, r being the number of spikes consumed by the rule and t being the delay from firing the rule and emitting the spike. A firing rule can be only used if the number n of spikes collected by the neuron is such that $a^n \in L(E)$, that is, a^n is covered by the regular expression E, and $n \geq r$. The neurons also have an interesting feature imitating the refractory period of real neural cells. Thus, a neuron in an SN P system is closed/blocked

for exactly t time steps after firing. During this period it cannot fire again. The second type of rules have the form $a^s \rightarrow \lambda$ and are called forgetting rules. They are used to simply forget (remove) s spikes from a cell. SN P systems start from an initial configuration of spikes and evolve in a synchronized manner (a global clock is assumed for the whole system). One of the neurons is designated as the output cell and the spikes it sends to the environment constitute the output of the system.

The first work in SN P systems [2] presented them as devices generating or accepting sets of natural numbers. Their universality was proven when no bound is imposed on the number of spikes. Otherwise, only a characterization of semilinear sets is obtained. Later, a new paper [8] dealing with normal forms of the model attacked its universality trying to improve the previous proofs. In that article, universality results were obtained even if some of the main features of the model were weakened. For instance, universality was proven even without the use of delays. Furthermore, the outdegree of neurons was reduced to the minimal bound of two. Next, SN P systems were found to be also universal when forgetting rules were removed and, finally, computational completeness was still achieved when using the simplest possible regular expressions λ and a^* over the alphabet $\{a\}$ in firing rules. Another work [10] has also shown that not only the outdegree but also the indegree of neurons can be bound to two without loosing universality.

In the present paper, we deal with some of the open problems stated in [8]. Actually, we try to keep the universality of the model when eliminating two of its key features simultaneously. Interesting results have been obtained. Surprisingly, SN P systems are still universal when we use neither delays nor forgetting rules. Moreover, one can also eliminate delays while simplifying regular expressions and the model keeps its computational completeness. Finally, we have proven the universality of the model in two more cases: 1) using simple regular expressions with strong halting condition and 2) using simple regular expressions with SN P systems working in the accepting mode.

In all these cases, the reader will observe that the simultaneous elimination of two features of the model has a price in terms of other complexity parameters, such as the maximal number of firing rules in a neuron, the complexity of regular expressions, the maximum number of spikes consumed in a firing rule or the maximal number of spikes removed in a forgetting rule.

The remainder of the paper is organized as follows. Section 2 presents some important definitions. In section 3 we present the universality result of the model using neither delays nor forgetting rules. Section 4 describes the power of SN P systems with simple regular expressions that do not use delays. Some more aspects concerning regular expressions are revisited in section 5. The paper concludes with some final remarks in section 6.

2 Definitions

In this section, we recall some useful definitions. We consider the reader to be familiar with elements of membrane computing. One can find in [11] the most updated information on this area. The reader is considered to be familiar with elements of language and automata theory, as well.

Nevertheless, we recall some basic notation. Let V denote an alphabet, while V^* denotes the set of all finite strings of symbols from V. The set of nonempty strings over V is denoted by V^+ and λ denotes the empty string. The length of a string $x \in V^*$ is denoted by $|x|$. In the domain of SN P systems, the alphabet V contains only one symbol, i.e., the alphabet is a singleton $V = \{a\}$. Then a^* and a^+ are normally used instead of $\{a\}^*$ and $\{a\}^+$.

A *spiking neural membrane system* (abbreviated as SN P system), of a degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_0),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m,$$

   where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^r \rightarrow a; t$, where E is a regular expression over a, $r \geq 1$, and $t \geq 0$;
      (2) $a^s \rightarrow \lambda$, for some $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses* between neurons);
4. $i_0 \in \{1, 2, \ldots, m\}$ indicates the *output neuron* (i.e., σ_{i_0} is the output neuron).

The rules of type (1) are *firing rules* (also called *spiking rules*). The notation $E/a^r \rightarrow a, t$ means that when the number of spikes present in a neuron is covered by the regular expression E, the neuron gets fired, r spikes are consumed and, after t time steps, one spike is emitted by the neuron to all its neighbors (the system is synchronized, a global clock is assumed for all its cells).

In a SN P system we have maximal parallelism at the level of the system as, in one step, all neurons that can use a rule have to use it. However, at the neuronal level, we work in a sequential mode with at most one rule used in each step by a neuron. SN P systems also incorporate an interesting bio-inspired feature called the refractory period. In the interval between using a spiking rule (getting fired) and releasing the spike, the neuron is assumed to be closed (it omits any other spike received during this interval and, of course, it cannot fire). Then, if $t = 0$

there is no restriction and the neuron can receive spikes in the same step it uses the rule. Similarly, a neuron can receive spikes in moment t when $t \geq 1$. When a neuron spikes, its spike is replicated to all its neighboring neurons that are not closed in that moment. These spikes are available in the receiving neuron already in the next step, so the transmission of a spike takes no time.

The rules of type (2) are called *forgetting rules*. They are written as $a^s \to \lambda$ and they can be applied only when the neuron contains exactly s spikes. After applying one of these rules, s spikes are simply removed from the cell.

Firing rules can be used in a non-deterministic way, that is, a neuron may contain two firing rules $E_1/a^{r_1} \to a; t_1$ and $E_2/a^{r_2} \to a; t_2$ such that $L(E_1) \cap L(E_2) \neq \emptyset$. However, this non-determinism is not allowed between firing and forgetting rules, that is, in a single step a neuron has either to fire or forget, without being possible to freely choose between these two actions. This is called the *minimal determinism-like restriction* or the *coherence* condition. Hence, we just allow branching in the case of spiking rules.

We define a *computation* of an SN P system as a sequence of steps during which rules are applied in the above described parallel manner. A computation starts in the initial configuration when each neuron σ_i contains n_i spikes, $1 \leq i \leq m$. A *halting computation* is that in which the system reaches a configuration where no more rules can be applied. A computation is called *strong halting* if, in addition, no spike is present in the system when it halts.

The usual way of interpreting *outputs* of SN P systems is considering *intervals* in which the output neuron i_0 spikes (not when it fires). For simplicity, one considers as successful only computations with the output neuron spiking at least twice. The set of numbers computed by an SN P system in this way is denoted by $N_2(\Pi)$. If we take into consideration only the computations having exactly 2 spikes (strong case) then this set is written as $N_{\underline{2}}(\Pi)$. Similarly, if only halting (strong halting) computations are taken into the account, we denote the resulting sets by $N_2^h(\Pi)$ ($N_2^{\underline{h}}(\Pi)$, respectively). The reader will find in [7] and [8] several other relevant definitions.

We denote by $Spik_2^{\beta}P_m(rule_k, cons_p, forg_q, dley_r, outd_s)$ the family of all sets $N_2^{\beta}(\Pi)$ (with $\beta = \{h, \underline{h}\}$), for all systems Π with at most m neurons, each neuron having at most k rules, each of the spiking rules consuming at most p spikes, each of the forgetting rules removing no more than q spikes, with all spiking rules having a delay small or equal to r and with all neurons having at most s outgoing synapses. We also may write $rule_k^*$ if the firing rules are of the form $E/a^r \to a; t$ with the regular expression of one of the forms $E = \lambda$ or $E = a^*$ (in the former case the rule is written as $a^r \to a; t$ to simplify).

Finally, we define a *register machine*, which is the computational model used to prove the universality of SN P systems, as it is known (see [5]) that register machines (even with a small number of registers, although this detail is not relevant here) characterize *NRE.*

A *register machine* is a construct $M = (m, H, l_0, l_h, I)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label (labeling an

ADD instruction), l_h is the halt label (assigned to the instruction HALT), and I is the set of instructions. Each label from H labels only one instruction from I (but the same instruction may be assigned to more labels). The instructions are of the following forms:

- $l_1 : (\texttt{ADD}(r), l_2, l_3)$ (add 1 to register r and then go to one of the instructions with labels l_2, l_3),
- $l_1 : (\texttt{SUB}(r), l_2, l_3)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_2, otherwise go to the instruction with label l_3),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M computes a number n in the following way: we start with all registers empty (storing the number zero), we apply the instruction with label l_0 and we proceed to apply instructions as indicated by the labels (and made possible by the contents of registers). If we reach the halt instruction, then the number n stored at that time in the first register is said to be computed by M. Therefore $N(M)$ denotes the set of all numbers computed by the register machine M.

Without loss of generality, we may assume in the next sections that in the halting configuration, all registers different from the first one are empty, and that the output register is never decremented during the computation, we only add to its contents.

A register machine can also work in the *accepting* mode: a number n is introduced in the first register (all other registers are empty) and we start computing with the instruction with label l_0; if the computation eventually halts, then the number n is accepted. Register machines are universal also in the accepting mode; moreover, this is true even for deterministic machines, having ADD rules of the form $l_1 : (\texttt{ADD}(r), l_2, l_3)$ with $l_2 = l_3$ (in such a case, the instruction is written in the form $l_1 : (\texttt{ADD}(r), l_2)$). Again, without loss of generality, we may assume that in the halting configuration all registers are empty.

3 Removing Delays and Forgetting Rules Simultaneously

In this section we extend the original result of Theorem 3.1 of [2], paying special attention to delays and forgetting rules. As it was shown in [8], computational completeness is achieved when eliminating each one of these two parameters separately. Here, we extend these results demonstrating that SN P Systems are also universal when eliminating both delays and forgetting rules at the same time. Simultaneously, we have also bounded the outdegree of neurons to two. However, for the sake of clarity, we don't show it graphically in our demonstration.

Nevertheless, this elimination has a price in terms of other parameters. Namely, the maximal number of rules used in a neuron rises to three and we lose the strong halting condition. As all rules we use have delay 0, we write them in the simpler form $E/a^r \to a$, that is, omitting the delay.

Theorem 1. *$Spik^{\beta}_{\underline{2}}P_*(rule_3, cons_3, forg_0, dley_0, outd_2) = NRE$, where $\beta = h$ or β is omitted.*

Proof. The inclusion $Spik^{\beta}_{\underline{2}}P_*(rule_*, cons_*, forg_*, dley_*, outd_*) \subseteq NRE$ is straightforward and therefore we omit it (Turing-Church thesis). To complete the proof we must show $NRE \subseteq Spik^{\beta}_{\underline{2}}P_*(rule_3, cons_3, forg_0, dley_0, outd_2)$. As in other demonstrations, we will construct an SN P system (Π) spiking only twice, at an interval of time which corresponds to a number computed by a register machine M. Our system consists of modules simulating the ADD and SUB instructions and the output module FIN which takes care of the final spiking of the system Π.

Every register r of M will be associated to a neuron of Π. However, in contrast with some other previous demonstrations, a register containing the value n will hold $2n + 2$ spikes. Any register representing the value 0 will therefore contain a couple of spikes. This slight modification in the way of representing numbers will allow us to detect correctly whether a register contains the value 0 in the SUB module without making use of delays or forgetting rules.

Simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ – module ADD (Figure 1).

This instruction adds one to the register r and branches non-deterministically to label l_j or l_k. This module is initiated when a spike enters neuron l_i (we can assume that the initial instruction of M, labeled with l_0, is always an ADD instruction). The neuron l_i sends then one spike to neurons c_1 and c_2. In the next step, one spike coming from each of these neurons reaches the neuron r, adding one to the content of the register. At the same time, the spike emitted by c_1 arrives to c_3 (which will in turn be send to c_6 in the following step) and c_4, while the spike of c_2 reaches c_4 and c_5. Neuron c_4 will allow us to branch non-deterministically to either l_j or l_k. If c_4 uses the rule $a^2 \to a$, then two spikes will be blocked in c_8 (those coming from c_4 and c_5), while just one will arrive to neuron c_7 waiting for another one to come. In the next step, the spike from c_6 reaches also c_7 and it gets fired, activating neuron l_j one step later.

On the other hand, if c_4 uses the rule $a^2/a \to a$ it consumes only one of its two spikes. This means that, in the following step, c_7 receives one spike and c_8 receives two (from c_4 and c_5). One step later, c_4 uses its rule $a \to a$ and sends another spike to c_7 (which also receives the one from c_6 and therefore cannot fire) and to c_8 that now contains three spikes and fires, activating l_k in the following step.

The reader will appreciate that, after each ADD instruction, neurons c_7 and c_8 will hold 3 or 2 spikes, respectively, depending on the rule selected in the non-deterministic neuron c_4. Thanks to the regular expressions used in the rules of c_7 and c_8, this does not disturb further computations using this instruction.

In this construction, neurons c_1 and c_2 have an outdegree of three. However, it is trivial to see that it could be reduced to two by placing more neurons between them and neurons c_3, c_4 and c_5, as it is explained in Section 5 of [8].

Simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ – module SUB (Figure 2).

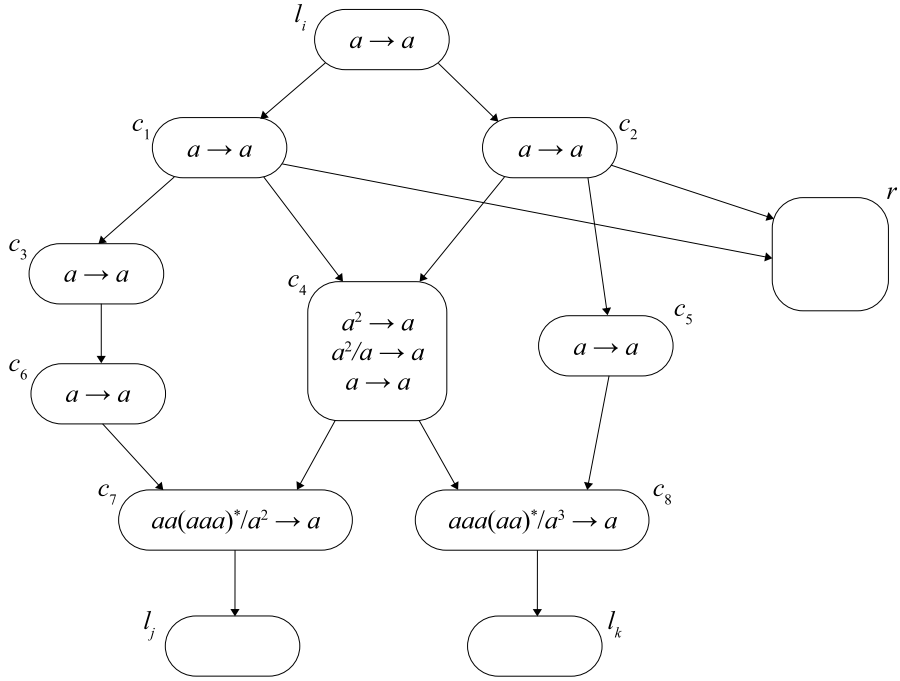


Fig. 1. Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)

The module is initiated when a spike is sent to neuron l_i. This neuron fires and its spike reaches neurons d_1, d_2 and r. The three rules of neuron r allow us to differentiate whether the register is empty or not. As we have previously explained, storing the value n means to contain $2n + 2$ spikes. Thus, when $r > 0$, (i.e., it contains at least 4 spikes) the spike coming from l_i makes the neuron fire (rule $aaa(aa)^+/a^3 \to a$) sending a spike to d_4 and d_5. At the same time, another spike coming from d_2 reaches d_5, not allowing it to fire. In parallel, a spike is sent from d_1 to d_3 and, in the following step, it arrives to d_4. This neuron fires because it already contains two spikes, allowing us to finally reach l_j.

On the other hand, when r stores number zero (it contains 2 spikes), the spike received from l_i fires the rule $a^3/a^2 \to a$. Then neuron r spikes, consuming two of the three spikes it contains. This spike is sent to neurons d_4 and d_5. Another spike reaches d_5 simultaneously (from d_2), while d_3 receives the spike coming from d_1. In the following step, r fires again (using the rule $a \to a$), consuming its last spike and sending a new spike to d_4 and d_5 (the value 0 of r is now degraded and needs to be reconstituted). This spike reaches neuron d_4 at the same time that the one coming from d_3. Then d_4 cannot fire as it contains now three spikes. Meanwhile, d_5 receives the new spike from r (d_5 now contains three spikes). It gets fired and spikes, allowing neurons d_6 and d_8 to fire in the following step. Each of these two

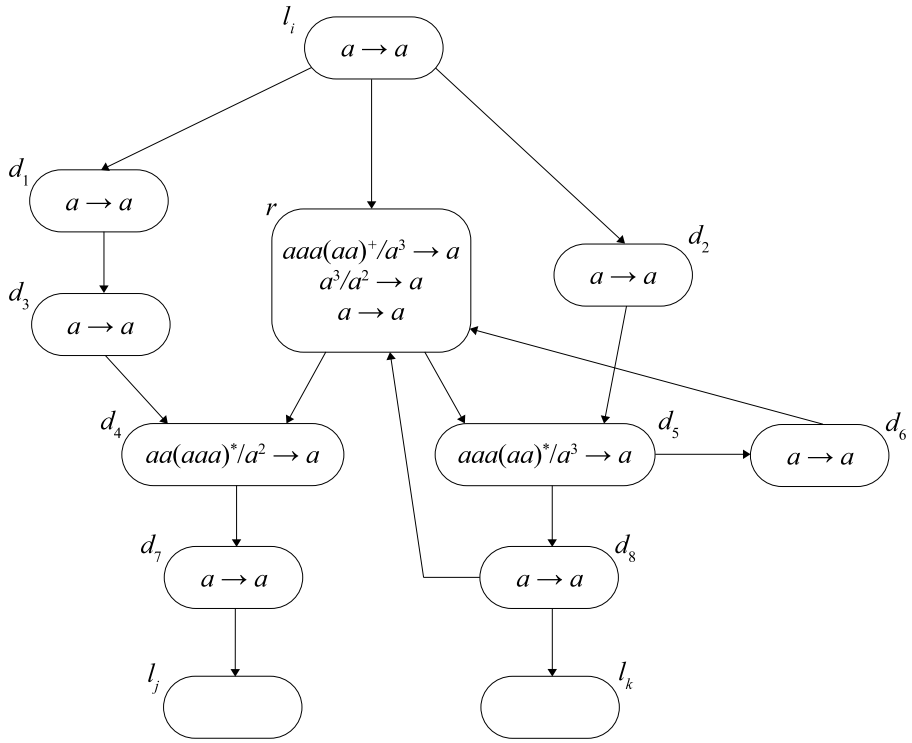


Fig. 2. Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$)

neurons emits then one spike to r which reconstitute the value 0 in the register before reaching l_k.

The reader can check that the remaining spikes in neurons d_4 and d_5 do not disturb further computations. In this case, the outdegree can also be easily reduced to two using a couple of intermediate neurons between l_i and d_1, r and d_2.

Ending a computation – module FIN (Figure 3).

When the computation in M halts, a spike reaches the neuron l_h of Π. In that moment, register 1 of M stores value n and neuron 1 of Π contains $2n + 2$ spikes. The spike emitted by l_h reaches neuron 1 (thus containing an odd number of spikes). This leads neuron 1 to fire continuously, consuming two spikes at each step. One step after receiving the spike from l_h, neuron 1 fires and one spike reaches neuron e_1 and neuron *out*. Next, neuron *out* fires and spikes for the first time. From that step on, neuron *out* simultaneously receives a couple of spikes from 1 and e_1 that do not let it fire again until one step after neuron 1 fires for the last time. When neuron 1 stops spiking, neuron *out* still receives one spike from e_1 making it fire and emitting its second and last spike (exactly n steps after the first one).

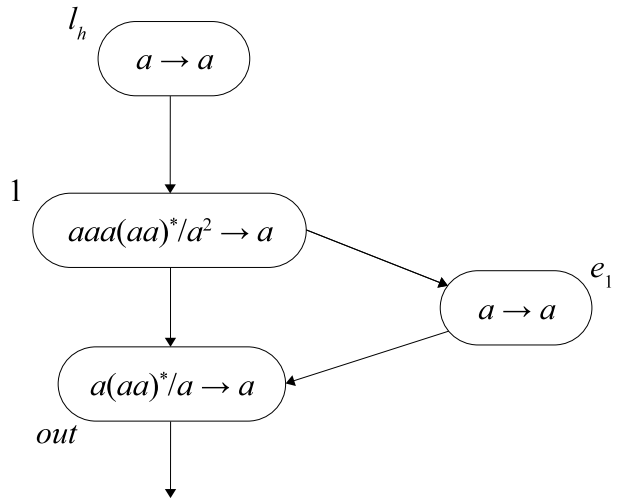


Fig. 3. The FIN module

Once the computation has ended, neuron 1 holds three spikes and neuron *out* contains $2(n-1)$ spikes. As this construction needs to leave some spikes in the system after halting, it cannot be extended to the case of the strong halting. □

4 Removing Delays and Simplifying Regular Expressions

Theorem 7.1 of [8] stated that the regular expressions used in firing rules could be simplified to the point of just using the simplest expressions over the alphabet $\{a\} : \lambda$ and a^*. In this section we consider the same problem, but removing delays simultaneously. Surprisingly, the proof construction shows that SN P systems are still universal even in that case. Moreover, we keep universality using, in each neuron, one rule of the form $a^*/a \to a$ or $(a^r \to a)$, and at most two rules $a^s \to \lambda$, with $r, s \leq 3$. (with the only exception of the non-deterministic neuron c_3 in the ADD module). Finally, we have also kept the limitation of outdegree ≤ 2 for each neuron. Comparing this result with that of Theorem 7.1 of [8], one can notice that removing delays has some computational cost in terms of other parameters, as the maximum degree of forgetting rules, the number of rules per neuron and the maximum number of spikes consumed in a rule.

Theorem 2. *$Spik_{\underline{2}}^{\beta}P_*(rule_3^*, cons_3, forg_3, dley_0, outd_2) = NRE$, where either $\beta = h$ or β is omitted.*

Proof. This proof is based on that of Theorem 7.1 from [8], trying of imitate, as long as possible, the structure and functioning of modules ADD, SUB and FIN as

well as of the dynamical register. In the proof of Theorem 7.1 [8], there exist two kinds of neurons using delays. The first one uses the delay just to slow down the emission of a spike, while the second one makes also use of the refractory period of the neuron. Remember that a neuron omits all spikes received during its refractory period. This property of neurons is then used to implement some desynchronizing circuits allowing, for instance, to decrement the dynamical register without using regular expressions that check the parity of spikes.

While eliminating delays in the first case is trivial (replacing the neuron by a chain of basic neurons with delay zero), it becomes a challenge to simulate the behavior of a neuron that makes use of its refractory period. If a neuron has delay 1 and it receives a spike in t, it fires in $t+1$, while remaining closed, and it finally spikes in $t+2$ (so it remains closed for one step). In turn, if a neuron with delay 2 receives a spike in t, it fires in $t+1$, remaining closed until $t+2$, and it finally emits a spike in $t+3$ (so it is closed during two steps).

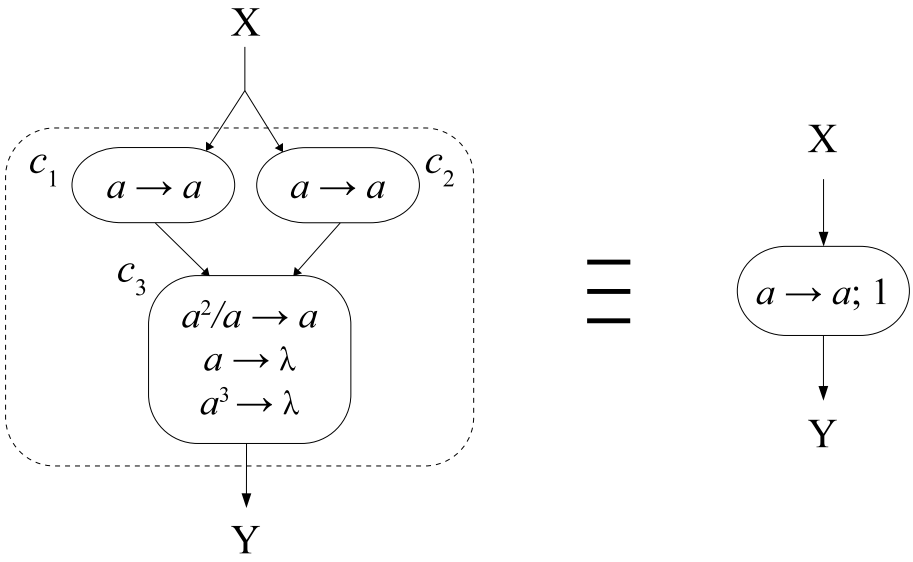


Fig. 4. Subsystem Π_{d_1} simulating a neuron with delay 1

Figure 4 shows a subsystem Π_{d_1} simulating the exact behavior of neurons with delay 1 which also use their refractory period. Let us consider X emits two spikes consecutively to Π_{d_1} in t and $t+1$. The spike received by c_1 and c_2 in t is emitted to c_3 in $t+1$ (meanwhile the second spike emitted by X reaches c_1 and c_2). Neuron c_3 fires, consuming just one of its two spikes, and spikes in $t+2$. In that moment c_1 and c_2 also spike to c_3 which now contains three spikes, which are forgotten in $t+3$ (using the rule $a^3 \to \lambda$). The reader can check that this system works also

appropriately in the trivial case of X emitting just one spike in t (c_3 then uses the rule $a \to \lambda$).

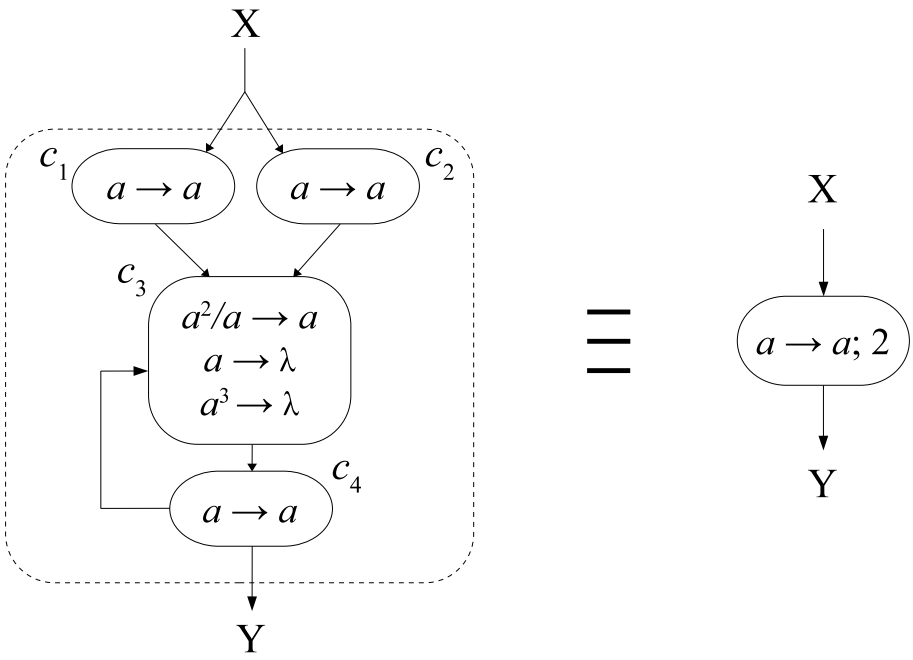


Fig. 5. Subsystem Π_{d_2} simulating a neuron with delay 2

Some more considerations have to be taken into account when simulating a neuron with delay 2 that has to remain closed during two steps. Figure 5 shows a subsystem that spikes at step $t+3$ (when it receives a spike in t) and omits any spike arriving at steps $t+1$ and/or $t+2$. Its function is analogous to that of Π_{d_1}.

We now present the dynamic register and the rest of modules where all the neurons having delays have been replaced, depending on the case, by either a chain of basic neurons or by one of the subsystems of type Π_{d_1} and Π_{d_2}. In the case of the dynamic register, neurons x, s and y do not use in any case their refractory period, so they are substituted by a trivial circuit of chained neurons with delay zero. On the other hand, neurons t and w have to be replaced by the subsystem of type Π_{d_1}. Finally, neuron r (which has delay two) is replaced by a subsystem of type Π_{d_2} as it needs to spike at every three steps whenever it contains any spike inside.

In the proof of Theorem 7.1 of [8] it is said that the dynamic register stores a number equal to the number of spikes that are continuously circulating in the close circuit $r-s-t-u$ (counting the pair of spikes simultaneously received and

later emitted by s and t as one spike). In our case, there is a slight modification in the way of representing the number stored in the dynamic register. This is due to the structural effects of replacing some neurons by subsystems Π_{d_1} and Π_{d_2}. As the reader can see, these subsystems have two input neurons c_1 and c_2 (hence the input synapses have to be doubled). Then, the need to maintain the outdegree ≤ 2 forces us to replicate some cells (with their respective synapses) in order to keep the same functionality of the original dynamic register. Figure 6 shows the aspect of the dynamic register after introducing all these changes.

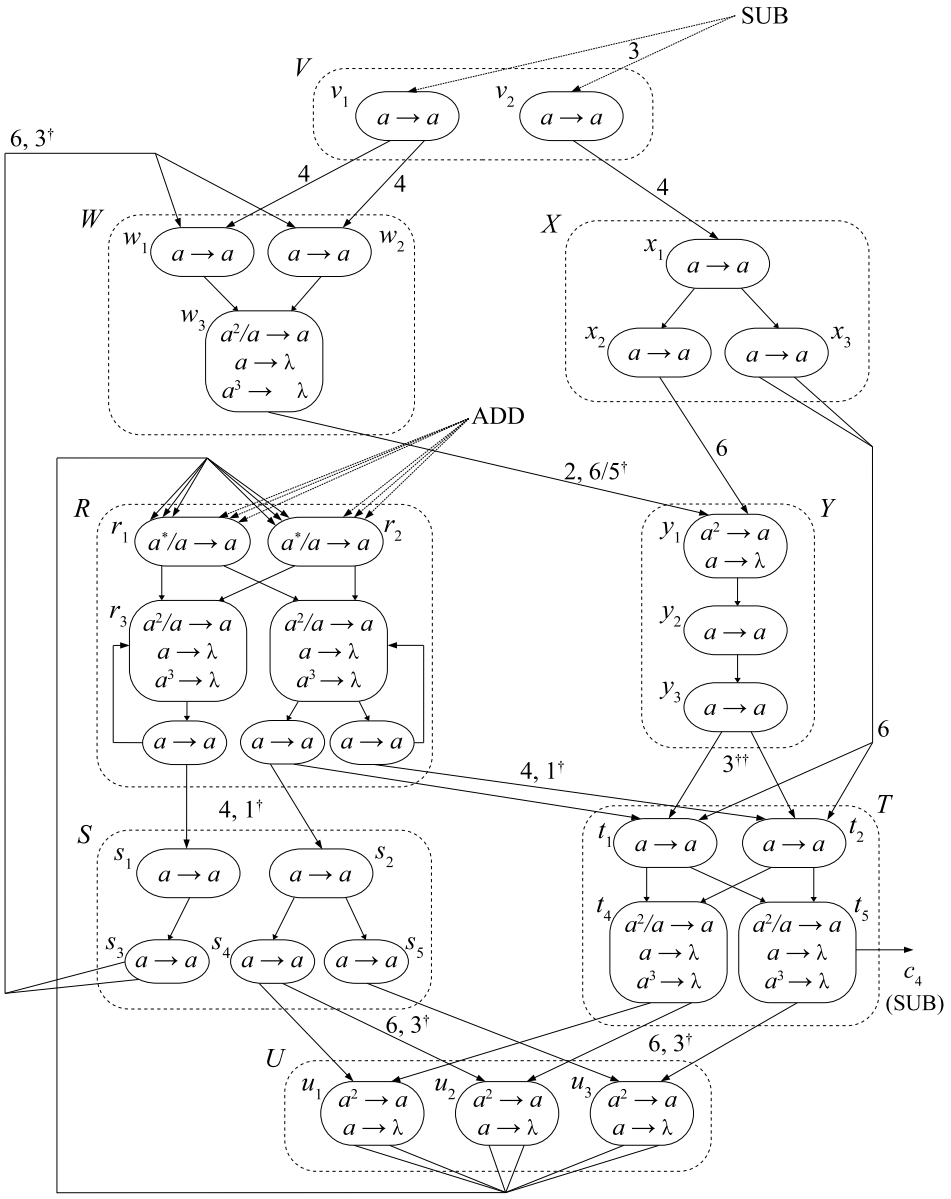
The reader can appreciate that one spike circulates from subsystem U to subsystem R (actually a group of 6 parallel spikes). It is easy to see that both r_1 and r_2 need to receive three spikes in order to have R behaving as expected (spiking every three steps as long as it holds some spike). Thus, to perform the ADD operation three spikes have also to be sent to both neurons r_1 and r_2. Subsystem R is connected to subsystems S and T, as well. Hence, when R spikes, one spike is sent simultaneously to neurons s_1, s_2, t_1 and t_2. Finally, a synapse also connects subsystems S and T with subsystem U. In this case, three spikes have to be sent from each S and T to feed the three equal neurons in U.

Hence, the computation in our dynamic register is cyclic every six steps when it stores the value $n = 1$ (all 6 spikes present at step t in R are consumed to make it spike once to S and T at $t + 3$). In turn, S and T send three spikes (one to each neuron in U) at step $t + 5$. Finally, U emits again six spikes to R at step $t + 6$ and the cycle is complete. Moreover, similarly as in the former dynamic register, the six-step computation cycle actually consists of two identical halves of three steps, when the value stored is $n > 1$. Thus, the functioning of our dynamic register is identical to that of [8], if we consider as a single spike the group of spikes emitted simultaneously from U to R, if we count as another spike every pack of six spikes stored in R and, finally, if we also consider as one spike the two ones simultaneously received and later emitted by S and T.

Simulating an ADD instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ – module ADD (Figure 7).

To avoid the use of delays, we replace the former non-deterministic neuron c_4 by a new one c_3 containing three rules. At step one, neurons l_i, l_i', l_i'' and l_i''' send one spike to c_1 and c_2 and a group of 6 spikes to the subsystem R of the dynamic register (incrementing by one the stored value). In the next step, two spikes reach neurons c_3 and c_4. Then, if rule $a^2/a \to a$ of c_3 is chosen, one spike is sent to c_6 and c_7 while another one still remains in c_3. In the following step, c_3 uses its rule $a \to a$ and two more spikes arrive to c_6 and c_7 (one from c_3 and another from c_5). This makes c_7 fire (leading the computation to l_k) while c_6 forgets its three spikes. On the other hand, if c_3 first chooses rule $a^2 \to a$, then it just emits one spike to c_6 and c_7 which will receive another one from c_5 in the next step. This situation makes c_6 fire (leading now the computation to l_j) while c_7 forgets its two spikes. Finally, it is easy to see that neurons c_8 and c_9 can be replaced by a chain of six basic zero-delay neurons.

Simulating a SUB instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ – module SUB (Figure 8).



† if the register stores a number $n > 1$
†† if the register stores a number $n \leq 1$

Fig. 6. A register with dynamical circulation of spikes without using delays

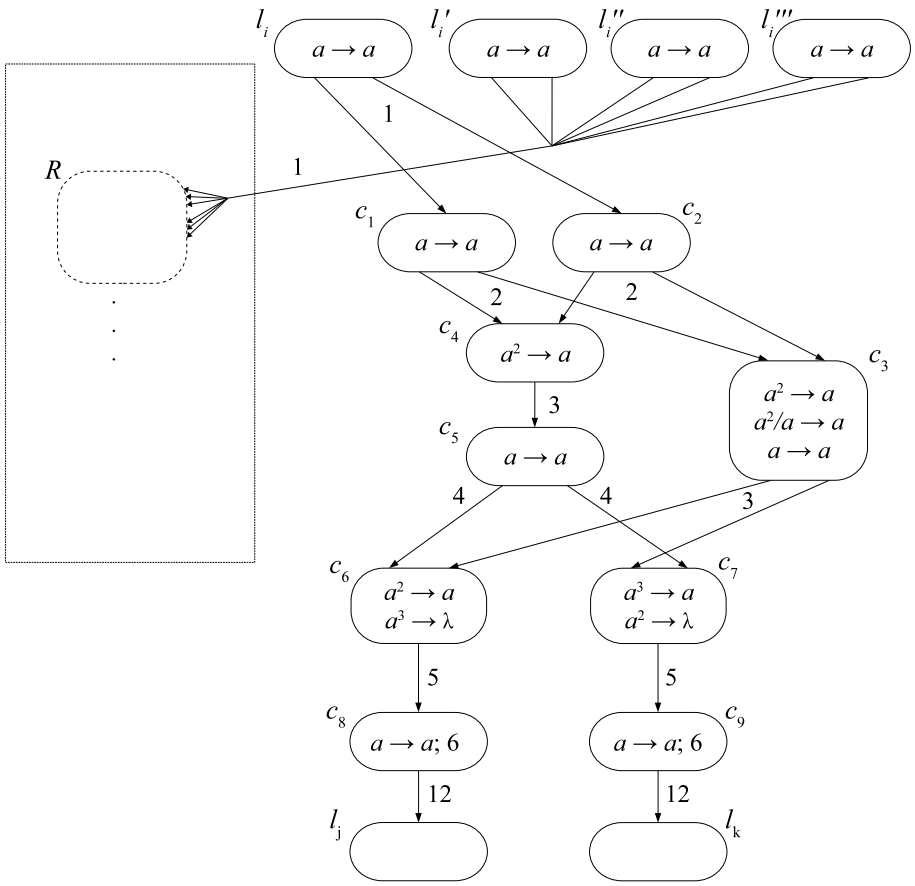


Fig. 7. Module ADD (simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$)

This module maintains the functioning of an analogous module of Theorem 7.1 with some structural changes. As neuron c_5 is the only one using the refractory period, it is replaced by a subsystem of type Π_{d_1}. The rest of neurons that have delays are replaced by a chain of basic neurons with delay 0 (except, for the sake of clarity, in the case of c_6). Some neurons are also replicated in order to maintain outdegree ≤ 2. This module is initiated when a spike is sent to neuron $l_i^{'}$. Then, two spikes are sent to subsystem V at step three and the de-synchronizing of the dynamic register starts, decrementing its value by 1. This forces T to sent a spike to c_4 at step 6. After that, $c_4^{''}$ spikes at step 8 and the computation continues by instruction l_j. If the register stored zero, then neuron $c_4^{''}$ do not spike at step 8 ($c_4^{'}$ forgets the spike emitted by c_1 at 6). Then, two spikes reach c_7 at step 11 and the computation continues by l_k. It is important to note that if there exists more than

one instruction SUB decrementing the same register, then we would need more connections from T to the neurons c_4 corresponding to these instructions.

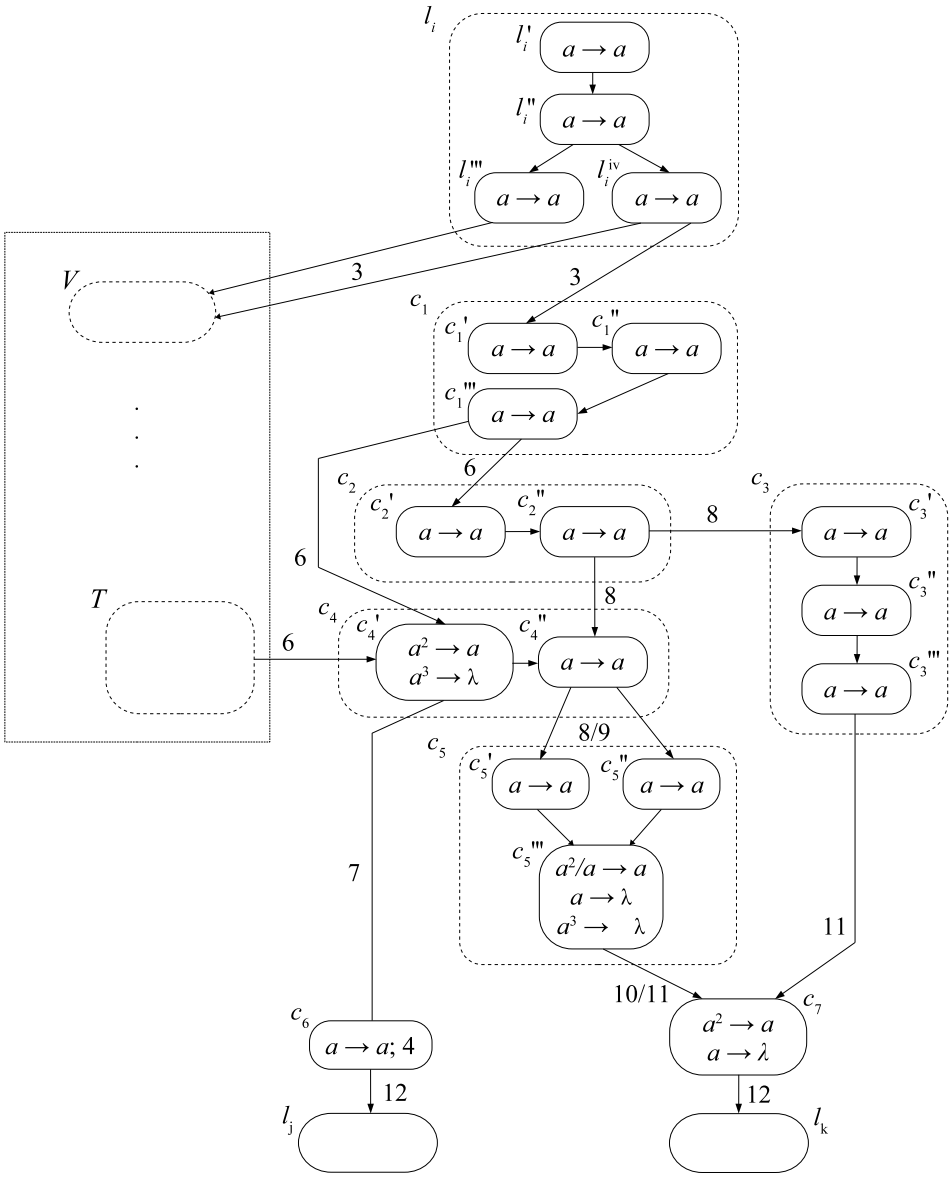


Fig. 8. Module SUB (simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$)

Ending a Computation – module FIN (Figure 9).

The module FIN has also the same behavior as that of Theorem 7.1 in [8]. However, as in the case of module SUB, some structural changes have been made to eliminate the delays of neurons l_h, c_9, c_{10}, d_9 and d_{10}. On the one hand, both neurons c_9 and d_9 are substituted by two basic cells with delay zero. On the other hand, as neurons c_{10} and d_{10} make use of their refractory period, we replace each of them by one of our subsystems of type Π_{d_1}. Finally, neurons c_2, c_7, d_7, c_8 and d_8 are duplicated to keep outdegree ≤ 2. Again, for the sake of clarity, we do not replace c_3 with its corresponding four basic neurons. □

5 Simplified Regular Expressions Revisited

Unlike some other results in [8], Theorem 7.1 mentioned above considered neither the case of strong halting, nor the case of accepting SN P systems. This sections extends this result and shows that it remains valid even if these additional restrictions are imposed. First we deal with the strong halting case.

Theorem 3. $Spik^{h}_{\underline{2}}P_*(rule^*_2, cons_2, forg_2, dley_2, outd_2) = NRE$.

Proof. Considering the strong halting case, the construction in the proof of Theorem 7.1 in [8] has to be changed slightly. Recall the assumption that all the registers except register 1 are empty at the end of computation. Under this assumption, one can verify by inspection of the above mentioned proof that the only neurons containing spikes at the moment of halting are in the module FIN. To remove these spikes, we have to release the additional restriction we stated in [8]: the rules of the form $a^r \to a; t$ and $a^s \to \lambda$ in the same neuron satisfy $s < t$. Removing this restriction allows to simplify dramatically the construction, while keeping all other properties of the normal form. The new module FIN which satisfies the strong halting condition can be found in Figure 10.

Function of the module FIN is described in Table 1. Assume that the output register 1 holds a value $n \geq 1$. Accordingly, the cycle consisting of neurons 1 and c_1 contains n spikes ($n-1$ in neuron 1 and one spike in neuron c_1). Both neurons fire at every step (except the case $n = 1$ which will be dealt with later.)

At step 1 neuron l_h receives spike and fires. At step 2 both neurons c_2 and c_3 receive spikes and start to fire at every step. Neuron c_4 receives at every step two spikes which are removed. From now on the number of spikes the cycle $1 - c_1$ decreases by one at every step. The output neuron *out* fires first time at step five. After $n + 2$ steps all the spikes in the cycle $1 - c_1$ are removed. At step $n + 3$ neuron c_2 receives no spike and does not fire. Consequently, at step $n + 4$ neuron c_4 receives only one spike and fires. Finally, at step $n + 5$, exactly n steps after its first firing, neuron *out* fires second time.

The case $n = 1$ needs a special attention. In this case neuron 1 fires at every even step and c_1 fires at every odd step. For a correct function of the module FIN, neuron l_h must receive spike at an odd step. However, this is already taken care

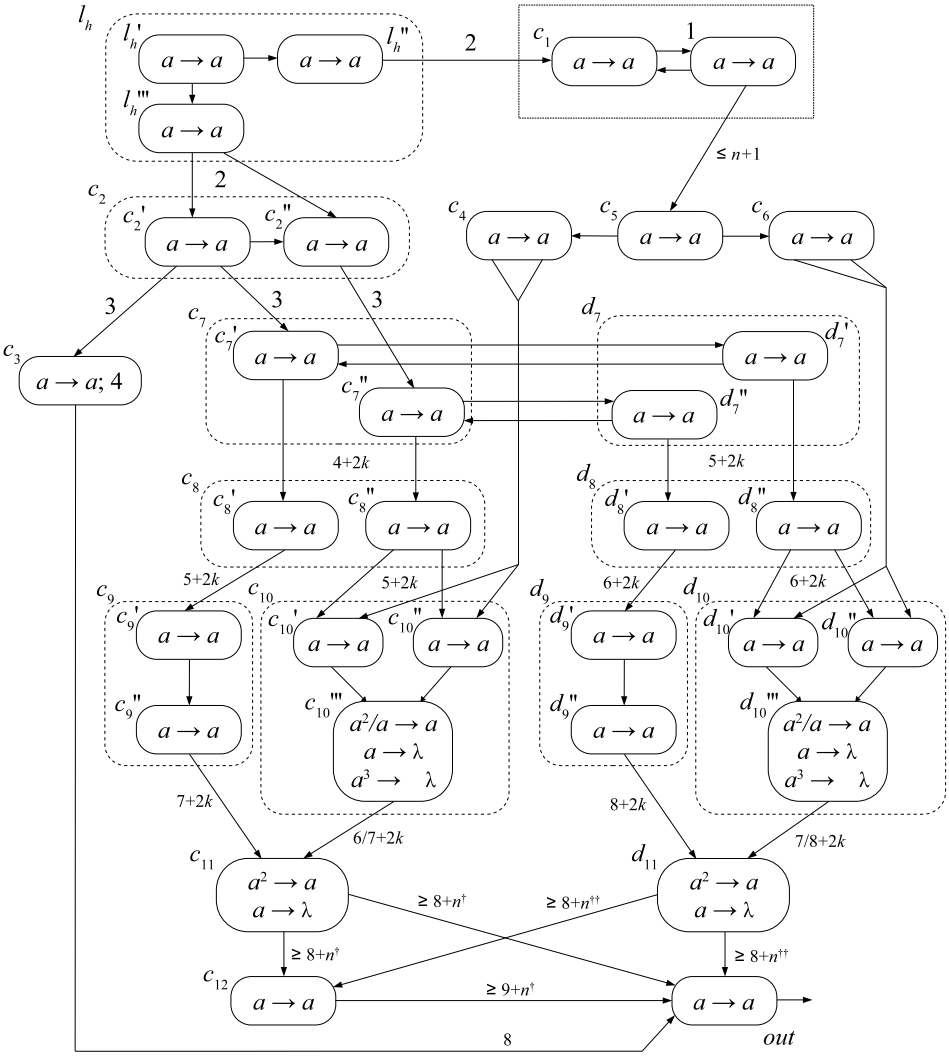


Fig. 9. Module FIN (ending the computation)

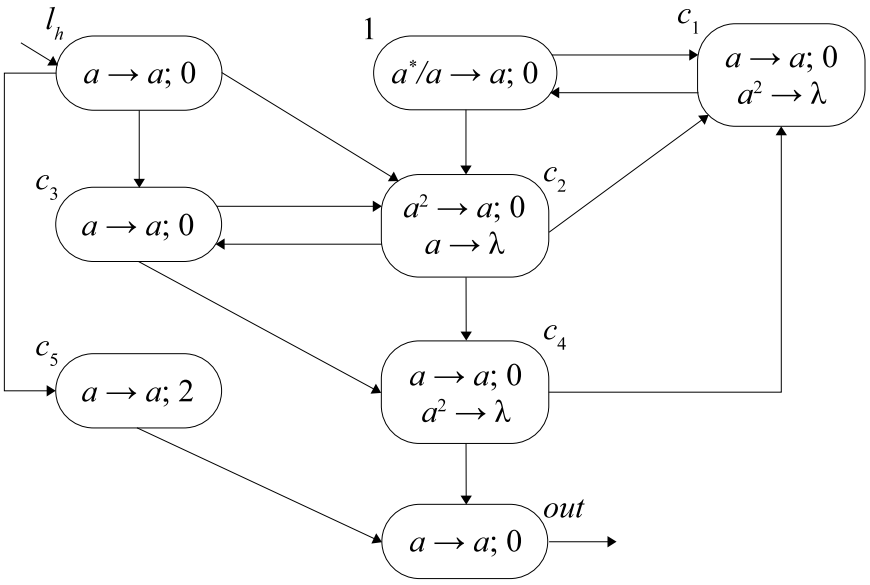


Fig. 10. Module FIN with simplified regular expressions and strong halting state

of in the proof of Theorem 7.1 in [8], as each instruction of the register machine is simulated in exactly 6 steps of the SN P system. Hence neuron l_h would receive spike at a step $6k+1$, $k \geq 0$, and the module FIN would function correctly again.

Finally, note that the module FIN of Figure 10 contains neurons with outdegree three. This version of the module was presented for the sake of simplicity, but it can be easily enhanced so that outdegree is reduced to two. The resulting diagram is shown in Figure 11. □

Another extension of the previous result in [8] mentioned above is the case of accepting SN P systems. In the accepting mode [2], the SN P system obtains an input in the form of an interval between two consecutive spikes sent from outside to the input neuron i_0. Therefore, we need a special module INPUT which translates this input value into the number of spikes present in a neuron labeled 1. Furthermore, the SN P system must behave deterministically. Both conditions can be satisfied and the resulting statement is given bellow.

Theorem 4. *$DSpik^{\beta}_{\underline{2}\,acc}P_*(rule^*_2, cons_2, forg_\alpha, dley_2, outd_2) = NRE$ where (i) $\beta = h$, $\alpha = 1$, or (ii) $\beta = \underline{h}$, $\alpha = 2$, or (iii) β is omitted and $\alpha = 1$.*

Proof. Considering the proof Theorem 7.1 in [8], we can observe that the module SUB is already deterministic. Then it remains only to "determinize" the module ADD and add a module INPUT. For the former goal, it is enough to remove the

Step / Neuron	1	2	3	4	5	...
l_h	$a \to a\,!$	—	—	—	—	...
1	$a \to a\,!$	$a \to a\,!$	$a \to a\,!$	$a \to a\,!$	$a \to a\,!$	...
(spikes)	$n-1$	$n-1$	$n-1$	$n-2$	$n-3$	...
c_1	$a \to a\,!$	$a \to a\,!$	$a^2 \to \lambda$	$a^2 \to \lambda$	$a^2 \to \lambda$	...
c_2	—	$a^2 \to a\,!$	$a^2 \to a\,!$	$a^2 \to a\,!$	$a^2 \to a\,!$	...
c_3	—	$a \to a\,!$	$a \to a\,!$	$a \to a\,!$	$a \to a\,!$	...
c_4	—	—	$a^2 \to \lambda$	$a^2 \to \lambda$	$a^2 \to \lambda$	...
c_5	—	$a \to a; 2$	—	!	—	...
out	—	—	—	—	$a \to a\,!$	...

Step / Neuron	...	$n+1$	$n+2$	$n+3$	$n+4$	$n+5$
l_h	...	—	—	—	—	—
1	...	$a \to a\,!$	—	—	$a \to a\,!$	—
(spikes)	...	1	0	0	1	0
c_1	...	$a^2 \to \lambda$	$a^2 \to \lambda$	$a \to a\,!$	—	$a^2 \to \lambda$
c_2	...	$a^2 \to a\,!$	$a^2 \to a\,!$	$a \to \lambda$	$a \to \lambda$	$a \to \lambda$
c_3	...	$a \to a\,!$	$a \to a\,!$	$a \to a\,!$	—	—
c_4	...	$a^2 \to \lambda$	$a^2 \to \lambda$	$a^2 \to \lambda$	$a \to a\,!$	—
c_5	...	—	—	—	—	—
out	...	—	—	—	—	$a \to a\,!$

Table 1. Function of the module FIN with strong halting state. Firing is denoted by !

rule $a \to a; 1$ from the neuron c_{i4} in the above mentioned module ADD, and the whole module becomes deterministic.

For the latter goal, one must construct a module INPUT which would fill-in register 1 with the number of spikes corresponding to the delay between two input spikes. However, as the module ADD works in a synchronized cycle of the length three, we have to send spikes to register 1 each three computational steps (or its multiple). Otherwise the spikes might be lost (consumed) within the module ADD. The module INPUT solving this task is presented in Figure 12.

One can observe that three steps after neuron i_0 receives the first input spike, neurons $c_3 - c_6$ start to fire at each step. Similarly, three steps after neuron i_0 receives the second input spike, neurons $c_3 - c_6$ stop firing and remove all their spikes. Therefore, neuron c_7 will receive exactly $3n$ spikes, where n is the period between the first and the second input spike. Neuron c_7 emits one spike at each step but neuron c_8 lets pass only each third spike. Therefore, neuron 1 corresponding to the input register receives spikes in steps 8, 11, 14..., and the number of spikes is exactly n.

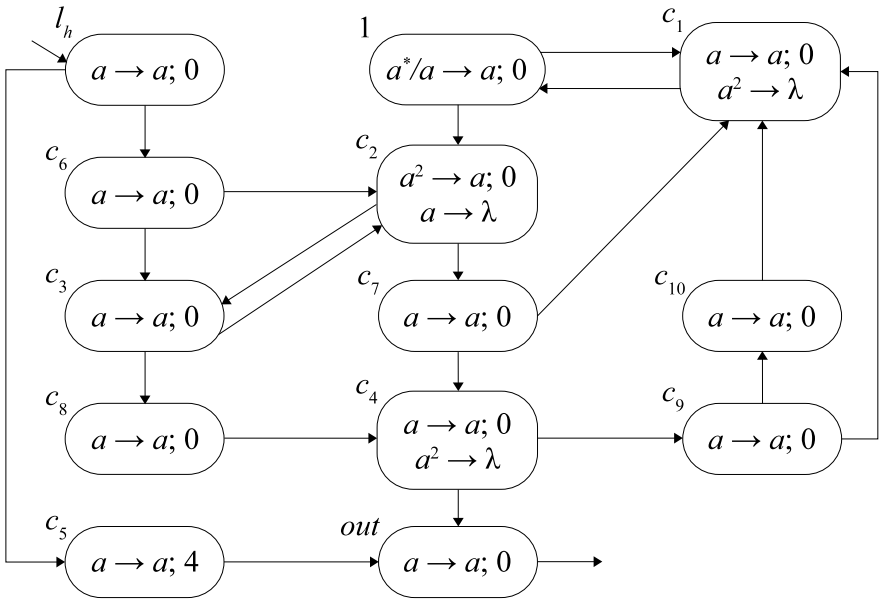


Fig. 11. Module FIN with simplified regular expressions and strong halting state, a version with outdegree two.

Finally, observe that there are no spikes left in neurons after finishing the computation. As this is also the case in the modules ADD and SUB described in the proof of Theorem 7.1 in [8] (provided that all the registers are empty), the system halts always in the strong halting state. If we do not require strong halting, the rules $a^2 \to \lambda$ in neurons $c_3 - c_6$ can be omitted. Therefore, the parameter *forg* is reduced to 1 in this case.

Therefore, the described SN P system correctly simulates a register machine in the accepting mode and the inclusion $NRE \subseteq DSpik^{\beta}_{\underline{2}acc}P_*(rule^*_2, cons_2, forg_\alpha, dley_2, outd_2)$. The converse inclusion follows by the Church-Turing thesis. □

6 Final Remarks

In this paper we have proven the universality of SN P systems even in the situations when we have eliminated more than one of its features simultaneously. Thus, this model has been found to be computationally complete 1) when using neither delays nor forgetting rules, 2) when simplifying regular expressions and eliminating delays, 3) when using simple regular expressions and the strong halting condition and 4) when using simple regular expressions in the accepting mode.

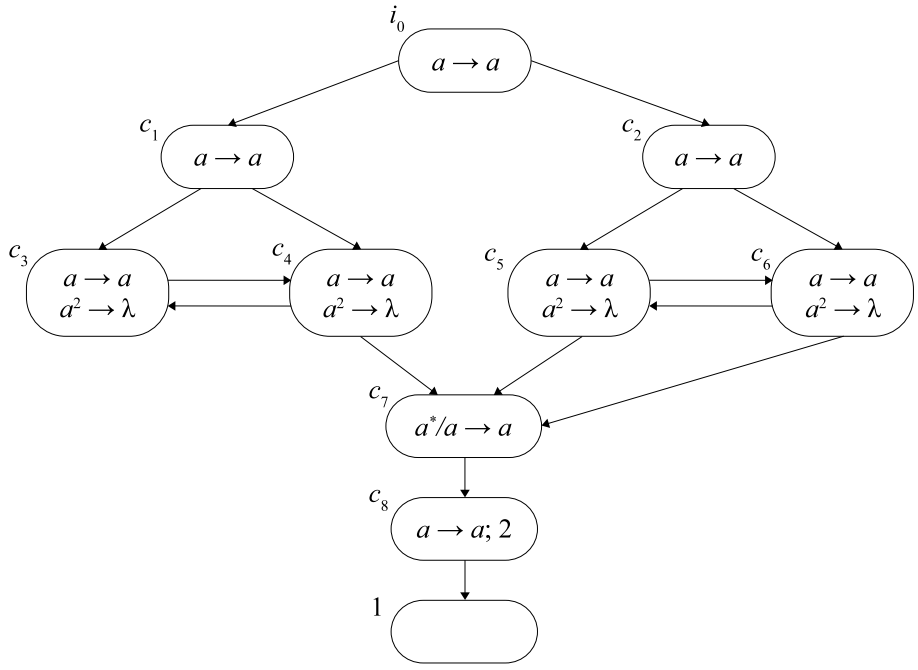


Fig. 12. The Module INPUT with simplified regular expressions.

We conjecture that in the cases 3) and 4), also delays could be removed without loss of computational universality. The case of simultaneously removing forgetting rules and simplifying regular expressions remains open but we conjecture that the universality would not be preserved in this case.

In all these results one can observe a trade-off between some other computational parameters, such as the number of neurons, the maximal number of firing rules per neuron, the complexity of regular expressions, the maximum number of spikes consumed in a firing rule or the maximal number of spikes removed in a forgetting rule. In all the above mentioned cases, however, the outdegree of neurons has been bounded by two.

It now remains an open problem whether these results concerning normal forms of SN P systems can be further improved. Would it be possible, for instance, to eliminate some more features of the model (three of them simultaneously) while keeping universality? If not, which would be the computational power of such a restricted model? Another open question would be, naturally, whether we can still achieve lower bounds for some of the computational parameters in our current proofs, as the number of rules in neurons, number of spikes consumed in one rule etc.

References

1. M.A. Gutiérrez-Naranjo et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing, Febr. 2006*. Fenix Editora, Sevilla, 2006.
2. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
3. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
4. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
5. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
6. Gh. Păun: *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.
7. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002
8. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth: Normal forms for spiking neural P systems. In [1], Vol. II, 105-136, and *Theoretical Computer Sci.*, 372, 2-3 (2007), 196–217.
9. Gh. Păun, Twenty Six Research Topics About Spiking Neural P Systems. In the present volume
10. Gh. Păun, M.J. Pérez-Jiménez, A. Salomaa: Bounding the indegree of spiking neural P systems. *TUCS Technical Report* 773, 2006.
11. The P Systems Web Page: `http://psystems.disco.unimib.it`.

A Membrane Computing Model for Ballistic Depositions

Carmen Graciani-Díaz, Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez

Research Group on Natural Computing
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
{cgdiaz,magutier,marper}@us.es

Summary. Ballistic Deposition was proposed by Vold [9] and Sutherland [8] as a model for colloidal aggregation. These early works were later extended to simulate the process of vapor deposition. In general, Ballistic Deposition models involve $(d+1)$-dimensional particles which rain down sequentially at random onto a d-dimensional substrate; when a particle arrives on the existing agglomeration of deposited particles, it sticks to the first particle it contacts, which may result in lateral growth. In this paper we present a first P system model for Ballistic Deposition with $d = 1$.

1 Introduction

Some recent discoveries on the dynamical process of surface growth have encouraged the scientific community to revisit the study of systems exhibiting rough interfaces. In Nature, there exist many examples of rough interfaces, actually, *all* surfaces in Nature can be seen as rough surfaces, since the concept of roughness is associated to the scale of observation and surfaces on Nature are far from be smooth if observed at appropriate scale.

The propagation of forest fires [5], the growth of a colony of bacteria [3] or the propagation of reaction fronts in catalyzed reactions [1] are real-world examples where the *frontier* between two media are far from being smooth. In this cases, the interfaces can be hardly modeled with Euclidean geometry and it is necessary to consider new tools in order to handle them. Moreover, in that cases, we are interested not only in the morphology of the interfaces from a static point of view, but in the dynamics of how the interface develops in time.

These dynamics can be studied from two complementary approaches:

- *Discrete approaches* where the position of each particle of the surface is well defined. This approach is getting more consideration at the atomic level in the last years due to the use of new technology as the scanning tunneling microscopy, capable of identifying not only the structure of the lattice of particles, but the position of individual atoms as well.

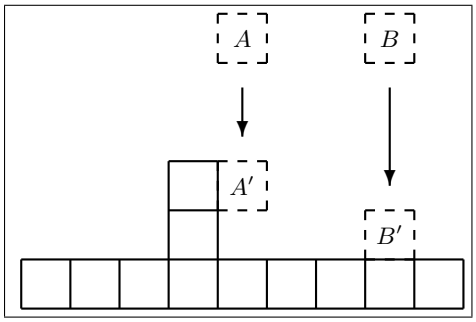


Fig. 1. Ballistic Deposition

- *Continuum approaches* view the surface on a *coarse-grained* scale, in which every property is averaged over a small volume containing many atoms. Their predictive power is limited to length scales larger than the typical inter-atom distance.

This paper is devoted to the study of a process of formation of rough surfaces called *Ballistic Deposition* (BD). To this aim, we will explore the capability of some Membrane Computing devices as tools for modeling BD in a discrete approach.

The paper is organized as follows: first the Ballistic Deposition model is briefly described. In Section 3, *deposition P systems* are presented and following this model, a P system simulating the dynamics of Ballistic Deposition is presented in Section 4. Some conclusions are presented in Section 5. The paper ends with the Bibliography and an Appendix with the proof of our main result.

2 Ballistic Deposition

In Nature, some interfaces are formed as result of a deposition process, other shrink due to erosion. A typical example of deposition process is the random fall of snowflakes on the ground floor. The randomness in the deposition process leads to a rough surface.

There exist many deposition models which try to represent different natural process. The simplest way to define such models is on a lattice where particles are deposited onto a surface oriented perpendicular to the particle trajectories, but other versions have been also investigated[1].

Ballistic Deposition (BD) was proposed by Vold [9] and Sutherland [8] as a model for colloidal aggregation. These early works were later extended to simulate the process of vapor deposition. In this model, a particle is released from a position above the surface. The particle follows a straight vertical trajectory until it reaches the surface, whereupon it sticks (see Figure 1).

[1] A good starting point for the study of depositions is [2].

In general, Ballistic Deposition models involve $(d+1)$-dimensional particles which rain down sequentially at random onto a d-dimensional substrate; when a particle arrives on the existing agglomeration of deposited particles, it sticks to the first particle it contacts, which may result in lateral growth. Many mathematical models exist in order to describe Ballistic Depositions. Here we follow M.D. Penrose in [6], where all particles are assumed identical is presented.

In Penrose's mathematical model, the substrate is $\mathbb{R}^d \times \{0\}$, identified with $\mathbb{R}^d$ or some subregion thereof. All particles are $(d+1)$-dimensional solids. Particles arrive sequentially at random positions in $\mathbb{R}^d$. When a particle arrives at a position $x \in \mathbb{R}^d$, it slides down the ray $\{x\} \times [0, \infty)$ until the particle hits a position adjacent to either the substrate or a previously deposited particle where is permanently fixed. The difference between lattice and continuum models is that in the lattice model the positions at which particle arrive are restricted to be in the integer lattice $\mathbb{Z}^d$.

Let $\mathbf{0}$ denote the origin in $\mathbb{Z}^d$. A *displacement function* is a mapping $D : \mathbb{Z}^d \to [-\infty, \infty)$ verifying:

- $D(\mathbf{0}) = 1$
- The set $\mathcal{N} = \{x \in \mathbb{Z}^d : D(x) \neq -\infty\}$ is finite but has at least two elements (one of which is the origin)

For $z \in \mathbb{Z}^d$, let $\mathcal{N}_x = \{x + y : y \in \mathcal{N}\}$ and $\mathcal{N}_x^* = \{x - y : y \in \mathcal{N}\}$. The set $\mathcal{N}$ is a neighborhood of the origin and $\mathcal{N}_x$ is a neighborhood of x. The idea of a displacement function is that if a particle arrives at $y \in \mathcal{N}_x$ then it cannot slide down the ray $y \times [0, +\infty)$ below the position at height $D(y - x)$. In this way, if $h(x, t)$ measures the height of the interface at site x at time t then

$$h(x, t+1) = max\{h(y, t) + D(x - y) : y \in \mathbb{Z}^d\}$$

since $-\infty + x = -\infty$ for all $x \in \mathbb{R}$ then

$$h(x, t+1) = max\{h(y, t) + D(x - y) : y \in \mathcal{N}_x^*\}$$

In this paper we follow the version of ballistic deposition considered in [7], the nearest neighbor model, where $\mathcal{N} = \{z \in \mathbb{Z}^d : \|z\|_1 \leq 1\}$ and the displacement function D is given by $D(x) = 0$ for $x \in \mathcal{N} - \{\mathbf{0}\}$. We are considering that the dimension of the substrate is $d = 1$ and therefore

$$\begin{aligned} h(x, t+1) &= max\{h(y, t) + D(x - y) : y \in \mathcal{N}_x^*\} \\ &= max\{h(y, t) + D(x - y) : y \in \{x - 1, x, x + 1\}\} \\ &= max\{h(x - 1, t) + D(1), h(x, t) + D(0), h(x + 1, t) + D(-1)\} \\ &= max\{h(x - 1, t) + 0, h(x, t) + 1, h(x + 1, t) + 0\} \\ &= max\{h(x - 1, t), h(x, t) + 1, h(x + 1, t)\} \end{aligned}$$

3 P systems

The chosen P systems model can be considered a subclass of tissue-like P system, since we do not consider membranes surrounding other ones, but a sequence of cells linked by communication channels. The intuition behind this structure is that each cell represent a column of the aggregate and the pieces of information needed for encoding the growth process are encoded on the multisets of objects in the cells.

In this model we use two very powerful membrane computing tools: the cooperation and the use of polarizations of the cells. Both features allow us an efficient design of P systems in order to perform the simulation. The study of minimal resources, i.e., to know whether the deposition process can be simulated without some of the used ingredients falls out of the scope of this paper.

Formally, a *deposition P system* of degree L is a construct of the form

$$\Pi = (O, \mu, env, v_1, \ldots, v_L, v_{env}, P, R)$$

where:

1. O is the alphabet of objects;
2. μ is a *cell structure* consisting of L cells bijectively labelled with $\{1, \ldots, L\}$. For all $i \in \{1, \ldots, L-1\}$ there exist an edge between the cell i and the cell $i+1$. We will also consider an edge between the cell L and the cell 1. For the sake of simplicity, we will identify the indices $L+1$ and 1; also, if a cell has polarization 0, we will omit the symbol 0.
3. env is the environment. It represents the region surrounding the cell structure μ. Some objects can be also placed in this region.
4. $v_1, \ldots, v_L, v_{env}$ are strings over O, describing the *multisets of objects* placed in the corresponding cells of μ or in the environment.
5. $P = \{0, +, -\}$ is the set of polarizations.
6. R is a finite set of *rules*, of the following forms:
   a) $[v_1 \rightarrow v_2]_i^e$ where $i \in \{1, \ldots, L\}$, $e \in P$ and v_1, v_2 are strings over O describing multisets of objects. These are *object evolution rules* associated with cells and depending only on the label and the polarization of the cell. The string v_1 has at least one object.
   b) $a[\,]_i^{e_1} \rightarrow [b]_i^{e_2}$ where $i \in \{1, \ldots, L\}$, $e_1, e_2 \in P$ and $a, b \in O$. These are *send-in rules*. An object of the environment is introduced in the membrane possibly modified. The polarization of the cell can also change.
   c) $[a]_i^{e_1} \rightarrow b[\,]_i^{e_2}$ where $i \in \{1, \ldots, L\}$, $e_1, e_2 \in P$ and $a, b \in O$. These are *send-out rules*. An object is sent out to the environment possibly modified. The polarization of the cell can also change.
   d) $[a]_i^{e_1}, [\,]_{i+1} \rightarrow [\,]_i, [b]_{i+1}^{e_2}$
   $[\,]_i, [a]_{i+1}^{e_1} \rightarrow [b]_i^{e_2}, [\,]_{i+1}$
   where $i \in \{1, \ldots, L\}$, $e_1, e_2 \in P$ and $a, b \in O$. These are *communication rules*. An object a is sent, possibly modified, to a contiguous cell.

Rules are applied according to the following principles:

- Rules are used as usual in the framework of membrane computing, that is, in a maximal parallel way. In one step, each object in a cell 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 a restriction, only one change of polarization can affect to a membrane.
- All the elements which are not involved in any of the operations to be applied remain unchanged.
- Several rules can be applied to different objects in the same cell simultaneously.
- If any rule of type (a) are used at the same time that one of type (b), (c) or (d), all rules are applied, but we will consider that the object evolution rules (a) are performed *before* the other one. This consideration is useful because the rule that sends an object across a membrane can also changes its polarization.

4 Modeling BD

In this section we will consider a BD system with L columns and we will provide a deposition P system which simulates its dynamics. Let us consider the deposition P systems of degree L, $\Pi = (O, \mu, env, v_1, \ldots, v_L, v_{env}, P, R)$ where:

- $O = \{p, c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_{n3}, c_{n4}, \alpha, x, y, z\}$
- $v_i = \emptyset$, for all $i \in \{1, \ldots, L\}$
- $v_{env} = p$

Let us consider the following sets of rules, where the index $i \in \{1, \ldots, L\}$. As remarked before, we will identify the indices $L+1$ and 1 and the indices 0 and L; and, if a cell has polarization 0, we will omit the symbol 0.

Set (A) – **Deposition rules:**

$$R^i_* \equiv p\,[\,]_i \rightarrow [c_0]^+_i$$

In the BD model, a particle is deposited on the top of a column randomly chosen. We simulate this process by these rules. A particle p in the environment is sent to one of the cells. This particle activates the cell (the polarization of the cells turns on positive) and goes into the cell as the object c_0.

Set (B) – **Rules for cells with positive polarization:**

$$\begin{array}{ll} R^i_1 \equiv [c_0 \rightarrow c_1]^+_i & R^i_4 \equiv [\,]_i, [c_2]^+_{i+1} \rightarrow [c_{n3}]^-_i, [\,]_{i+1} \\ R^i_2 \equiv [c_1 \rightarrow c_2]^+_i & R^i_5 \equiv [\,]_i, [z]^+_{i+1} \rightarrow [y]_i, [\,]_{i+1} \\ R^i_3 \equiv [y \rightarrow z\,\alpha]^+_i & \end{array}$$

The sets R^i_1, R^i_2 and R^i_3 are object evolution rules and R^i_4 and R^i_5 are communication rules. Note that the counter c_k sent into a cell i, by the particle p gives two waiting step before being sent to the cell $i-1$ transformed into c_{n3} (by the rule R^{i-1}_4). These waiting steps check the occurrence of objects y inside the cell. If any y occur, each of then evolves to $z\,\alpha$ at the same time in which c_0 evolves to c_1 and

in the following step each z is sent to the cell $i-1$ transformed into y and the polarization of the cell i changes. If this happens, the rule R_4^{i-1} is not triggered because the cell containing c_2 has not positive charge.

Set (C) – Rules for cells with negative polarization:

$$\begin{array}{ll} R_6^i \equiv [c_3 \to c_4]_i^- & R_{10}^i \equiv [c_{n3} \to c_{n4}]_i^- \\ R_7^i \equiv [c_4 \to c_5]_i^- & R_{11}^i \equiv [c_{n4} \to x\, y\, c_5]_i^- \\ R_8^i \equiv [c_5 \to c_6]_i^- & R_{12}^i \equiv [x]_i^-, [\,]_{i+1} \to [\,]_i, [z]_{i+1} \\ R_9^i \equiv [c_6]_i^- \to p\,[\,]_i & \end{array}$$

The sets R_6^i, R_7^i, R_8^i, R_{10}^i and R_{11}^i are object evolution rules, R_{12}^i are communication rules and R_9^i are send-out rules. The counter c_k goes on with the objects c_3 and c_4 or with c_{n3} and c_{n4}. In both cases, the counter reaches c_5 and c_6. The object c_6 sends to the environment an object p which will go into a cell in the next step according to the set of rules R_*^i.

Set (D) – Rules for cells with polarization zero:

$$\begin{array}{ll} R_{13}^i \equiv [c_{n4} \to c_5]_i & R_{16}^i \equiv [z \to x\,\alpha]_i \\ R_{14}^i \equiv [c_5 \to c_6]_i & R_{17}^i \equiv [x\, y \to \lambda]_i \\ R_{15}^i \equiv [c_6]_i \to p\,[\,]_i & R_{18}^i \equiv [\,]_i, [c_2]_{i+1} \to [c_3]_i^-, [\,]_{i+1} \end{array}$$

The sets R_{13}^i, R_{14}^i, R_{16}^i, and R_{17}^i are object evolution rules, R_{18}^i are communication rules and R_{15}^i are send-out rules. As in the set of rule (B), the counter c_k goes on till it reaches c_6. The object c_6 sends to the environment an object p which will go into a cell in the next step according to the set of rules R_*^i. Notice that rules of R_{17}^i are cooperative rules. Each pair of objects $\langle x, y\rangle$ which occur in a cell disappears in the next step.

4.1 Informal description of the computation

For a better understanding of the computation, let us remark that the configurations at time $8t$ with $t \in \mathbb{N}$ represent the state of a BD system after the deposition of the t-th particle. Below we will formalize this idea, but before giving a description of the computation, we provide the intuitive meaning of some of the symbols of the alphabet at time $8t$:

- p represents the particle that arrives to the substrate. When it is deposited, it disappears from the environment, then the information encoded in the cells change. When the computation inside the cells finishes, a new particle is sent to the environment and the process starts again. In this way only in the steps $8t$, there exists a particle p in the environment.
- The multiplicity of α in the cell i represent the height of the column i in the BD model.

Time	Rules	Env.	Configuration
T_0		p	$-\boxed{}_1 - \boxed{}_2 - \boxed{}_3 - \boxed{}_4 -$
T_1	R^3_*		$-\boxed{}_1 - \boxed{}_2 - \boxed{c_0}^{+}_3 - \boxed{}_4 -$
T_2	R^3_1		$-\boxed{}_1 - \boxed{}_2 - \boxed{c_1}^{+}_3 - \boxed{}_4 -$
T_3	R^3_2		$-\boxed{}_1 - \boxed{}_2 - \boxed{c_2}^{+}_3 - \boxed{}_4 -$
T_4	R^2_4		$-\boxed{}_1 - \boxed{c_{n3}}^{-}_2 - \boxed{}_3 - \boxed{}_4 -$
T_5	R^2_{10}		$-\boxed{}_1 - \boxed{c_{n4}}^{-}_2 - \boxed{}_3 - \boxed{}_4 -$
T_6	R^2_{11}		$-\boxed{}_1 - \boxed{x\,y\,c_5}^{-}_2 - \boxed{}_3 - \boxed{}_4 -$
T_7	R^2_8, R^2_{12}		$-\boxed{}_1 - \boxed{y\,c_6}_2 - \boxed{z}_3 - \boxed{}_4 -$
T_8	R^2_{15}, R^3_{16}	p	$-\boxed{}_1 - \boxed{y}_2 - \boxed{x\,\alpha}_3 - \boxed{}_4 -$

Fig. 2. Table with the first steps

Finally, the remaining objects inside the cells at time $8t$ are of type x and y. The objects x or y inside the cell i represent the difference of height between the cell i and the cell $i+1$.

- The multiplicity of x in the cell i represent how many units is the column i higher than column $i+1$ in the BD model.
- The multiplicity of y in the cell i represent how many units is the column i lower than column $i+1$ in the BD model.

From the previous description we have that at time $8t$, we can find inside a cell objects x, y or none of them, but we will never find both simultaneously.

Tables 2 and 3 show an example of evolution of a simple BD system with four columns where four particles have been deposited sequentially on the columns 3, 2, 2 and 1. Notice that the configurations at times $8t$ with $t \in \{0, \ldots, 4\}$ represent the *surface* of the BD model after the fall of the t-th particle (Fig. 4).

We finish this section by formally showing that this deposition P system of degree L simulates the ballistic deposition on a substrate with L columns. In this way we need the definition of *representative configuration*. The idea behind the

Time	Rules	Env.	Configuration
T_9	R^2_*		$-[\ ]_1 - [y\,c_0]^+_2 - [x\,\alpha]_3 - [\ ]_4 -$
T_{10}	R^1_2, R^2_3		$-[\ ]_1 - [z\,\alpha\,c_1]^+_2 - [x\,\alpha]_3 - [\ ]_4 -$
T_{11}	R^2_2, R^1_5		$-[y]_1 - [\alpha\,c_2]_2 - [x\,\alpha]_3 - [\ ]_4 -$
T_{12}	R^1_{18}		$-[y\,c_3]^-_1 - [\alpha]_2 - [x\,\alpha]_3 - [\ ]_4 -$
T_{13}	R^1_6		$-[y\,c_4]^-_1 - [\alpha]_2 - [x\,\alpha]_3 - [\ ]_4 -$
T_{14}	R^1_7		$-[y\,c_5]^-_1 - [\alpha]_2 - [x\,\alpha]_3 - [\ ]_4 -$
T_{15}	R^1_8		$-[y\,c_6]^-_1 - [\alpha]_2 - [x\,\alpha]_3 - [\ ]_4 -$
T_{16}	R^1_9	p	$-[y]_1 - [\alpha]_2 - [x\,\alpha]_3 - [\ ]_4 -$
...	...	...	$-[\ldots]_1 - [\ldots]_2 - [\ldots]_3 - [\ldots]_4 -$
T_{24}	...	p	$-[y^2]_1 - [x\,\alpha^2]_2 - [\alpha\,x]_3 - [\ ]_4 -$
...	...	...	$-[\ldots]_1 - [\ldots]_2 - [\ldots]_3 - [\ldots]_4 -$
T_{32}	...	p	$-[\alpha^2]_1 - [x\,\alpha^2]_2 - [x\,\alpha]_3 - [y^2]_4 -$

Fig. 3. Table with the first steps (Cont.)

definition is quite intuitive. Along the computation, some of the configurations have no meaning with respect to the deposition process, there are merely auxiliary steps of the computation. Only some of the configuration represent states of the aggregate in the deposition process. Such configurations will be called *representative configurations*

We will denote by C_t the configuration of the P systems at time t, by $C_t(i)$ the multiset of objects at the cell labelled by i at time t and by $|C_t(i)|_a$ the multiplicity of the object a in $C_t(i)$.

Definition 1. *Let C_t be a configuration of a deposition P systems of degree L at time t. We will say that C_t is representative if for all $i \in \{1, \ldots, L\}$,*

- *Only objects α, x and y occur inside the cells and they polarization* 0

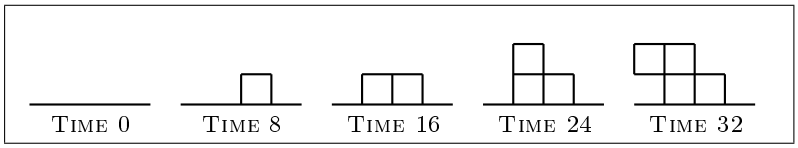


Fig. 4. Example

- $C_t(env) = \{p\}$
- *If* $|C_t(i)|_\alpha \geq |C_t(i+1)|_\alpha$ *then*
  - $|C_t(i)|_x = |C_t(i)|_\alpha - |C_t(i+1)|_\alpha$
  - $|C_t(i)|_y = 0$
- *If* $|C_t(i)|_\alpha < |C_t(i+1)|_\alpha$ *then*
  - $|C_t(i)|_y = |C_t(i+1)|_\alpha - |C_t(i)|_\alpha$
  - $|C_t(i)|_x = 0$

Finally, the next theorem claims that the sequence of configurations at times $0, 8, 16, \ldots, 8t, \ldots$ represent the states of an aggregate with a ballistic deposition process.

Theorem 1. *For all* $t \in \mathbb{N}$, C_{8t} *is a representative configuration and, if* $i \in \{1, \ldots, L\}$ *is the chosen cell for depositing a new particle, then*

- $|C_{8t+8}(i)|_\alpha = max\{|C_{8t}(i-1)|_\alpha, |C_{8t}(i)|_\alpha + 1, |C_{8t}(i+1)|_\alpha\}$
- *For all* $j \in \{1, \ldots, L\}$, $i \neq j$, $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$

Proof. See Appendix

5 Conclusions

Understanding how Nature works involves experimental observation and theoretical modeling. This paper is a contribution to the theoretical modeling of a particular case of one of the most interesting process in Physics: the dynamical evolution of the frontier between two different media. In this paper, the chosen model has been Ballistic Deposition, but many other deposition processes from Physics, Chemistry and Biology can be also modeled by using similar techniques.

On the other hand, Membrane Computing techniques had been used for studying problems from many different areas, since Linguistics or Complexity Theory to Computer Graphics or Cancer Modeling[2], but this is the first time that P systems are used to model deposition processes.

This paper can be extended in several ways. One of them is to extend the study to more dimensions, i.e., to consider the particles as 3D solids falling down

[2] See [4] for details.

onto a 2D surface. Other possible research line is to develop computer software which simulates Ballistic Depositions according with the Membrane Computing techniques presented in this paper and of course, a final research line is to follow this study by modeling other deposition models.

Acknowledgment

The authors wish acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excellence TIC-581 of the Junta de Andalucía.

References

1. E.V. Albano: Displacement of inactive phases by the reactive regime in a lattice gas model for a dimer-monomer irreversible surface reaction. *Physical Review E*, 55 (1997), 7144–7152.
2. A.-L. Barabási, H.E. Stanley: *Fractal Concepts in Surface Growth*. Cambridge University Press, 1995.
3. E. Ben-Jacob, O. Schochet, A. Tenenbaum, I. Cohen, A. Czirok, T. Vicsek: Generic modelling of cooperative growth patterns in bacterial colonies. *Nature*, 368 (1994), 46–49.
4. G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez (Eds.) *Applications of Membrane Computing*. Springer, 2006.
5. S. Clar, B. Drossel, F. Schwabl: Forest fires and other examples of self-organized criticality. *Journal of Physics: Condensed Matter*, 8 (1996), 6803–6824.
6. M.D. Penrose: Growth and roughness of the interface forr ballistic deposition. `arXiv:math.PR/0608540` (2006).
7. T. Seppäläinen: Strong law of large numbers for the interface in ballistic deposition. *Annales de L'Institut Henri Poincaré (B) Probabilités et Statistiques*, 36 (2000), 691-735.
8. D.N. Sutherland: Comments on Vold's Simulation of Folc Formation. *Journal of Colloid and Interface Science*, 22 (1966), 300-302.
9. M.J. Vold: A Numerical Approach to the Problem of Sediment Volume. *Journal of Colloid Science*, 14 (1959), 168–174.

6 Appendix

Proof of the theorem 1.

Proof. The proof is by induction on $t \in \mathbb{N}$. The key idea of the proof is to notice that the P system is deterministic with the only exception of the set of rules $R^* \equiv p\,[\,]_i \rightarrow [c_0]_i^+$ for $i \in \{1, \ldots, L\}$. This set of rules represent the non-deterministic choice of a cell in order to deposit a new particle.

In the initial configuration C_0 all cells are empty and have polarisation 0; also, $C_0(env) = \{p\}$, then it is a representative configuration.
Let us suppose that i is the chosen cell in the non-deterministic step. The first configurations are

$C_0(i) = \{\,\} \xrightarrow{R^i_*} C_1(i) = \{c_0\}^+ \xrightarrow{R^i_1} C_2(i) = \{c_1\}^+ \xrightarrow{R^i_2} C_3(i) = \{c_2\}^+$

$C_k(j) = \emptyset$, for all $j \in \{1, \ldots, i-1, i+1, \ldots, L, env\}$ and $k \in \{1, 2, 3\}$ as no rules affect to them.

As the cell i has positive electrical charge in C_3, then we apply the rule $[\,]_{i-1}, [c_2]^+_i \to [c_{n3}]^-_{i-1}, [\,]_i$ obtaining

$C_4(i-1) = \{c_{n3}\}^-$

$C_4(j) = \emptyset$ for all $j \in \{1, \ldots, i-2, i, \ldots, L, env\}$

Hence

$C_4(i-1) = \{c_{n3}\}^- \xrightarrow{R^{i-1}_{10}} C_5(i-1) = \{c_{n4}\}^- \xrightarrow{R^{i-1}_{11}} C_6(i-1) = \{x\, y\, c_{n5}\}$

$C_k(j) = \emptyset$ for all $j \in \{1, \ldots, i-2, i, \ldots, L, env\}$ and $k \in \{5, 6\}$ as no rules affect to them.

At this step rules $[c_5 \to c_6]^-_{i-1}$ and $[x]^-_{i-1}, [\,] \to [\,]_{i-1}[z]_i$ are applied simultaneously

$C_7(i-1) = \{c_6\, y\}$

$C_7(i) = \{z\}$

$C_7(j) = \emptyset$ for all $j \in \{1, \ldots, i-2, i+1, \ldots, L, env\}$

Finally, rules $[c_6]_{i-1} \to p\,[\,]_{i-1}$ and $[z \to x\,\alpha]_i$ are applied

$C_8(i-1) = \{y\}$

$C_8(i) = \{x\,\alpha\}$

$C_8(env) = \{p\}$

$C_8(j) = \emptyset$ for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$

In configuration C_8, all the cells has polarisation 0 and only objects x, y and α occur inside them, $C_8(env) = \{p\}$ and

$|C_8(i-1)|_\alpha = 0 < |C_8(i)|_\alpha = 1$

$|C_8(i-1)|_y = 1 = |C_8(i)|_\alpha - |C_8(i-1)|_\alpha$

$|C_8(i-1)|_x = 0$

$|C_8(i)|_\alpha = 1 \geq |C_8(i+1)|_\alpha = 0$

$|C_8(i)|_x = 1 = |C_8(i)|_\alpha - |C_8(i+1)|_\alpha$

$|C_8(i)|_y = 0$

$C_8(j) = \emptyset$ for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$

Hence, C_8 is a representative configuration. We also have that

$|C_8(j)|_\alpha = |C_0(j)|_\alpha$ for $j \in \{1, \ldots, i-1, i+1, \ldots, L\}$

$|C_8(i)|_\alpha = max\{|C_0(i-1)|_\alpha, |C_0(i)|_\alpha + 1, |C_0(i+1)|_\alpha\} = 1$

$\boxed{\text{t} \to \text{t+1}}$

Let us suppose that C_{8t} is a representative configuration and i is the cell chosen non-deterministically. We will prove that C_{8t+8} is also a representative configuration and

$$|C_{8t+8}(i)|_\alpha = max\{|C_{8t}(i-1)|_\alpha, |C_{8t}(i)|_\alpha + 1, |C_{8t}(i+1)|_\alpha\}$$

$$\text{For all } j \in \{1,\dots,L\}, i \neq j, |C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$$

We will consider five possible cases depending on the relation among $|C_{8t}(i-1)|_\alpha$, $|C_{8t}(i)|_\alpha$ and $|C_{8t}(i+1)|_\alpha$.

Case 1: $|C_{8t}(i-1)|_\alpha > |C_{8t}(i)|_\alpha \geq |C_{8t}(i+1)|_\alpha$
Case 2: $|C_{8t}(i-1)|_\alpha > |C_{8t}(i+1)|_\alpha > |C_{8t}(i)|_\alpha$
Case 3: $|C_{8t}(i+1)|_\alpha \geq |C_{8t}(i-1)|_\alpha > |C_{8t}(i)|_\alpha$
Case 4: $|C_{8t}(i)|_\alpha \geq |C_{8t}(i-1)|_\alpha$ and $|C_{8t}(i)|_\alpha \geq |C_{8t}(i+1)|_\alpha$
Case 5: $|C_{8t}(i+1)|_\alpha > |C_{8t}(i)|_\alpha \geq |C_{8t}(i-1)|_\alpha$

The proof is made by inspection of these cases.

Case 1: Let us suppose that

$$|C_{8t}(i-1)|_\alpha > |C_{8t}(i)|_\alpha \geq |C_{8t}(i+1)|_\alpha$$

In this case $|C_{8t}(i-1)|_x = |C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha > 0$, $|C_{8t}(i)|_x = |C_{8t}(i)|_\alpha - |C_{8t}(i+1)|_\alpha \geq 0$ and $|C_{8t}(i-1)|_y = |C_{8t}(i)|_y = 0$.
Also, $max\{|C_{8t}(i-1)|_\alpha, |C_{8t}(i)|_\alpha + 1, |C_{8t}(i+1)|_\alpha\} = |C_{8t}(i-1)|_\alpha$. We will prove that C_{8t+8} is a representative configuration, $|C_{8t+8}(i)|_\alpha = |C_{8t}(i-1)|_\alpha$ and $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$ for $j \in \{1,\dots,i-1,i+1,\dots,L\}$
As no y are present in cell i,

$C_{8t}(i) \xrightarrow{R^i_*} C_{8t+1}(i) = C_{8t}(i) \cup \{c_0\}^+ \xrightarrow{R^i_1} C_{8t+2}(i) = C_{8t}(i) \cup \{c_1\}^+ \xrightarrow{R^i_2} C_{8t+3}(i) = C_{8t}(i) \cup \{c_2\}^+$

$C_{8t+k}(j) = C_{8t}(j)$, for all $j \in \{1,\dots,i-1,i+1,\dots,L\}$ as no rules affect to them and $C_{8t+k}(env) = \emptyset$ for $k \in \{1,2,3\}$.
As cell i has positive electrical charge in C_{8t+3}, then we apply the rule $[\,]_{i-1}, [c_2]^+_i \to [c_{n3}]^-_{i-1}, [\,]_i$ obtaining

$C_{8t+4}(i-1) = C_{8t}(i-1) \cup \{c_{n3}\}^-$

$C_{8t+4}(j) = C_{8t}(j)$ for all $j \in \{1,\dots,i-2,i,\dots,L\}$ and $C_{8t+4}(env) = \emptyset$

At this step rules $[c_{n3} \to c_{n4}]^-_{i-1}$ and $[x]^-_{i-1}, [\,] \to [\,]_{i-1}[z]_i$ are applied simultaneously and therefore all the copies of x in cell $i-1$ are sent into cell i transformed into copies of z. Hence

$C_{8t+5}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\, c_{n4}\}$

$C_{8t+5}(i) = C_{8t}(i) \cup \{z^{|C_{8t}(i-1)|_x}\}$

$C_{8t+5}(j) = C_{8t}(j)$ for all $j \in \{1,\dots,i-2,i+1,\dots,L\}$ and $C_{8t+5}(env) = \emptyset$

Now, rules $[c_{n4} \to c_5]_{i-1}$ and $[z \to x\,\alpha]_i$ can be applied, then

$C_{8t+6}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\, c_5\}$

$C_{8t+6}(i) = C_{8t}(i) \cup \{x^{|C_{8t}(i-1)|_x}\, \alpha^{|C_{8t}(i-1)|_x}\} =$
$= \{x^{|C_{8t}(i)|_x + |C_{8t}(i-1)|_x}\, \alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i-1)|_x}\}$ as $|C_{8t}(i-1)|_y = 0$

$C_{8t+6}(j) = C_{8t}(j)$ for all $j \in \{1,\dots,i-2,i+1,\dots,L\}$ and $C_{8t+6}(env) = \emptyset$

After that,

$C_{8t+6}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\, c_5\} \xrightarrow{R^{i-1}_8} C_{8t+7}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\, c_6\}$

$C_{8t+7}(i) = C_{8t+6}(i)$

$C_{8t+7}(j) = C_{8t}(j)$ for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+7}(env) = \emptyset$

Next, the rule $[c_6]_{i-1} \to p\,[\,]_{i-1}$ is applied,

$C_{8t+8}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\}$

$C_{8t+8}(i) = C_{8t+6}(i) = \{x^{|C_{8t}(i)|_x + |C_{8t}(i-1)|_x}\ \alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i-1)|_x}\}$

$C_{8t+8}(j) = C_{8t}(j)$ for all $j \in \{1, \ldots, i-1, i+1, \ldots, L\}$ and $C_{8t+8}(env) = \{p\}$.

Note that $|C_{8t+8}(i)|_\alpha = |C_{8t}(i)|_\alpha + |C_{8t}(i-1)|_x$ and by hypothesis $|C_{8t}(i-1)|_x = |C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha$ so $|C_{8t+8}(i)|_\alpha = |C_{8t}(i-1)|_\alpha = |C_{8t+8}(i-1)|_\alpha$, and that there is no x or y in $C_{8t+8}(i-1)$.

Finally as in this case $|C_{8t+8}(i)|_\alpha = |C_{8t}(i-1)|_\alpha > |C_{8t}(i)|_\alpha$ and $|C_{8t}(i)|_\alpha \geq |C_{8t}(i+1)|_\alpha = |C_{8t+8}(i+1)|_\alpha$, then

$$|C_{8t+8}(i)|_\alpha > |C_{8t+8}(i+1)|_\alpha$$

and

$$\begin{aligned} |C_{8t+8}(i)|_x &\overset{(1)}{=} |C_{8t}(i)|_x + |C_{8t}(i-1)|_x \\ &\overset{(2)}{=} (|C_{8t}(i)|_\alpha - |C_{8t}(i+1)|_\alpha) + |C_{8t}(i-1)|_x \\ &\overset{(3)}{=} (|C_{8t}(i)|_\alpha + |C_{8t}(i-1)|_x) - |C_{8t+8}(i+1)|_\alpha \\ &\overset{(4)}{=} |C_{8t+8}(i)|_\alpha - |C_{8t+8}(i+1)|_\alpha \end{aligned}$$

The equality (1) holds by the explicit description obtained for $C_{8t+8}(i)$. The equality (2) holds by hypothesis of induction. The third equality holds because for all $j \in \{1, \ldots, L\}$, $i \neq j$, $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$, in particular for $j = i+1$ and the last equality holds by the explicit description obtained for $C_{8t+8}(i)$.

In order to finish the proof that C_{8t+8} is a representative configuration we need to see that $|C_{8t+8}(i)|_y = 0$ but this holds by the explicit description obtained for $C_{8t+8}(i)$.

Case 2: Let us suppose that

$$|C_{8t}(i-1)|_\alpha > |C_{8t}(i+1)|_\alpha > |C_{8t}(i)|_\alpha$$

In this case $|C_{8t}(i-1)|_x = |C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha > 0$, $|C_{8t}(i)|_x = |C_{8t}(i-1)|_y = 0$ and $|C_{8t}(i)|_y = |C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha > 0$.

Also, $max\{|C_{8t}(i-1)|_\alpha, |C_{8t}(i)|_\alpha + 1, |C_{8t}(i+1)|_\alpha\} = |C_{8t}(i-1)|_\alpha$. We will prove that C_{8t+8} is a representative configuration, $|C_{8t+8}(i)|_\alpha = |C_{8t}(i-1)|_\alpha$ and $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$ for $j \in \{1, \ldots, i-1, i+1, \ldots, L\}$

As y is present in cell i,

$C_{8t}(i) \xrightarrow{R^i_*} C_{8t+1}(i) = C_{8t}(i) \cup \{c_0\}^+ \xrightarrow{R^i_1, R^i_3} C_{8t+2}(i) = \{\alpha^{|C_{8t}(i)|_\alpha}\} \cup \{z^{|C_{8t}(i)|_y}\ \alpha^{|C_{8t}(i)|_y}\ c_1\}^+$

$C_{8t+k}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-1, i+1, \ldots, L\}$ as no rules affect to them and $C_{8t+k}(env) = \emptyset$ for $k \in \{1, 2\}$.

As cell i has positive electrical charge in C_{8t+2}, then we apply the rules $[\,]_{i-1}, [z]^+_i \to [y]_{i-1}, [\,]_i$ and $[c_1 \to c_2]^+_i$ obtaining

$C_{8t+3}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\}$

$C_{8t+3}(i) = \{\alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\, c_2\}$

$C_{8t+3}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+3}(env) = \emptyset$.

In this situation $|C_{8t+3}(i-1)|_x = |C_{8t}(i-1)|_x > 0$ and $|C_{8t+3}(i-1)|_y = |C_{8t}(i)|_y > 0$ so the rule $[x\, y \to \lambda]_{i-1}$ can be applied. In order to compute the number of copies of x and y that remain in cell $i-1$ after the application of this rule we consider that

$$\begin{aligned} |C_{8t}(i-1)|_x &\overset{(1)}{=} |C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha \\ &\overset{(2)}{>} |C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha \\ &\overset{(3)}{=} |C_{8t}(i)|_y \end{aligned}$$

The equality (1) holds because C_{8t} is a representative configuration and in this case $|C_{8t}(i-1)|_\alpha > |C_{8t}(i)|_\alpha$. The inequality (2) holds because in this case $|C_{8t}(i-1)|_\alpha > |C_{8t}(i+1)|_\alpha$ and finally, the last equality holds by the definition of representative configuration and $|C_{8t}(i+1)|_\alpha > |C_{8t}(i)|_\alpha$.

The rule $[\,]_{i-1}, [c_2]_i^+ \to [c_3]_{i-1}^-, [\,]_i$ can also be applied. Therefore,

$C_{8t+4}(i-1) = \{x^{|C_{8t}(i-1)|_x - |C_{8t}(i)|_y}\, \alpha^{|C_{8t}(i-1)|_\alpha}\, c_3\}^-$

$C_{8t+4}(i) = \{\alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\}$

$C_{8t+4}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+4}(env) = \emptyset$.

Since polarisation of cell $i-1$ is now negative, rules $[x]_{i-1}^-, [\,]_i \to [\,]_{i-1}, [z]_i$ and $[c_3 \to c_4]_{i-1}^-$ can be applied and all the copies of x from cell $i-1$ are sent into the cell i transformed into copies of z. Hence

$C_{8t+5}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\, c_4\}$

$C_{8t+5}(i) = \{z^{|C_{8t}(i-1)|_x - |C_{8t}(i)|_y}\, \alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\}$

$C_{8t+5}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+5}(env) = \emptyset$.

After that, rules $[c_{n4} \to c_5]_i$ and $[z \to x\, \alpha]_i$ can be applied, then

$C_{8t+6}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\, c_5\}$

$$\begin{aligned} C_{8t+6}(i) &= \{x^{|C_{8t}(i-1)|_x - |C_{8t}(i)|_y}\, \alpha^{|C_{8t}(i-1)|_x - |C_{8t}(i)|_y + |C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\} = \\ &= \{x^{|C_{8t}(i-1)|_x - |C_{8t}(i)|_y}\, \alpha^{|C_{8t}(i-1)|_x + |C_{8t}(i)|_\alpha}\} \end{aligned}$$

$C_{8t+6}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+6}(env) = \emptyset$.

In next step only the rule $[c_5 \to c_6]_{i-1}$ can be applied and we obtain

$C_{8t+7}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\, c_6\}$

$C_{8t+7}(i) = C_{8t+6}(i)$

$C_{8t+7}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+7}(env) = \emptyset$.

Next, the rule $[c_6]_{i-1} \to p\,[\,]_{i-1}$ is applied

$C_{8t+8}(i-1) = \{\alpha^{|C_{8t}(i-1)|_\alpha}\}$

$C_{8t+8}(i) = C_{8t+6}(i) = \{x^{|C_{8t}(i-1)|_x - |C_{8t}(i)|_y}\, \alpha^{|C_{8t}(i-1)|_x + |C_{8t}(i)|_\alpha}\}$

$C_{8t+8}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+8}(env) = \{p\}$.

Note that $|C_{8t+8}(i)|_\alpha = |C_{8t}(i-1)|_x + |C_{8t}(i)|_\alpha$ and by hypothesis $|C_{8t}(i-1)|_x = |C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha$ so $|C_{8t+8}(i)|_\alpha = |C_{8t}(i-1)|_\alpha = |C_{8t+8}(i-1)|_\alpha$, and that there is no x or y in $C_{8t+8}(i-1)$.

Finally as in this case $|C_{8t}(i-1)|_\alpha > |C_{8t}(i+1)|_\alpha = |C_{8t+8}(i+1)|_\alpha$ then

$$|C_{8t+8}(i)|_\alpha > |C_{8t+8}(i+1)|_\alpha$$

and

$$\begin{aligned}|C_{8t+8}(i)|_x &\overset{(1)}{=} |C_{8t}(i-1)|_x - |C_{8t}(i)|_y = \\ &\overset{(2)}{=} (|C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha) - (|C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha) = \\ &\overset{(3)}{=} |C_{8t}(i-1)|_\alpha - |C_{8t}(i+1)|_\alpha \\ &\overset{(4)}{=} |C_{8t+8}(i)|_\alpha - |C_{8t+8}(i+1)|_\alpha\end{aligned}$$

The equality (1) holds by the explicit description obtained for $C_{8t+8}(i)$. The equality (2) holds by hypothesis of induction. The third equality by arithmetic and the last equality holds by the above reasoning about $|C_{8t+8}(i)|_\alpha$ and because for all $j \in \{1,\ldots,L\}$, $i \neq j$, $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$, in particular for $j = i+1$.
In order to finish the proof that C_{8t+8} is a representative configuration we need to see that $|C_{8t+8}(i)|_y = 0$ but this holds by the explicit description obtained for $C_{8t+8}(i)$.

Case 3: Let us suppose that

$$|C_{8t}(i+1)|_\alpha \geq |C_{8t}(i-1)|_\alpha > |C_{8t}(i)|_\alpha$$

In this case $|C_{8t}(i-1)|_x = |C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha > 0$, $|C_{8t}(i)|_x = |C_{8t}(i-1)|_y = 0$ and $|C_{8t}(i)|_y = |C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha > 0$.
Also, $max\{|C_{8t}(i-1)|_\alpha, |C_{8t}(i)|_\alpha + 1, |C_{8t}(i+1)|_\alpha\} = |C_{8t}(i+1)|_\alpha$. We will prove that C_{8t+8} is a representative configuration, $|C_{8t+8}(i)|_\alpha = |C_{8t}(i+1)|_\alpha$ and $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$ for $j \in \{1,\ldots,i-1,i+1,\ldots,L\}$
As y is present in cell i,

$C_{8t}(i) \xrightarrow{R^i_*} C_{8t+1}(i) = C_{8t}(i) \cup \{c_0\}^+ \xrightarrow{R^i_1, R^i_3} C_{8t+2}(i) = \{\alpha^{|C_{8t}(i)|_\alpha}\} \cup \{z^{|C_{8t}(i)|_y}\, \alpha^{|C_{8t}(i)|_y}\, c_1\}^+$

$C_{8t+k}(j) = C_{8t}(j)$, for all $j \in \{1,\ldots,i-1,i+1,\ldots,L\}$ as no rules affect to them and $C_{8t+k}(env) = \emptyset$ for $k \in \{1,2\}$.
As cell i has positive electrical charge in C_{8t+2}, then we apply the rules $[\,]_{i-1}, [z]^+_i \to [y]_{i-1}, [\,]_i$ and $[c_1 \to c_2]^+_i$ obtaining

$C_{8t+3}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\}$

$C_{8t+3}(i) = \{\alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\, c_2\}$

$C_{8t+3}(j) = C_{8t}(j)$, for all $j \in \{1,\ldots,i-2,i+1,\ldots,L\}$ and $C_{8t+3}(env) = \emptyset$.
In this situation $|C_{8t+3}(i-1)|_x = |C_{8t}(i-1)|_x > 0$ and $|C_{8t+3}(i-1)|_y = |C_{8t}(i)|_y > 0$ so the rule $[x\, y \to \lambda]_{i-1}$ can be applied. In order to compute the number of copies of x and y that remain in cell $i-1$ after the application of this rule we consider that

$$\begin{aligned}|C_{8t}(i-1)|_x &\overset{(1)}{=} |C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha \\ &\overset{(2)}{\leq} |C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha \\ &\overset{(3)}{=} |C_{8t}(i)|_y\end{aligned}$$

The equality (1) holds because C_{8t} is a representative configuration and in this case $|C_{8t}(i-1)|_\alpha > |C_{8t}(i)|_\alpha$. The inequality (2) holds because in this case $|C_{8t}(i+1)|_\alpha \geq |C_{8t}(i-1)|_\alpha$ and finally, the last equality holds by the definition of representative configuration and $|C_{8t}(i+1)|_\alpha > |C_{8t}(i)|_\alpha$.
The rule $[\,]_{i-1}, [c_2]_i^+ \to [c_3]_{i-1}^-, [\,]_i$ can also be applied. Therefore,

$C_{8t+4}(i-1) = \{y^{|C_{8t}(i)|_y - |C_{8t}(i-1)|_x}\, \alpha^{|C_{8t}(i-1)|_\alpha}\, c_3\}^-$

$C_{8t+4}(i) = \{\alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\}$

$C_{8t+4}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+4}(env) = \emptyset$.

As no x are present in cell $i-1$,

$C_{8t+4}(i-1) \xrightarrow{R_6^{i-1}} C_{8t+5}(i-1) = \{y^{|C_{8t}(i)|_y - |C_{8t}(i-1)|_x}\, \alpha^{|C_{8t}(i-1)|_\alpha}\, c_4\}^- \xrightarrow{R_7^{i-1}} C_{8t+6}(i-1) = \{y^{|C_{8t}(i)|_y - |C_{8t}(i-1)|_x}\, \alpha^{|C_{8t}(i-1)|_\alpha}\, c_5\}^- \xrightarrow{R_8^{i-1}} C_{8t+7}(i-1) = \{y^{|C_{8t}(i)|_y - |C_{8t}(i-1)|_x}\, \alpha^{|C_{8t}(i-1)|_\alpha}\, c_6\}^-$

$C_{8t+k}(i) = C_{8t+4}(i)$ and $C_{8t+k}(j) = C_{8t}(j)$ for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ as no rules affect to them and $C_{8t+k}(env) = \emptyset$ for $k \in \{5, 6, 7\}$.
Next, the rule $[c_6]_{i-1}^- \to p\,[\,]_{i-1}$ is applied,

$C_{8t+8}(i-1) = \{y^{|C_{8t}(i)|_y - |C_{8t}(i-1)|_x}\, \alpha^{|C_{8t}(i-1)|_\alpha}\}$

$C_{8t+8}(i) = C_{8t+4}(i)$ and $C_{8t+8}(j) = C_{8t}(j)$ for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ and $C_{8t+8}(env) = \{p\}$.
Note that $|C_{8t+8}(i)|_\alpha = |C_{8t}(i)|_y + |C_{8t}(i)|_\alpha$ and by hypothesis $|C_{8t}(i)|_y = |C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha$ so $|C_{8t+8}(i)|_\alpha = |C_{8t}(i+1)|_\alpha$, and that there is no x or y in $C_{8t+8}(i)$.
Finally as in this case $|C_{8t+8}(i)|_\alpha = |C_{8t}(i+1)|_\alpha \geq |C_{8t}(i-1)|_\alpha = |C_{8t+8}(i-1)|_\alpha$ then

$$|C_{8t+8}(i)|_\alpha \geq |C_{8t+8}(i-1)|_\alpha$$

and

$$\begin{aligned} |C_{8t+8}(i-1)|_y &\overset{(1)}{=} |C_{8t}(i)|_y - |C_{8t}(i-1)|_x = \\ &\overset{(2)}{=} (|C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha) - (|C_{8t}(i-1)|_\alpha - |C_{8t}(i)|_\alpha) = \\ &\overset{(3)}{=} |C_{8t}(i+1)|_\alpha - |C_{8t}(i-1)|_\alpha = \\ &\overset{(4)}{=} |C_{8t}(i+1)|_\alpha - |C_{8t+8}(i)|_\alpha \end{aligned}$$

The equality (1) holds by the explicit description obtained for $|C_{8t+8}(i-1)|_y$. The equality (2) holds by hypothesis of induction. The third equality by arithmetic and the last by the above reasoning about $|C_{8t+8}(i)|_\alpha$.
In order to finish the proof that C_{8t+8} is a representative configuration we need to see that $|C_{8t+8}(i-1)|_x = 0$ but this holds by the explicit description obtained for $C_{8t+8}(i-1)$.

Case 4: Let us suppose that

$$|C_{8t}(i)|_\alpha \geq |C_{8t}(i-1)|_\alpha \quad \text{and} \quad |C_{8t}(i)|_\alpha \geq |C_{8t}(i+1)|_\alpha$$

In this case $|C_{8t}(i)|_x = |C_{8t}(i)|_\alpha - |C_{8t}(i+1)|_\alpha \geq 0$, $|C_{8t}(i-1)|_x = |C_{8t}(i)|_y = 0$ and $|C_{8t}(i-1)|_y = |C_{8t}(i)|_\alpha - |C_{8t}(i-1)|_\alpha \geq 0$.

Also, $max\{|C_{8t}(i-1)|_\alpha, |C_{8t}(i)|_\alpha + 1, |C_{8t}(i+1)|_\alpha\} = |C_{8t}(i)|_\alpha + 1$. We will prove that C_{8t+8} is a representative configuration, $|C_{8t+8}(i)|_\alpha = |C_{8t}(i+1)|_\alpha + 1$ and $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$ for $j \in \{1, \ldots, i-1, i+1, \ldots, L\}$

$C_{8t}(i) \xrightarrow{R^i_*} C_{8t+1}(i) = C_{8t}(i) \cup \{c_0\}^+ \xrightarrow{R^i_1} C_{8t+2}(i) = C_{8t}(i) \cup \{c_1\}^+ \xrightarrow{R^i_2} C_{8t+3}(i) = C_{8t}(i) \cup \{c_2\}^+$

$C_{8t+k}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-1, i+1, \ldots, L\}$ as no rules affect to them and $C_{8t+k}(env) = \emptyset$ for $k \in \{1, 2, 3\}$.

As cell i has positive electrical charge in C_{8t+3}, then we apply the rule $[\,]_{i-1}, [c_2]^+_i \to [c_{n3}]^-_{i-1}, [\,]_i$ obtaining

$C_{8t+4}(i-1) = C_{8t}(i-1) \cup \{c_{n3}\}^-$

$C_{8t+4}(j) = C_{8t}(j)$ for all $j \in \{1, \ldots, i-2, i, \ldots, L\}$ and $C_{8t+4}(env) = \emptyset$

After that,

$C_{8t+4}(i-1) = C_{8t}(i-1) \cup \{c_{n3}\}^- \xrightarrow{R^{i-1}_{10}} C_{8t+5}(i-1) = C_{8t}(i-1) \cup \{c_{n4}\}^- \xrightarrow{R^{i-1}_{11}} C_{8t+6}(i-1) = C_{8t}(i-1) \cup \{x\, y\, c_5\}^-$

$C_{8t+k}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i, \ldots, L\}$ as no rules affect to them and $C_{8t+k}(env) = \emptyset$ for $k \in \{5, 6\}$.

At this step rules $[c_5 \to c_6]^-_{i-1}$ and $[x]^-_{i-1}, [\,] \to [\,]_{i-1}[z]_i$ are applied simultaneously and therefore the object x in cell $i-1$ is sent into cell i transformed into an object z. Hence

$C_{8t+7}(i-1) = C_{8t}(i-1) \cup \{y\, c_6\}$

$C_{8t+7}(i) = C_{8t}(i) \cup \{z\}$

$C_{8t+7}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ as no rules affect to them and $C_{8t+7}(env) = \emptyset$.

Now, rules $[c_6]_{i-1} \to p\,[\,]_{i-1}$ and $[z \to x\,\alpha]_i$ can be applied, then

$C_{8t+8}(i-1) = C_{8t}(i-1) \cup \{y\}$

$C_{8t+8}(i) = C_{8t}(i) \cup \{x\,\alpha\}$

$C_{8t+8}(j) = C_{8t}(j)$, for all $j \in \{1, \ldots, i-2, i+1, \ldots, L\}$ as no rules affect to them and $C_{8t+8}(env) = \{p\}$.

Note that $|C_{8t+8}(i)|_\alpha = |C_{8t}(i)|_\alpha + 1$ and that there is no x in $C_{8t+8}(i-1)$ and no $y$in $C_{8t+8}(i)$.

Finally, as in this case $|C_{8t+8}(i-1)|_\alpha = |C_{8t}(i-1)|_\alpha < |C_{8t}(i)|_\alpha + 1 = |C_{8t+8}(i)|_\alpha$ and $|C_{8t+8}(i+1)|_\alpha = |C_{8t}(i+1)|_\alpha < |C_{8t}(i)|_\alpha + 1 = |C_{8t+8}(i)|_\alpha$, then

$$|C_{8t+8}(i-1)|_\alpha < |C_{8t+8}(i)|_\alpha \quad \text{and} \quad |C_{8t+8}(i+1)|_\alpha < |C_{8t+8}(i)|_\alpha$$

so

$$\begin{aligned} |C_{8t+8}(i-1)|_y &\overset{(1)}{=} |C_{8t}(i-1)|_y + 1 \\ &\overset{(2)}{=} |C_{8t}(i)|_\alpha - |C_{8t}(i-1)|_\alpha + 1 \\ &\overset{(3)}{=} |C_{8t+8}(i)|_\alpha - |C_{8t+8}(i-1)|_\alpha \end{aligned}$$

and

$$\begin{aligned} |C_{8t+8}(i)|_x &\overset{(1)}{=} |C_{8t}(i)|_x + 1 \\ &\overset{(2)}{=} |C_{8t}(i)|_\alpha - |C_{8t}(i+1)|_\alpha + 1 \\ &\overset{(3)}{=} |C_{8t+8}(i)|_\alpha - |C_{8t+8}(i+1)|_\alpha \end{aligned}$$

The equality (1) holds by the explicit description obtained for $C_{8t+8}(i)$. The equality (2) holds by hypothesis of induction and the last equality holds by the above reasoning about $|C_{8t+8}(i)|_\alpha$ and because for all $j \in \{1,\ldots,L\}$, $i \neq j$, $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$, in particular for $j = i-1$ and $j = i+1$.

Case 5: Let us suppose that

$$|C_{8t}(i+1)|_\alpha > |C_{8t}(i)|_\alpha \geq |C_{8t}(i-1)|_\alpha$$

In this case $|C_{8t}(i-1)|_x = |C_{8t}(i)|_x = 0$, $|C_{8t}(i-1)|_y = |C_{8t}(i)|_\alpha - |C_{8t}(i-1)|_\alpha \geq 0$, $|C_{8t}(i)|_y = |C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha > 0$.
Also, $max\{|C_{8t}(i-1)|_\alpha, |C_{8t}(i)|_\alpha + 1, |C_{8t}(i+1)|_\alpha\} = |C_{8t}(i+1)|_\alpha$. We will prove that C_{8t+8} is a representative configuration, $|C_{8t+8}(i)|_\alpha = |C_{8t}(i+1)|_\alpha$ and $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$ for $j \in \{1,\ldots,i-1,i+1,\ldots,L\}$
As y is present in cell i,

$C_{8t}(i) \xrightarrow{R^i_*} C_{8t+1}(i) = C_{8t}(i) \cup \{c_0\}^+ \xrightarrow{R^i_1, R^i_3} C_{8t+2}(i) = \{\alpha^{|C_{8t}(i)|_\alpha}\} \cup \{z^{|C_{8t}(i)|_y}\, \alpha^{|C_{8t}(i)|_y}\, c_1\}^+$

$C_{8t+k}(j) = C_{8t}(j)$, for all $j \in \{1,\ldots,i-1,i+1,\ldots,L\}$ as no rules affect to them and $C_{8t+k}(env) = \emptyset$ for $k \in \{1,2\}$.
As cell i has positive electrical charge in C_{8t+2}, then we apply the rules $[\,]_{i-1}, [z]^+_i \to [y]_{i-1}, [\,]_i$ and $[c_1 \to c_2]^+_i$ obtaining

$C_{8t+3}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\}$

$C_{8t+3}(i) = \{\alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\, c_2\}$

$C_{8t+3}(j) = C_{8t}(j)$, for all $j \in \{1,\ldots,i-2,i+1,\ldots,L\}$ and $C_{8t+3}(env) = \emptyset$.

Now, the rule $[\,]_{i-1}, [c_2]^+_i \to [c_3]^-_{i-1}, [\,]_i$ can be applied. Therefore,

$C_{8t+4}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\, c_3\}^-$

$C_{8t+4}(i) = \{\alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\}$

$C_{8t+4}(j) = C_{8t}(j)$, for all $j \in \{1,\ldots,i-2,i+1,\ldots,L\}$ and $C_{8t+4}(env) = \emptyset$.

After that,

$C_{8t+4}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\, c_3\}^- \xrightarrow{R^{i-1}_6} C_{8t+5}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\, c_4\}^- \xrightarrow{R^{i-1}_7} C_{8t+6}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\, c_5\}^- \xrightarrow{R^{i-1}_8} C_{8t+7}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\, c_6\}^-$

$C_{8t+k}(i) = \{\alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\}$ and $C_{8t+k}(j) = C_{8t}(j)$, for all $j \in \{1,\ldots,i-2,i+1,\ldots,L\}$ and $C_{8t+k}(env) = \emptyset$ for $k \in \{5,6,7\}$.

Next, the rule $[c_6]_{i-1} \to p\,[\,]_{i-1}$ is applied,

$C_{8t+8}(i-1) = C_{8t}(i-1) \cup \{y^{|C_{8t}(i)|_y}\}^-$

$C_{8t+8}(i) = \{\alpha^{|C_{8t}(i)|_\alpha + |C_{8t}(i)|_y}\}$ and $C_{8t+8}(j) = C_{8t}(j)$, for all $j \in \{1,\ldots,i-2,i+1,\ldots,L\}$ and $C_{8t+8}(env) = \{p\}$.

Note that $|C_{8t+8}(i)|_\alpha = |C_{8t}(i)|_\alpha + |C_{8t}(i)|_y$ and by hypothesis $|C_{8t}(i)|_y = |C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha$ so $|C_{8t+8}(i)|_\alpha = |C_{8t}(i+1)|_\alpha = |C_{8t+8}(i+1)|_\alpha$, and that there is no x or y in $C_{8t+8}(i)$.
Finally, as $|C_{8t+8}(i-1)|_\alpha = |C_{8t}(i-1)|_\alpha < |C_{8t}(i)|_\alpha + |C_{8t}(i)|_y = |C_{8t+8}(i)|_\alpha$, then

$$|C_{8t+8}(i-1)|_\alpha < |C_{8t}(i)|_\alpha$$

and

$$\begin{aligned}
|C_{8t+8}(i-1)|_y &\overset{(1)}{=} |C_{8t}(i-1)|_y + |C_{8t}(i)|_y \\
&\overset{(2)}{=} (|C_{8t}(i)|_\alpha - |C_{8t}(i-1)|_\alpha) + (|C_{8t}(i+1)|_\alpha - |C_{8t}(i)|_\alpha) \\
&\overset{(3)}{=} |C_{8t}(i+1)|_\alpha - |C_{8t}(i-1)|_\alpha \\
&\overset{(4)}{=} |C_{8t+8}(i)|_\alpha - |C_{8t+8}(i-1)|_\alpha
\end{aligned}$$

The equality (1) holds by the explicit description obtained for $C_{8t+8}(i-1)$. The equality (2) holds by hypothesis of induction. The third equality by arithmetic and the last equality holds by the above reasoning about $|C_{8t+8}(i)|_\alpha$ and because for all $j \in \{1, \ldots, L\}$, $i \neq j$, $|C_{8t+8}(j)|_\alpha = |C_{8t}(j)|_\alpha$, in particular for $j = i-1$. In order to finish the proof that C_{8t+8} is a representative configuration we need to see that $|C_{8t+8}(i-1)|_x = 0$ but this holds by the explicit description obtained for $C_{8t+8}(i-1)$.

P Systems with Adjoining Controlled Communication Rules

Mihai Ionescu[1], Dragoş Sburlan[2]

[1] Rovira i Virgili University
Research Group on Mathematical Linguistics
Tarragona, Spain
armandmihai.ionescu@urv.net
[2] Ovidius University
Faculty of Mathematics and Informatics
Constantza, Romania
dsburlan@univ-ovidius.ro

Summary. This paper proposes a new model of P systems where the rules are activated by objects present in the neighboring regions. We obtain the computational completeness considering only two membranes, external inhibitors and carriers. Leaving the carriers apart we obtain equality with ET0L systems in terms of number sets.

1 Introduction

Having as inspiration the way living cells are divided by membranes into compartments where various biochemical processes take place, *P systems* (also known as *membrane systems*) area grew rapidly since Gheorghe Păun, proposed the first model in 1998 ([4]). A complete bibliography of P systems can be found on the P system webpage ([8]).

Within the living cell there are several energy consuming activities. Among them there is the transport activity which is of three types: *diffusion*, *facilitated diffusion*, and *active transport*. Simple diffusion means that the molecules can pass directly through the membrane, always down a concentration gradient, while in the case of facilitated diffusion and active transport molecules can pass both down an up the concentration gradient. In the facilitated diffusion membrane protein channels are used to allow charged molecules (which otherwise could not diffuse across the cell membrane) to freely diffuse the cell, while active transport requires the expenditure of energy to transport the molecule from one side of the membrane to the other.

Hence, living cells get/expel from/to their environment many substances and for this aim they have developed specific transport systems across membranes, even against a concentration gradient. Often enough this necessity of the living

cell to expel or attract various molecules is triggered by the presence or the absence of certain chemicals in the immediate neighboring (inner or outer) regions.

Here we deal with P systems where the rules from a given region are activated precisely by the presence or the absence of certain symbols in the neighboring regions. This model has a biological counterpart and it is inspired by the chemicals that pass through the membranes of the cell, from one region to another, in the sense of polarization gradient. In this case, the electrical charge plays the role of the promoter.

Before going into the definition of the new model and its computational power (Section 3) let us briefly remind the reader some basic notions and notations (Section 2). Section 4 is dedicated to the conclusions and challenges for further research.

2 Preliminaries and Definitions

We assume familiarity with the basics of formal language theory (see [6]), as well as with the basics of membrane computing (see [5]).

An *alphabet* is a finite set of symbols (letters), and a word (string) over an alphabet Σ is a finite sequence of letters from Σ. We denote the empty word by λ, the length of a word w by $|w|$, and the number of occurrences of a symbol a in w by $|w|_a$. The (con)catenation of two words x and y is denoted by xy.

A *language* over Σ is a (possibly infinite) set of words over Σ. The language consisting of all words over Σ is denoted by Σ^*, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. We denote by `REG`, `CF`, `ET0L`, `CS`, `RE` the families of languages generated by regular, context-free, table Lindemayer interactionless systems context-sensitive, and of arbitrary grammars, respectively (`RE` stands for recursively enumerable languages). The following strict inclusions hold: `REG` $\subset$ `CF` $\subset$ `ET0L` $\subset$ `CS` $\subset$ `RE`.

For a family `FL` of languages, `NFL` denotes the family of length sets of languages in `FL`. The following relations hold: `NREG` = `NCF` $\subset$ `NET0L` $\subset$ `NCS` $\subset$ `NRE`.

The multisets over a given finite support (alphabet) are represented by strings of symbols. The order of symbols does not matter, because the number of copies of an object in a multiset is given by the number of occurrences of the corresponding symbol in the string (see [1] for other ways to specify multisets).

3 The Model

Based on the biological observations mentioned in the introductory section we introduce the following new class of P systems.

3.1 Defining the Model

Definition 1. *A P system with adjoining controlled communication rules (called in short, a PACC system) is a construct*

$$\Pi = (V, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_0),$$

where:

- V *is the alphabet of objects;*
- $C \subseteq V$ *is the set of carriers;*
- μ *is a membrane structure with* m *membranes (labeled in a one-to-one manner by* $1, \ldots, m$*);*
- $w_1, \ldots, w_m$ *are the multisets of objects initially present in the regions of* Π*;*
- $R_1, \ldots, R_m$ *are finite sets of communication rules associated to membranes, that are of the following types:*
  - ◇ simple rules:
    $[A]_i \longrightarrow [\ ]_i\alpha$ *or* $A[\ ]_i \longrightarrow [\alpha]_i$*, for* $A \in V \setminus C$, $\alpha \in (V \setminus C)^*$,
  - ◇ promoted simple rules:
    $[A]_i B \longrightarrow [\ ]_i\alpha$ *or* $A[B]_i \longrightarrow [\alpha]_i$*, for* $A, B \in V \setminus C$, $\alpha \in (V \setminus C)^*$,
  - ◇ inhibited simple rules:
    $[A]_i \neg B \longrightarrow [\ ]_i\alpha$ *or* $A[\neg B]_i \longrightarrow [\alpha]_i$*, for* $A, B \in V \setminus C$, $\alpha \in (V \setminus C)^*$,
  - ◇ carrier rules:
    pairs of rules $[cA]_i \longrightarrow [\ \ ]_i c\alpha$ *and* $c[\ ]_i \longrightarrow [c]_i$*, for* $A \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$, *or*
    pairs of rules $cA[\ ]_i \longrightarrow [c\alpha]_i$ *and* $[c]_i \longrightarrow [\ \ ]_i c$ *for* $A \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$,
  - ◇ promoted carrier rules:
    pairs of rules $[cA]_i B \longrightarrow [\ ]_i c\alpha$ *and* $c[\ ]_i \longrightarrow [c]_i$*, for* $A, B \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$, *or*
    pairs of rules $cA[B]_i \longrightarrow [c\alpha]_i$ *and* $[c]_i \longrightarrow [\ ]_i c$*, for* $A, B \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$*;*
  - ◇ inhibited carrier rules:
    pairs of rules $[cA]_i \neg B \longrightarrow [\ ]_i c\alpha$ *and* $c[\ ]_i \longrightarrow [c]_i$*, for* $A, B \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$, *or*
    pairs of rules $cA[\neg B]_i \longrightarrow [c\alpha]_i$ *and* $[c]_i \longrightarrow [\ ]_i c$*, for* $A, B \in V \setminus C$, $c \in C$, $\alpha \in (V \setminus C)^*$*;*
- $i_0 \in \{1, \ldots, m\}$ *is an elementary membrane of* μ *(the output membrane).*

In a *simple rule* an object is rewritten in a string of objects, in the inner or outer region with respect to the initial object. A *promoted simple rule/inhibited simple rule* has the same action as a simple rule but it can be applied only in the presence/absence of certain objects (chemicals) called promoters/inhibitors. To be more precise we take as example the rule $[A]_i B \longrightarrow [\ ]_i\alpha$, which implies that object A is rewritten in α in the outer membrane only if *promoter* B is present there. If we replace B with $\neg B$, the object plays the role of the *inhibitor*, and only by its presence it blocks the execution of the rule.

In a *carrier rule* the objects can be rewritten only if they are guided by an object, the *carrier*. Note that the carrier is not actively participating in the reaction. Its role is to "accompany" the reaction and to inhibit the parallelism. As an

example, by rule $\left[cA\right]_i \longrightarrow \left[\ \right]_i c\alpha$ we mean that object A evolves to α (in the outer region of object A) iff there is an object c that helps A to be rewritten.

Promoted/Inhibited carrier rules can be applied if besides the carrier there is also a promoter/inhibitor which triggers/blocks the reaction.

As usual in membrane computing, the rules are used in a nondeterministic maximally parallel manner starting from an initial configuration. In this way, we obtain transitions between the configurations of the system. A configuration is described by the m-tuple of the multisets of objects present in the m regions of the system. The initial configuration is $(w_1, \ldots, w_m)$.

A sequence of transitions between configurations of the system constitutes a *computation*; a computation is successful if it *halts*, i.e., it reaches a configuration (the halting configuration) where no rule can be applied to any of the objects.

The *result* of a successful computation is the number of objects present within the membrane with the label i_o in the halting configuration. A computation which never halts yields no result.

We use the notation $NPACC_m(\alpha, \beta)$, where $\alpha \in \{smp\} \cup \{cat_k \mid k \geq 0\}$, $\beta \in \{proR_i, inhR_i\}$ to denote the family of sets of natural numbers generated by P systems with adjoining controlled communication rules having at most m membranes, communication rules that can be simple $\alpha = smp$, or carrier $\alpha = cat_k$, using at most k carriers, and external promoters $\beta = proR_i$ or external inhibitors $\beta = inhR_i$ of weight i at the level of rules.

3.2 An Example

Let us now exemplify the functioning of the model defined above throughout an **example**. Here it shown how such machines can be used to compute functions.

Consider the following system:

$$\Pi_1 = (\{A, B, D\}, C = \{c\}, [\ [\ ]_2]_1, w_1 = \{A^n\}, w_2 = \{c\}, R_1, R_2, 2),$$

where:

- R_1=$\emptyset$, R_2=$\{A\left[\ \right]_2 \longrightarrow \left[ABD\right]_2$, $\left[B\right]_2 \longrightarrow \left[\ \right]_2 B$, $\left[cD\right]_2 \longrightarrow \left[\ \right]_2 c$, $B\left[D\right]_2 \longrightarrow \left[AB\right]_2$, $c\left[\ \right]_2 \longrightarrow \left[c\right]_2\}$.

The system Π is fed with $n \geq 1$ copies of object A in region 1 and when it halts, the contents of the output region contains n^2 copies of A.

The functioning of the system is rather simple. The only rule we can apply in the initial configuration is the one which rewrites object A in ABD in the inner region, hence in the second step of the computation we will have all the objects of the system (n copies of A, n copies of B, n copies of D and the object initially present here, carrier c) in region 2. Then, we expel all objects B in region 1 and we start consuming objects D by applying the rule $\left[cD\right]_2 \longrightarrow \left[\ \right]_2 c$, hence object D is sent outside membrane 2 and is rewritten to λ having carrier c accompanying the reaction.

Note that object D plays the role of the counter and each time a copy of D is deleted (for example in step i of the computation), n more copies of A are produced (in step $i+2$ of the computation). One by one the n-th copies of D are consumed, adding for each of them n copies to object A. In the rule $B\big[D\big]_2 \longrightarrow \big[AB\big]_2$, object D plays also the role of promoter and object B can be rewritten into AB only in its presence. The computation ends with n^2 copies of A in region 2, hence the system computes the number-theoretic function $f(n) = n^2$, $n \geq 1$.

3.3 The Results

In what follows we will prove that the class of sets of numbers generated by P systems with external inhibitors equals the class of sets of numbers generated by P systems with external inhibitors and only two membranes.

Lemma 1. $NPACC_m(smp, inhR_1) = NPACC_2(smp, inhR_1), m \geq 2$.

Proof. Obviously, $NPACC_m(inh) \supseteq NPACC_2(inh)$. For the opposite inclusion we have to show that for any P system with external inhibitors $\overline{\Pi} = (\overline{V}, \overline{C}, \overline{\mu}, \overline{R}, \overline{i_0})$ generating a set of natural numbers, there exists an equivalent P system with external inhibitors $\Pi = (V, C, \mu, R, i_0)$ with only 2 membranes.

To this aim, we simulate the computation of $\overline{\Pi}$, with the system Π defined as follows.

Let us denote by $\mathcal{L} = \{1, 2, \ldots, m\}$ the set of labels of the regions in $\overline{\Pi_m}$. In addition, assume that $\overline{R} = \{R_1, \ldots, R_m\}$, and each $\overline{R_i} \in \overline{R}$, $1 \leq i \leq m$, contains all the rules that cross membrane i. Then, we define:

- $V = \{a_i \mid a \in \overline{V}, i \in \mathcal{L}\}$;
- $C = \overline{C} = \emptyset$;

Let $h : \overline{V}^* \times \mathcal{L} \to V^*$ be a mapping such that

1) $h(a, i) = a_i, a \in \overline{V}, i \in \mathcal{L}$,
2) $h(\lambda, j) = \lambda$, $j \in \mathcal{L}$,
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 $w = h(\overline{w_1})h(\overline{w_2}) \ldots h(\overline{w_m})$, 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 $A\big[\ \big]_i \longrightarrow \big[\alpha\big]_i \in R_i$, $A \in \overline{V}$, $\alpha \in \overline{V}^*$, $i \in \mathcal{L}$, we add to R the rule $h(A, j)\big[\ \big]_1 \longrightarrow \big[h(\alpha', i)\big]_1$, providing that j is the label of the outer membrane of membrane i.

  For each rule $A\big[\neg B\big]_i \longrightarrow \big[\alpha\big]_i \in R_i$, $A, B \in \overline{V}$, $\alpha \in \overline{V}^*$, $i \in \mathcal{L}$, we add to R the rule $h(A, j)\big[\neg h(B, i)\big]_1 \longrightarrow \big[h(\alpha', 2)\big]_1$, providing that j is the label of the outer membrane of membrane i.

  For each rule $\big[A\big]_i \longrightarrow \big[\ \ \big]_i\alpha \in R_i$, $A, B \in \overline{V}$, $\alpha \in \overline{V}^*$, $i \in \mathcal{L}$, we add to R the rule $\big[h(A, i)\big]_1 \longrightarrow \big[\ \ \big]_1 h(\alpha', j)$ providing that j is the outer membrane of membrane i.

For each rule $\left[A\right]_i \neg B \longrightarrow \left[\ \right]_i \alpha \in R_i$, $A, B \in \overline{V}$, $\alpha \in \overline{V}^*$, $i \in \mathcal{L}$, we add to R the rule $\left[h(A,i)\right]_1 \neg h(B,j) \longrightarrow \left[\ \right]_1 h(\alpha', j)$ providing that j is the outer membrane of membrane i.

Generally speaking, the purpose of membranes is to keep private the interior rules and objects from the neighboring ones and vice-versa. However, in our case we can express the passage of certain symbol through the membranes by using new symbols that we add to vocabulary and that encode both the crossed membrane label and the symbols from where they derive. In this way we can rewrite the rules, using the new symbols that perfectly describe the passage of objects in the membrane structure; consequently, in our case, we can shrink an arbitrarily membrane structure to only two membranes. The morphism used by the above construction accomplishes the encoding procedure.

The system Π simulates all the moves of $\overline{\Pi}$ and it stops whenever $\overline{\Pi}$ stops. However, in the halting configuration, in the designated output region of Π, there could be some objects representing the encoded version of the objects present in the regions of $\overline{\Pi}$. Therefore, we have to modify the above set of rules such that Π eliminates all these objects in order to generate the same set of numbers as $\overline{\Pi}$. This can be accomplish by producing an object D whenever a rule of $\overline{\Pi}$ is simulated (by adding the object D at the right hand side of each above rule), deleting it at each step (we add to R rules of type $D\left[\ \right]_1 \longrightarrow \left[\lambda\right]_1$ and $\left[D\right]_1 \longrightarrow \left[\ \right]_1 \lambda$). Finally, if $\overline{\Pi}$ stops, then Π will not produce the object D anymore, hence the absence of this object can trigger an inhibited rule that deletes all the unnecessary objects. Consequently, we have that $NPACC_m(smp, inhR_1) = NPACC_2(smp, inhR_1), m \geq 2$.

Here we will prove that the family of sets of vectors of numbers generated by P systems with external inhibitors equals the family of sets of numbers generated by ET0L systems.

Theorem 1. $NPACC_2(smp, inhR_1) = NET0L$.

Proof. We will prove the result by showing that communicative P systems with external inhibitors are equivalent with P systems with inhibitors, which at their turn, generates the same class of sets of numbers as the Parikh image of $ET0L$ as shown in [7]. Let $NP_1(smp, inhR_1)$ be the family of sets of numbers generated by P systems with inhibitors.

The proof of the inclusion $NP_1(smp, inhR_1) \supseteq NPACC_2(smp, inhR_1)$ is rather simple and is based on a similar encoding of regions into new objects as was presented above.

For the inclusion $NP_1(smp, inhR_1) \subseteq NPACC_2(smp, inhR_1)$ we will simulate the computation of a P system with one region $\Pi_{inh} = (V, C, \mu, w, R, i_0)$. We assume that the set of rules R contains rules of type $A \rightarrow \alpha$ or $A \rightarrow \alpha|_{\neg B}$, $A, B \in V$, $\alpha \in V^*$.

Let us consider the sets $\widetilde{V} = \{\widetilde{A} \mid A \in V\}$ and $\dot{V} = \{\dot{A} \mid A \in V\}$. In addition, let us define the morphisms:

$h_1 : V^* \to \widetilde{V}^*$, such that $h_1(A) = \widetilde{A}$ for all $A \in V$;
$h_2 : V^* \to \dot{V}^*$, such that $h_3(A) = \dot{A}$ for all $A \in V$.

We construct a P system $\Pi_{cc} = (\overline{V}, \overline{C}, \overline{\mu}, \overline{R}, \overline{i_0})$, simulating Π_{inh}, defined as follows:

$$\begin{array}{ll} \overline{V} = V \cup \widetilde{V} \cup \dot{V} \cup \{F\}; & w_1 = w; \\ C = \emptyset; & w_2 = w; \\ \mu = \left[\ \left[\ \right]_2 \right]_1; & i_0 = 1. \end{array}$$

The set of rules R is defined as follows[3]:

$$\begin{array}{rl} \text{step } i & A\big[\neg B\big] \longrightarrow \big[\mathbf{h_1}(\alpha)\mathbf{h_2}(\alpha)\big], \text{ for all rules } A \to \alpha|_{\neg B} \in R_{inh}, \\ \text{step } i & A\big[\ \big] \longrightarrow \big[\mathbf{h_1}(\alpha)\mathbf{h_2}(\alpha)\big], \text{ for all rules } A \to \alpha \in R_{inh}, \\ \text{step } i & \big[A\big] \longrightarrow \big[\ \big]F, \text{ if exists } A \to \alpha \in R_{inh}, \\ \text{step } i & \big[A\big]\neg B \longrightarrow \big[\ \big]F, \text{ if exists } A \to \alpha|_{\neg B} \in R_{inh}, \\ \text{step } i+1 & F\big[\ \big] \longrightarrow \big[\ \big], \\ \text{step } i+1 & \big[\mathbf{h_1}(A)\big] \longrightarrow \big[\ \big]\mathbf{h_1}(A), \text{ for all objects } A \in V, \\ \text{step } i+2 & \mathbf{h_1}(A)\big[\ \big] \longrightarrow \big[A\big], \text{ for all objects } A \in V, \\ \text{step } i+2 & \big[\mathbf{h_2}(A)\big]\neg R \longrightarrow \big[\ \big]A, \text{ for all } A \in V. \end{array}$$

Here is how the system Π_{cc} simulates the computation of Π_{inh}. First, remark that in order to correctly simulate the moves of Π_{inh}, we will maintain during the computation in both regions of Π_{cc} a copy of the multiset w – the multiset that represent the current configuration of Π_{inh}. This is especially useful when trying to simulate rules of type $A \to \alpha|_{\neg B} \in R_{inh}$ because we have to know whether or not the external inhibitor is present.

We assume that the system is in a configuration given by the strings $w_1 = w_2 = w$. The system attempts to execute simultaneously the rules of type

$$\begin{array}{rl} \text{step } i & A\big[\neg B\big] \longrightarrow \big[\mathbf{h_1}(\alpha)\mathbf{h_2}(\alpha)\big], \text{ for all rules } A \to \alpha|_{\neg B} \in R_{inh}, \\ \text{step } i & A\big[\ \big] \longrightarrow \big[\mathbf{h_1}(\alpha)\mathbf{h_2}(\alpha)\big], \text{ for all rules } A \to \alpha \in R_{inh}, \\ \text{step } i & \big[A\big] \longrightarrow \big[\ \big]F, \text{ if exists } A \to \alpha \in R_{inh}, \\ \text{step } i & \big[A\big]\neg B \longrightarrow \big[\ \big]F, \text{ if exists } A \to \alpha|_{\neg B} \in R_{inh}. \end{array}$$

Remark that the rules of first two types are used to generate inside the inner region, two copies of multiset α (represented by $\mathbf{h_1}(\alpha)$ and $\mathbf{h_2}(\alpha)$). In the same

[3] For the present proof, we will simplify the notation by not including the membrane labels into the syntax of the rules; this is possible here since we have only two membranes and we do not allow the interaction with the environment. In addition, we have specified on their left hand side the moment of their executions during the simulation of one computational step in Π_{inh}.

time, the rules of second type delete from region 2 the objects that were within the scope of rules of first type. In addition, remark that there are no other rules that can be applied in this step. Moreover, they produce in region 1 objects R; these objects will be used later for synchronizing the moments when multiset α appears in both regions.

Next, are executed the rules of type:

$$\begin{aligned}&\text{step } i+1 \quad F[\ ] \longrightarrow [\ ],\\ &\text{step } i+1 \quad [\mathbf{h_1}(A)] \longrightarrow [\ ]\mathbf{h_1}(A), \text{ for all objects } A \in V.\end{aligned}$$

Observe that the presence of object(s) R in this computational step inhibits the executions of rules of type $[\mathbf{h_2}(A)]\neg F \longrightarrow [\ ]A$, for all $A \in V$. Hence, in the third step, the rules of type

$$\begin{aligned}&\mathbf{h_1}(A)[\ ] \longrightarrow [A], \text{ for all objects } A \in V,\\ &[\mathbf{h_2}(A)]\neg F \longrightarrow [\ ]A, \text{ for all } A \in V,\end{aligned}$$

will be executed. The new objects appear at the same time in both regions of the system Π_{cc} and the simulation of the next computational step of Π_{inh} can start. Finally, if the system Π_{inh} stops because there are no rules to be applied, then also Π_{cc} halts.

Before we conclude, remark that the maximal parallelism as well as the universal clock is fundamental for the construction.

Consequently we have proved that the computation of an arbitrary P system with inhibitors can be simulated by a P system with external inhibitors, hence we have $NP_1(smp, inhR_1) \subseteq NPACC_2(smp, inhR_1)$. Therefore we have that $NP_1(smp, inhR_1) = NPACC_2(smp, inhR_1) = PsET0L$.

The following theorem shows that P systems with external inhibitors and carriers are computationally complete.

Theorem 2. $NPACC_2(cat, inhR_1) = NRE$.

Proof. The inclusion $NPACC_2(cat, inhR_1) \subseteq NRE$ is assumed true by invoking the Turing-Church thesis.

For the inclusion $NPACC_2(cat, inhR_1) \supseteq NRE$ we will simulate the computation of an arbitrary non-deterministic register machine $M = (n, \mathcal{P}, l_0, l_h)$. Such register machines are computational universal if $n \geq 3$.

We construct $\Pi = (V, C, \mu, w_1, w_2, R_1, i_0)$ as follows.

$$\begin{aligned}V &= \{a_i, A_i, S_i \mid 1 \leq i \leq n\} \cup \{l, \overline{l}, \overline{\overline{l}}, \widetilde{l}, \widetilde{\widetilde{l}}, L \mid l \in Lab(\mathcal{P})\} \cup \{c\}\\ &\quad \cup \{K, \overline{K}, \overline{\overline{K}}, \overline{\overline{\overline{K}}}, T_0, T_1, X, \overline{X}\};\\ C &= \{c\};\\ \mu &= [\ [\ ]_2\ ]_1;\end{aligned}$$

$$w_1 = l_0 L_0 a_1^{k_1} \dots a_n^{k_n} c;$$
$$w_2 = A_1 \dots A_n S_1 \dots S_n;$$
$$i_0 = 1.$$

The set of rules R is defined as follows:

• for each instruction $(l_1 : \mathrm{ADD}(j), l_2, l_3) \in \mathcal{P}$, the set R contains the rules:

$l_1[\ ] \longrightarrow [A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \dots S_n a_j l_2]$, $l_1 \neq l_h$,

$l_1[\ ] \longrightarrow [A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \dots S_n a_j l_3]$, $l_1 \neq l_h$,

$L_1[\neg A_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n]$,

$[l_2] \longrightarrow [\ ] l_2$,

$[l_3] \longrightarrow [\ ] l_3$,

$[a_j] \longrightarrow [\ ] a_j$,

$[A_i] \longrightarrow [\ ] \lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ] \lambda$, $1 \leq i \leq n$;

• for each instruction $(l_1 : \mathrm{SUB}(r), l_2, l_3) \in \mathcal{P}$, the set R contains the rules:

$l_1[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_{j-1} S_{j+1} \dots S_n \overline{l_1}]$, $l_1 \neq l_h$,

$ca_j[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n X]$,

$L_1[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n K]$,

$[\overline{l_1}] \longrightarrow [\ ] \overline{\overline{l_1}}$,

$[X] \longrightarrow [\ ] \overline{X}$,

$\overline{\overline{l_1}}[\ ] \longrightarrow [\overline{\overline{\overline{l_1}}} T_0 A_1 \dots A_n S_1 \dots S_n]$,

$[K] \longrightarrow [\ ] \overline{K}$,

$[\overline{\overline{\overline{l_1}}}] \neg X \longrightarrow [\ ] \widetilde{l_3}$,

$\overline{X}[\neg T_0] \longrightarrow [l_2]$,

$\overline{K}[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_n \overline{\overline{K}}]$,

$[T_0] \longrightarrow [\ ] T_1$,

$[\overline{\overline{\overline{l_1}}}] \neg \overline{K} \longrightarrow [\ ] \lambda$,

$[l_2] \longrightarrow [\ ] l_2 L_2$,

$T_1[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_n]$,

$\widetilde{l_3}[\ ] \longrightarrow [\widetilde{\widetilde{l_3}}]$,

$[\widetilde{\widetilde{l_3}}] \longrightarrow [\ ] l_3 L_3$,

$[\overline{\overline{K}}] \longrightarrow [\ ] \overline{\overline{\overline{K}}}$,

$\overline{\overline{\overline{K}}}[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_n]$,

$[A_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$.

Here is how the P system Π simulates the computation of the register machine M. Observe for the beginning that in the P system Π we will represent the number stored into register j of M as the multiplicity of the object a_j. In addition, remark that objects A_j, S_j, $1 \leq j \leq n$, stand for the addition/subtraction command over register j – both in the simulation of an ADD or SUB instruction, the absence of symbol A_j or S_j allows the addition or deletion of one occurrence of object a_j. Objects A_j, S_j, $1 \leq j \leq n$, are produced all the time during the computation except the moment when we actually want to increment or subtract one occurrence of object a_j from the multiset; at that moment we generate all objects A_i, S_i, $1 \leq i \leq n$, such that $i \neq j$.

Let us see in more details how the simulation of the addition instruction $(l_1 : \mathrm{ADD}(j), l_2) \in \mathcal{P}$ works. Assume that at a certain moment during the computation, the current multisets in regions 1 and 2 are represented by the strings $w_1 = l_1 L_1 a_1^{k_1} \dots a_n^{k_n} c$ and $w_2 = A_1 \dots A_n S_1 \dots S_n$ respectively. Then, the rules that can be executed are:

$l_1[\ ] \longrightarrow [A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \dots S_n a_j l_2]$ or the rule involving l_3,

$[A_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$.

As a consequence of executing the above rules the next configuration will be represented by $w_1 = L_1 a_1^{k_1} \dots a_n^{k_n} c$ and $w_2 = A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \dots S_n a_j l_2$. Now, since in region 2 the object A_j is missing, then the rule

$L_1[\neg A_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n]$

can be executed; its role is to reestablish the initial configuration in region 2. Simultaneously, the system runs the rules

$[l_2] \longrightarrow [\ ]l_2$,

$[a_j] \longrightarrow [\ ]a_j$,

$[A_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$.

The rule $[l_2] \longrightarrow [\ ]l_2$ produces in region 1 the object l_2 that corresponds to register machine label l_2. In addition, by the execution of the rule $[a_j] \longrightarrow [\ ]a_j$, the number of objects a_j in region 1 (that corresponds to the number stored in register j of M) is incremented.

Concerning the simulation of the subtract instruction $(l_1 : \mathrm{SUB}(j), l_2, l_3) \in \mathcal{P}$, the system Π, being in a configuration represented by $w_1 = l_1 L_1 a_1^{k_1} \dots a_n^{k_n} c$ and $w_2 = A_1 \dots A_n S_1 \dots S_n$, executes first the rules:

$l_1[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_{j-1} S_{j+1} \dots S_n \overline{l_1}]$,

$[A_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$,

$[S_i] \longrightarrow [\ ]\lambda$, $1 \leq i \leq n$.

In a similar manner as presented in the addition simulation, the rule $l_1[\ ] \longrightarrow [A_1 \dots A_n S_1 \dots S_{j-1} S_{j+1} \dots S_n \overline{l_1}]$ creates the context required for starting the simulation. The absence of object S_j in region 2 allows, in the second step, the (possible) execution of the rules

$ca_j[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n X]$,

$L_1[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n K]$.

Observe that in case there exists an object a_j in region 1, both rules are executed, while if there is not, only the rule $L_1[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n K]$ will be executed.

In the same step, the rule $[\overline{l_1}] \longrightarrow [\ ]\overline{\overline{l_1}}$ performs. As we will see, the objects derived from object l_1 will be used later to check whether or not the rule $ca_j[\neg S_j] \longrightarrow [A_1 \dots A_n S_1 \dots S_n X]$ was executed. Moreover, they will be also used to introduce in region 2 objects $A_1 \dots A_n S_1 \dots S_n$ that forbids a new addition or subtraction of objects a_j.

Let us consider the first case, i.e. the region 1 contains at least one object a_j. Then, as a consequence of executing the above rules we will have the multisets $w_1 = a_1^{k_1} \dots a_j^{k_j - 1} \dots a_n^{k_n} c$ and $w_2 = A_1^2 \dots A_n^2 S_1^2 \dots S_n^2 X K$. The following rules will be further applied:

$\overline{\overline{l_1}}[\ ] \longrightarrow [\overline{\overline{l_1}} T_0 A_1 \dots A_n S_1 \dots S_n]$,

$[K] \longrightarrow [\ ]\overline{K}$,

and possibly the rule:

$[X] \longrightarrow [\ ]\overline{X}$.

Remark that the objects derived from $\overline{l_1}$ are within the scope of rules that introduce at each odd step objects $A_1 \cdots A_n S_1 \dots S_n$ (or $A_1 \dots A_{j-1} A_{j+1} \dots A_n S_1 \cdots S_n$ in the first step). In a similar manner the objects derived from K are within the scope of rules that introduce at each even step objects $A_1 \cdots A_n S_1 \dots S_n$. Anyway, at each step we delete by rules $A_i \to \lambda$ and $S_i \to \lambda$, $1 \leq i \leq n$ all objects A_i and S_i.

Now, since in the third step an object $\overline{X}$ was introduced in region 1 then, in the fourth step, the rule $[\overline{\overline{l_1}}]\neg X \longrightarrow [\ ]\widetilde{l_3}$ cannot be executed. Moreover, because in region 2 exists an object T_0 also the rule $\overline{X}[\neg T_0] \longrightarrow [l_2]$ cannot be executed. However, in the fourth step the rule $[T_0] \longrightarrow [\ ]T_1$ runs and it will allow, in the fifth step, the execution of the rule $\overline{X}[\neg T_0] \longrightarrow [l_2]$. In the same time, rule $[\overline{\overline{l_1}}]\neg \overline{K} \longrightarrow [\ ]\lambda$ is executed and so there will be no way to rewrite $\overline{\overline{l_1}}$ into $\widetilde{l_3}$ and furthermore into l_3. Finally, by rule $[l_2] \longrightarrow [\ ]l_2 L_2$ the label of the new register machine instruction to be simulated is generated.

Now let us see what how the simulation is done when the system Π attempts to simulate the instruction $(l_1 : \mathrm{SUB}(j), l_2, l_3) \in \mathcal{P}$ in the case when the register j is empty. Then, the simulation works in a similar manner as in the above presented

case with the main difference being that in the fourth step the rule $\left[\overline{\overline{l_1}}\right]\neg X \longrightarrow \left[\ \right]\widetilde{l_3}$ is executed because the object $\overline{X}$ was not produced (the rules $ca_j\left[\neg S_j\right] \longrightarrow \left[A_1 \dots A_n S_1 \dots S_n X\right]$ and $\left[X\right] \longrightarrow \left[\ \right]\overline{X}$ were not ran since the object a_j was missing from the initial multiset). So, the following rules are executed in sequence $\widetilde{l_3}\left[\ \right] \longrightarrow \left[\widetilde{\widetilde{l_3}}\right]$, $\left[\widetilde{\widetilde{l_3}}\right] \longrightarrow \left[\ \right]l_3 L_3$. As a consequence, the symbol that corresponds to the next instruction to be simulated is generated.

If l_h is generated then the computation stops, having in the output region a number of objects a_i, $1 \leq i \leq n$, equals with the contents of register i of M. In this way the execution of the entire register machine program is simulated.

Since one can easily construct a register machine, equivalent with M, that in a successful computation clears its registers except a special designated one (the output register) we have that $NPACC_2(cat, inhR_1) \supseteq NRE$.

Therefore, we have proved the equality $NPACC_2(cat, inhR_1) = NRE$.

4 Conclusions and Further Research

The model we introduced is based on the observation that various chemical reactions within a compartment of a living cell are activated from the neighboring compartments of the cell. We have proved that the family of sets of vectors of numbers generated by P systems with adjoining controlled communication rules when only simple inhibited rules are used equals the family of sets of numbers generated by ET0L systems. We have also proved the computational completeness if, in addition, carriers are used. As a plus, we want to emphasize that similar results can be obtained if, instead of inhibited simple rules, promoted ones are considered.

Trying to get more "realistic", we believe that it is worthwhile to investigate the power of the above systems to whom we add execution times for the rules and to study their properties (for more details we refer to [2]). Another possible line for further research is to investigate the power of the systems not considering the family of sets of vectors of numbers generated as we have done here, but considering the family of Parikh images generated by such systems.

References

1. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa (Eds.): *Multiset Processing*, LNCS 2235, Springer-Verlag, Berlin, 2001.
2. M. Cavaliere, D. Sburlan: Time and Synchronization in Membrane Systems, *Fundamenta Informaticae* 64(1–4), 65–77, 2005.
3. M. Ionescu, D. Sburlan: On P Systems with Promoters/Inhibitors, *Journal of Universal Computer Science*, 10(5), 581–599, 2004.
4. Gh. Păun: Computing with Membranes, *Journal of Computer and System Sciences*, 618(1), 108–143, 2000.

5. Gh. Păun: *Membrane Computing. An Introduction*, Springer-Verlag, Berlin, 2002.
6. A. Salomaa, G. Rozenberg (Eds.): *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.
7. D. Sburlan: Further Results on P Systems with Promoters/Inhibitors. *International Journal of Foundations of Computer Science*, 17, 1 (2006), 205–221;
8. http://psystems.disco.unimib.it/

Several Applications of Spiking Neural P Systems

Mihai Ionescu[1], Dragoş Sburalan[2]

[1] Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
Pl. Imperial Tàrraco 1, 43005 Tarragona, Spain
armandmihai.ionescu@urv.cat
[2] Ovidius University
Faculty of Mathematics and Informatics
Constantza, Romania
dsburlan@univ-ovidius.ro

Summary. In this paper we investigate some applications of Spiking Neural P Systems regarding their capability to solve some classical computer science problems. In this respect it is studied the versatility of such systems to simulate a well known parallel computational model, namely the Boolean circuits. In addition, another notorious application - the sorting - is considered within this framework.

1 Introduction

Spiking neural P systems (shortly called SN P systems) are a class of computing models introduced in [9]. They are using ideas from neural computing, area currently under high investigation, with a focus on spiking neurons (see, e.g., [4], [12], [13]).

The new models are based on the tissue-like and neural-like P systems structure to which various features were added, and can be found on the website of the Membrane Computing community ([21]). For an introduction in the area we refer to [16], while for an up-to-date information regarding P systems one can consult the above mentioned website.

In short, an SN P system consists of a set of neurons placed in the nodes of a graph and sending signals (spikes) along synapses (edges of the graph), under the control of firing rules. One also uses forgetting rules, which remove spikes from neurons. Hence, the spikes are moved and created, destroyed, but never modified (there is only one type of objects in the system).

A generalization of the original model was considered in [15], [3] where rules of the form: $E/a^c \rightarrow a^p; d$ where introduced. The meaning is that when using the rule, c spikes are consumed and p spikes are produced. Because p can be 0 or greater than 0, we obtain at the same time a generalization of both spiking and forgetting rules. Different from the original model of SN P systems, in [10],

parallelism inside a neuron was introduced. By that we mean that when a rule $E/a^c \to a; d$ can be applied (the contents of a neuron is described by the regular expression E), then we apply it as many times as possible in that neuron.

Based on the above features, we investigate their power to simulate boolean gates and circuits. We also introduce here a modality to sort natural numbers (given as number of spikes) with SN P systems in the initial version.

2 Prerequisites

In this section we first introduce the definition of SN P system which we will use during our endeavor, altogether with some explanations on the exhaustive use of the rules. Then, we recall (some) basic notions on boolean functions and circuits.

2.1 SN P systems

A *spiking neural P system* (in short, an SN P system), of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained by the neuron;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^c \to a; d$, where E is a regular expression over O, $c \geq 1$, and $d \geq 0$;
      (2) $a^s \to \lambda$, for some $s \geq 1$, with the restriction that $a^s \in L(E)$ for no rule $E/a^c \to a; d$ of type (1) from R_i;
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$, for $1 \leq i \leq m$ (*synapses*);
4. $out \in \{1, 2, \ldots, m\}$ indicates the *output neuron*.

The rules of type (1) are *firing* (also called *spiking*) *rules*, and the rules of type (2) are called *forgetting rules*. The first ones are applied as follows: if the neuron contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \to a; d$ can be applied, and this means that c spikes are consumed, only $k - c$ remain in the neuron, the neuron is fired, and it produces one spike after d time units (a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized). If $d = 0$, then the spike is emitted immediately, if $d = 1$, then the spike is emitted in the next step, and so on. In the case $d \geq 1$, if the rule is used in step t, then in steps $t, t + 1, t + 2, \ldots, t + d - 1$ the neuron is *closed*, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends a spike along it, then the spike is lost). In step $t + d$, the neuron spikes and becomes again open, hence can receive spikes (which can be used in step $t + d + 1$).

A spike emitted by a neuron σ_i is replicated and goes to all neurons σ_j such that $(i, j) \in syn$.

The forgetting rules, are applied as follows: if the neuron contains exactly s spikes, then the rule $a^s \to \lambda$ can be used, and this means that all s spikes are removed from the neuron.

In each time unit, in each neuron which can use a rule we have to use a rule, either a firing or a forgetting one. Because two firing rules $E_1/a^{c_1} \to a; d_1$ and $E_2/a^{c_2} \to a; d_2$ can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and then one of them is chosen non-deterministically. Note however that we cannot interchange a firing rule with a forgetting rule, as all pairs of rules $E/a^c \to a; d$ and $a^s \to \lambda$ have disjoint domains, in the sense that $a^s \notin L(E)$.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron. Starting from the initial configuration and applying the rules, we can define transitions among configurations. A transition between two configurations C_1, C_2 is denoted by $C_1 \Longrightarrow C_2$. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where all neurons are open and no rule can be used.

With any computation, halting or not, we associate a *spike train*, a sequence of digits 0 and 1, with 1 appearing in positions $1 \leq t_1 < t_2 < \ldots$, indicating the steps when the output neuron sends a spike out of the system (we also say that the system itself spikes at that time). With any spike train containing at least two spikes we associate a result, in the form of the number $t_2 - t_1$; we say that this number is computed by Π. By definition, if the spike train contains only one occurrence of 1, then we say that we have computed the number zero. The set of all numbers computed in this way by Π is denoted by $N_2(\Pi)$ (the subscript indicates that we only consider the distance between the first two spikes of any computation). Then, by $Spik_2P_m(rule_k, cons_q, forg_r)$ we denote the family of all sets $N_2(\Pi)$ computed as above by spiking neural P systems with at most $m \geq 1$ neurons, using at most $k \geq 1$ rules in each neuron, with all spiking rules $E/a^c \to a; t$ having $c \leq q$, and all forgetting rules $a^s \to \lambda$ having $s \leq r$. When one of the parameters m, k, q, r is not bounded, it is replaced with $*$.

In this paper, we use SN P systems of the form introduced above, but using the rules in the exhaustive way. Namely if a rule $E/a^c \to a^p; d$ is associated with a neuron σ_i which contains k spikes, then the rule is enabled (we also say *fired*) if and only if $a^k \in L(E)$. Using the rule means the following. Assume that $k = sc+r$, for some $s \geq 1$ (this means that we must have $k \geq c$) and $0 \leq r < c$ (the remainder of dividing k by c). Then sc spikes are consumed, r spikes remain in the neuron σ_i, and sp spikes are produced and sent to the neurons σ_j such that $(i, j) \in syn$ (as usual, this means that the sp spikes are replicated and exactly sp spikes are sent to each of the neurons σ_j). In the case of the output neuron, sp spikes are also sent to the environment. Of course, if neuron σ_i has no synapse leaving from it, then the produced spikes are lost.

We stress two important features of this models. First, it is important to note that only one rule is chosen and applied, the remaining spikes cannot evolve by another rule. For instance, even if a rule $a(aa)^*/a \to a; 0$ exists, it cannot be used for the spike remaining unused after applying the rule $a(aa)^*/a^2 \to a; 0$. Second, is that the covering of the neuron is checked only for enabling the rule, not step by step during its application. For instance, the rule $a^5/a^2 \to a; 0$ has the same effect as $a(aa)^*/a^2 \to a; 0$ in the case of a neuron containing exactly 5 spikes: the rule is enabled, 4 spikes are consumed, 2 are produced; both applications of the rule are concomitant, not one after the other, hence all of them have the same enabling circumstances.

If several rules of a neuron are enabled at the same time, one of them is nondeterministically chosen and applied. The computations proceed as in the SN P systems with usual rules, and a spike train is associated with each computation by writing 0 for a step when no spike exits the system and 1 within a step when one or more spikes exit the system. Then, a number is associated – and said to be generated/computed by the respective computation – with a spike train containing at least two occurrences of the digit 1, in the form of the steps elapsed between the first two occurrences of 1 in the spike train. Number 0 is computed by computations whose spike trains contain only one occurrence of 1.

2.2 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 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 $T \vDash \varphi$, and $f(t)$ is false if $T \nvDash \varphi$, where $T(x) = t_i$ for $i = 1, \ldots, n$.

There are three primary boolean functions that are widely used: 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. The AND function - the output of an AND function is true only if its first input and its second input and its third input (etc.) are all true. The OR function - the output of an OR function is true if the first input is true or the second input is true or the third input is true (again, etc.). 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 that 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 the

"in-degree" (number of incoming edges) equal to 0, 1, or 2. Also, each gate $i \in V$ has a *sort* $s(i)$ associated with it, where $s(i) \in \{true, false, \vee, \wedge, \neg\} \cup \{x_1, x_2, \dots\}$. If $s(i) \in \{true, false\} \cup \{x_1, x_2, \dots\}$, then the in degree of i is 0, that is, i must have no 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, $X(C) = \{x \in X \mid s(i) = x$ for some gate i 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.

3 Simulating Logical Gates and Circuits

In this section we show how SNP systems can simulate logical gates. We consider that input is given in one neuron while the output will be collected from the output neuron of the system. Boolean value 1 is encoded in the spiking system by two spikes, hence a^2, while 0 is encoded as one spike.

We collect the result as follows. If the output neuron fires two neurons in the second step of the computation, then the boolean calculus computed by the system is 1. If it fires only one spike, then the result is 0.

3.1 Simulating Logical Gates

Lemma 1. *Boolean AND gate can be simulated by SN P systems using two neurons and no delay on the rules, in two steps.*

Proof. We construct the SNP system

$$\Pi_{AND} = (\{a\}, \sigma_1, \sigma_2, \{(1, 2)\}, 2),$$

where:

- $\sigma_1 = (0, \{a \to a; 0\})$,
- $\sigma_2 = (0, \{a^2 \to a; 0, a^3 \to a; 0, a^4/a^2 \to a; 0\})$,

The system is given in its initial configuration in Figure 1 (a.). This gives us the opportunity to introduce the way we graphically represent a SN P system: as a directed graph, with the neurons as nodes and the synapses indicated by arrows. Each neuron has inside its specific rules and the spikes present in the initial configuration.

The functioning of the system is rather simple. Suppose in neuron 1 we introduce three spikes. This means we compute the logical AND between 1 and 0 (or 0 and 1). Neuron 1 fires and, in the same time, all three spikes are sent to the output neuron. In the second step of the computation, the output neuron uses rule $a^3 \to a; 0$ and the correct result (in this case 0) is sent to the environment.

If 4 spikes are introduced in neuron 1 (the case 11), in the second step of the computation the output neuron will fire using the rule $a^4/a^2 \to a; 0$, and will send two spikes in the environment. The system with the input 00 behaves similarly to the 01 or 10 cases. We have shown how the system we have constructed gives the right answer in two computational steps and gets back to its initial configuration for a further use, if necessary.

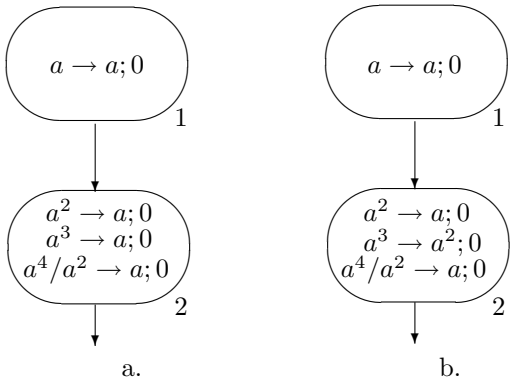


Figure 1. SN P systems simulating **AND** (a.) and **OR** (b.) gates

We want to emphasize here that no "extended" rule was used. Of course, a rule $a^4 \to a^2$ can substitute, with the same effect, the rule we have preferred above (namely $a^4/a^2 \to a; 0$) but, in simulating boolean gates, we have tried to minimize the use of such rules. An extended rule is used only once in simulating Boolean gates, more precisely in the simulation of OR gate.

If in the system above, in the output neuron, we change only the rule $a^3 \to a; 0$ (with the rule $a^3 \to a^2; 0$) we obtain the OR gate.

Lemma 2. *Boolean OR gate can be simulated by SN P systems using two neurons and no delay on the rules, in two steps.*

Proof. In order to simulate OR gate we construct a similar system to the one above. Hence,

$$\Pi_{OR} = (\{a\}, \sigma_1, \sigma_2, \{(1,2)\}, 2),$$

with:

- $\sigma_1 = (0, \{a \to a; 0\})$,
- $\sigma_2 = (0, \{a^2 \to a; 0, a^3 \to a^2; 0, a^4/a^2 \to a; 0\})$.

The system works in the same manner as described above. The difference is when the output neuron receives three spikes in the first step of the computation. In the second step it does not fire only one, but two (hence the output 1), thus giving the right answer for the input 01 (or 10).

We now pass to the simulation of logical gate NOT.

Lemma 3. *Boolean NOT gate can be simulated by SNP systems using eight neurons, no delay on the rules, in two steps.*

Proof. We first want to stress that in simulating this gate we did not use any extended rules. The case when such rules are used is left to the reader.

Let us construct the following SN P system:

$$\Pi_{NOT} = (\{a\}, \sigma_1, \sigma_2, \cdots, \sigma_8, syn, 2),$$

and:

- $\sigma_1 = (0, \{a \to a; 0\})$,
- $\sigma_2 = (a^3, \{a^4/a^2 \to a; 0, a^5 \to a; 0\})$,
- $\sigma_3 = \sigma_4 = \sigma_5 = (0, \{a/a \to a; 0, a^2/a^2 \to \lambda\})$,
- $\sigma_6 = \sigma_7 = \sigma_8 = (0, \{a^2/a^2 \to a; 0, a/a \to \lambda\})$,
- $syn = \{(1,2), (2,3), (3,2), (2,4), (4,2), (2,5), (5,2), (2,6), (6,2), (2,7), (7,2), (2,8), (8,2)\}$.

Let us emphasize that in order to simulate boolean gate NOT, in the initial configuration, neuron 2 contains 3 spikes, which, once used to correctly simulate the gate, have to be present again in the neuron such that the system returns to its initial configuration. This is done with the help of 3 neurons (3, 4, and 5 if the result of the gate is 1, and 6, 7, and 8 otherwise) which in step 3 of the computation refill neuron 2 with 3 spikes.

If the input in the boolean gate is 1, then 2 spikes are placed in neuron 1 which will be sent to neuron 2 in one computational step (applying the rule $a \to a; 0$). There, the rule $a^5 \to a; 0$ is used and the system expels one spike to the environment, corresponding to 0. In the same time one spike is also sent to the neurons 3, 4, 5, 6, 7, and 8. Neurons 3, 4, and 5 will send it to neuron 2 (which regains its initial 3 spikes) while neurons 6, 7, and 8 are deleting it.

If only one spike is given in neuron one (hence the input 0), it is sent immediately to neuron 2. Here, at the end of the first computational step there will be 4 spikes which will be consumed (and sent to the environment) in the second step of the computation when the rule $a^4/a^2 \to a; 0$ is used twice. The two spikes

(representing the result 1 for the input 0) are also sent to neurons 3, 4, 5, 6, 7, and 8. This time the spikes are used by neurons 6, 7, and 8 which are sending one spike to neuron 2, while neurons 3, 4 and 5 are forgetting them.

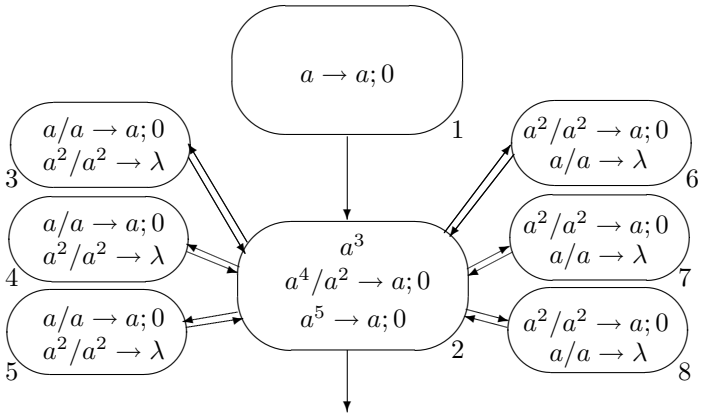


Figure 2. SN P systems simulating **NOT** gate

After showing how SN P systems can simulate logical gates, we pass to the simulation of circuits.

3.2 Simulating Circuits

Next, we are presenting an example of how to construct a SN P system to simulate a Boolean circuit designed to evaluate a Boolean function. Of course, in our goal we are using the systems Π_{AND}, Π_{OR}, and Π_{NOT} constructed before, to which we add extra neurons to synchronize the system for a correct output.

We start with the same **example** considered in [1] and [11] and we have 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 circuit corresponding to the above formula is depicted in Figure 3, and in Figure 4 we have depicted the spiking system assigned to it.

In order for the system that simulates the circuit to output the correct result it is necessary for each sub-system (that simulates the gates AND, OR, and NOT) to receive the input from the above gate(s) at the same time. To this aim, we have to add (pairs of) synchronization neurons, initially empty with a single rule inside ($a \to a; 0$). Note that in Figure 4. we have added such a pair of neurons in order for the output of the first AND gate to enter gate OR at the same time with the output of NOT gate (at the end of the fourth step of the computation).

Having the overall image of the functioning of the system, let us give some more details on the simulation of the above formula. For that we construct the SN P system

$$\Pi_C = (\Pi^{(1)}_{AND}, \Pi^{(2)}_{AND}, \Pi^{(3)}_{NOT}, \Pi^{(4)}_{OR})$$

formed by the sub-SN P systems for each gate, and we obtain the unique result as follows:

1. for every gate of the circuit with inputs from the input gates we have a SN P system to simulate it. The input is given in neuron labeled 1 of each gate;
2. for each gate which has at least one input coming as an output of a previous gate we construct a SN P system to simulate it by "constructing" a synapse between the output neuron of the gate from which the signal (spike) comes and the input neuron of the system that simulates the new gate.
   Note that if synchronization is needed the new synapse is constructed from the output neuron of the output gate to the first neuron in the (pair of) neurons used for synchronization and from here another synapse is constructed to the input of the new gate in the circuit.

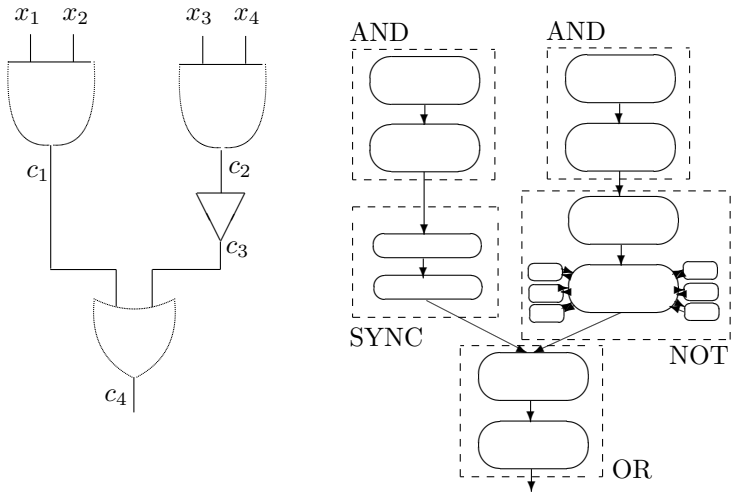


Figure 3. Boolean Circuit

For the above formula and the circuit depicted in Figure 3 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 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 passes through the two synchronization neurons.
- The input enters in the first neuron of OR gate, and SN P system $\Pi^{(4)}_{OR}$ completes its task. The result of the computation for OR gate (which is the result of the global P system), is sent into the environment of the whole system.

Based on the previous explanations the following result holds:

Theorem 1. *Every Boolean circuit α, whose underlying graph structure is a rooted tree, can be simulated by a SN P system, Π_α, in linear time. Π_α is constructed from SN P systems of type Π_{AND}, Π_{OR} and Π_{NOT}, by reproducing in the architecture of the neural structure, the structure of the tree associated to the circuit.*

4 A Sorting Algorithm

We pass now to a different problem SN P systems can solve, namely to sort n natural numbers, this time not using the rules in the exhaustive way, but as in the original definition of such systems.

We first exemplify our sorting procedure through an example. Let us presume we want to sort the natural numbers 1, 3, and 2, given in this order. For that we construct the following system given only in its pictorial format below:

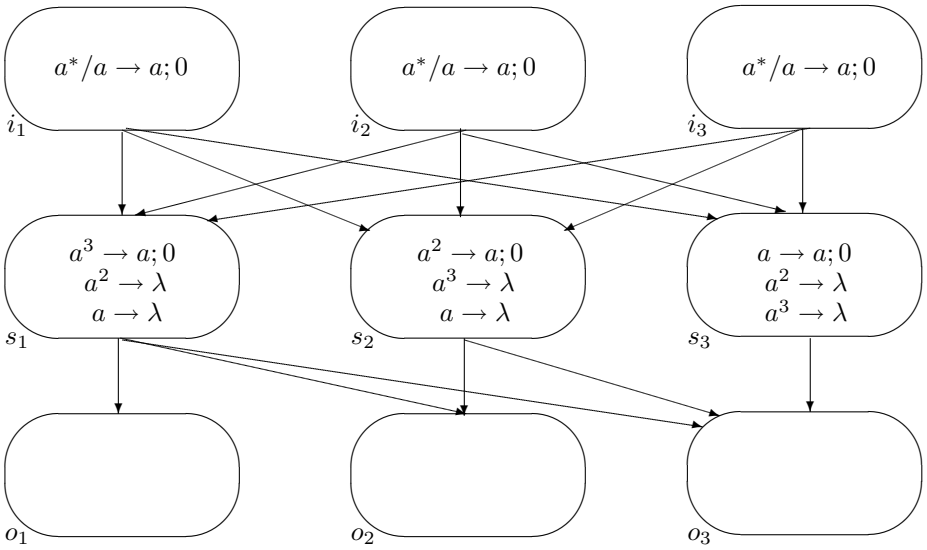


Figure 4. Sorting three natural numbers

We encode natural numbers in the number of spikes (1 – one spike, 3 – three spikes, 2 – two spikes) which we input in the first line of the system (hence in the neurons labeled i_1, i_2, an i_3). It can be noticed that the neurons in the first layer of the structure are having the same rule inside ($a^*/a \to a; 0$) and outgoing synapses to all the neurons in the second layer of the structure (the ones denoted s_1, s_2, and s_3). Neuron labeled s_1 has outgoing synapses with all neurons in the third layer of the system, only one spiking rule inside ($a^3 \to a; 0$, where 3 is the

number of numbers that have to be sorted), and two deletion rules ($a^2 \to \lambda$, and $a \to \lambda$). For the other neurons in the second layer, the exponent of the firing rule decreases one by one as well as the synapses with the neurons from the third layer of the system.

In the initial configuration of the system we have one spike in neuron i_1, three spikes in neuron i_2 and 2 spikes in neuron i_3. In the *first step* of the computation, one spike from each neuron is consumed and sent to neurons from the second layer of the system. Each of them receives the same number of spikes, namely 3.

In the *second step* of the computation, neuron labeled s_1 consumes all three spikes previously received and fires to neurons o_1, o_2 and o_3. Hence, each neuron from the output layer has one spike inside. The other neurons from the second layer delete the three spikes they have received. In the same time neurons i_2 and i_3 fire again sending 2 spikes (one each) to all neurons from the second layer.

In the *third step* of the computation, neuron s_2 fires only to neurons o_2 and o_3 (so, they will have one more spike inside, hence 2, while o_1 remains with only one spike), the other spikes from neurons s_1 and s_3 being deleted. In the same time neuron i_2 refills the neurons in the second layer of the system with one spike, which will be consumed in the *forth step* of the computation by neuron s_3 and sent to the output neuron o_3.

So, in the last step of the computation there are: 1 spike in the neuron o_1, 2 spikes in the neuron o_2, and 3 spikes in the neuron o_3.

We pass now to the general case, constructing only the system in the pictorial form:

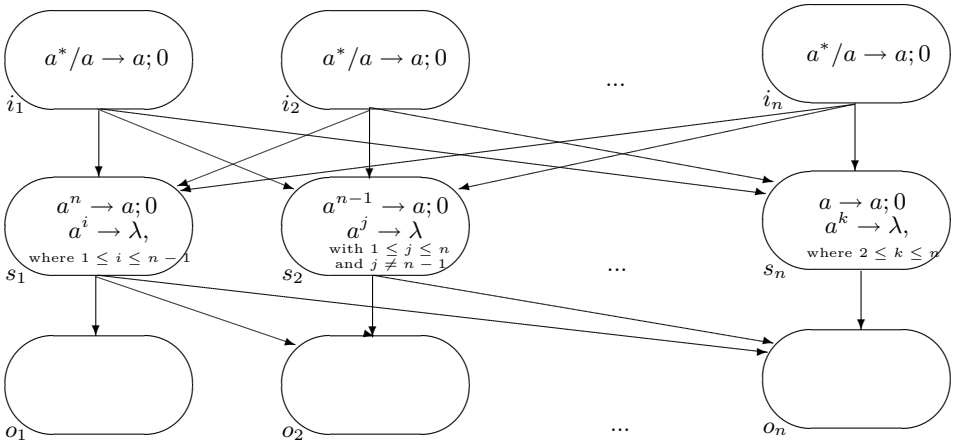


Figure 5. Sorting n natural numbers

The functioning of the system is similar with the one described in the above example, and consequently we have the following result.

Theorem 2. *SN P systems can sort a vector of natural numbers where each number is given as number of spikes introduced in the neural structure.*

Based on the above construction, the time complexity (measured as usually as the number of configurations reached during the computation) is $O(n)$. Although the time complexity is better than the "classical", sequential algorithm, in this case one can notice that the construction presented depends on the magnitude of the numbers to be sorted.

5 Final Remarks

Spiking neural P systems are a versatile formal model of computation that can be used for designing efficient parallel algorithms for solving known computer science problems. Here we firstly studied the ability of SN P systems to efficiently simulate Boolean circuits since, apart for being a well known computational model, there exists many "fast" algorithms solving various problems. In addition, this simulation, enriched with some "memory modules" (given in the form of some SN P sub-systems), may constitute an alternative proof of the computational completeness of the model.

Another issue studied here regards the sorting of a vector of natural numbers using SN P systems. In this case, due to its parallel features, the obtained time complexity for the proposed algorithm overcome the classical sequential ones.

Several open problems arose during our research. For instance, in case of Boolean circuits the simulation is done for such circuits whose underlying graphs have rooted tree structures, therefore a constraint that need further investigations.

In what regards the sorting algorithm, the presented construction depends on the magnitude of the numbers to be sorted. We conjecture that this inconvenient might be eliminated. Also, we conjecture that further improvements concerning time complexity can be made.

Acknowledgements

The work of the authors was supported as follows. M. Ionescu: fellowship "Formación de Profesorado Universitario" from the Spanish Ministry of Education, Culture and Sport.

References

1. R. Ceterchi, D. Sburlan: Simulating Boolean Circuits with P Systems, *LNCS*, 2933, 104–122, 2004.
2. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In [5], Vol. I, 169–194.

3. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In [5], Vol. I, 241–265.
4. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
5. M.A. Gutiérrez-Naranjo et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing*, Febr. 2006, Fenix Editora, Sevilla, 2006.
6. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. In [5], Vol. II, 105–136, and *Theoretical Computer Sci.*, to appear.
7. O.H. Ibarra, S. Woodworth: Characterizations of some restricted spiking neural P systems. In *Pre-proceedings of Seventh Workshop on Membrane Computing, WMC7*, Leiden, The Netherlands, July 2006, 387–396.
8. O.H. Ibarra, S. Woodworth, F. Yu, A. Păun: On spiking neural P systems and partially blind counter machines. In *Proceedings of Fifth Unconventional Computation Conference, UC2006*, York, UK, September 2006, 123–135.
9. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
10. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with an exhaustive use of rules, International Journal of Unconventional Computing, accepted.
11. M. Ionescu, T.-O. Ishdorj: Boolean Circuits and a DNA Algorithm in Membrane Computing. *LNCS*, 3850, 272–291.
12. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
13. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
14. M. Minsky: *Computation – Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.
15. A. Păun, Gh. Păun: Small universal spiking neural P systems. In [5], Vol. II, 213–234, and *BioSystems*, in press.
16. Gh. Păun: *Membrane Computing – An Introduction*. Springer, Berlin, 2002.
17. Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. *LNCS* 4036, 2006, 20–35.
18. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
19. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted 2005.
20. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*, 3 volumes. Springer-Verlag, Berlin, 1997.
21. The P Systems Web Page: `http://psystems.disco.unimib.it`.

On the Computational Power of Spiking Neural P Systems

Alberto Leporati, Claudio Zandron,
Claudio Ferretti, Giancarlo Mauri

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Via Bicocca degli Arcimboldi 8, 20126 Milano, Italy

{leporati,zandron,ferretti,mauri}@disco.unimib.it

Summary. In this paper we study some computational properties of spiking neural P systems. In particular, we show that by using nondeterminism in a slightly extended version of spiking neural P systems it is possible to solve in constant time both the numerical **NP**–complete problem Subset Sum and the strongly **NP**–complete problem 3-SAT. Then, we show how to simulate a universal *deterministic* spiking neural P system with a deterministic Turing machine, in a time which is polynomial with respect to the execution time of the simulated system. Surprisingly, it turns out that the simulation can be performed in polynomial time with respect to the *size of the description* of the simulated system only if the regular expressions used in such a system are of a very restricted type.

1 Introduction

Membrane systems (also called *P systems*) were introduced in [16] 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. Usually, the rules are applied in a nondeterministic and maximally parallel way; moreover, all the objects that may evolve are forced to evolve. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be applied. The result of a computation is the multiset of objects contained into an *output membrane*, or emitted from the skin of the system. For a systematic introduction to P systems we refer the reader to [18], whereas the latest information can be found in [23].

In an attempt to pass from cell-like to tissue-like architectures, in [13] *tissue P systems* were defined, in which cells are placed in the nodes of a (directed) graph. Since then, this model has been further elaborated, for example, in [4] and [19], with recent results about both theoretical properties [1] and applications [14]. This evolution has led to explore also *neural-like* architectures, yielding to the introduction of *spiking neural P systems* (SN P systems, for short) [8], based on the neurophysiological behavior of neurons sending electrical impulses (*spikes*) along axons to other neurons. We recall that this biological background has already led to several models in the area of neural computation, e.g., see [11, 12, 6].

Similarly to tissue P systems, in SN P systems the cells (neurons) are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consist of a number of copies of a single object type, called the *spike*. The *firing rules* assigned to a cell allow a neuron to send information to other neurons in the form of electrical impulses (also called spikes) which are accumulated at the target cell. The application of the rules depends on the contents of the neuron; in the general case, applicability is determined by checking the contents of the neuron against a regular set associated with the rule. As inspired from biology, after a cell sends out spikes it becomes "closed" (inactive) for a specified period of time, that reflects the refractory period of biological neurons. During this period, the neuron does not accept new inputs and cannot "fire" (that is, emit spikes). Another important feature of biological neurons is that the length of the axon may cause a time delay before a spike arrives at the target. In SN P systems this delay is modeled by associating a delay parameter to each rule which occurs in the system. If no firing rule can be applied in a neuron, there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

In the original model of SN P systems defined in [8], computations occur as follows. A *configuration* specifies, for each neuron of the system, the number of spikes it contains and the number of computation steps after which the neuron will become "open" (that is, not closed). Starting from an initial configuration, a positive integer number is given in input to a specified *input neuron*. The number is encoded as the interval of time steps elapsed between the insertion of two spikes into the neuron. To pass from a configuration to another one, for each neuron a rule is chosen among the set of applicable rules, and is executed. The computation proceeds in a sequential way into each neuron, and in parallel among different neurons. Generally, a computation may not halt. However, in any case the output of the system is considered to be the time elapsed between the arrival of two spikes in a designated *output cell*. Defined in this way, SN P systems compute functions of the kind $f : \mathbb{N} \to \mathbb{N}$ (they can also indirectly compute functions of the kind $f : \mathbb{N}^k \to \mathbb{N}$ by using a bijection from $\mathbb{N}^k$ to $\mathbb{N}$). By ignoring the output neuron we can define *accepting* SN P systems, in which the natural number given in input is accepted if the computation halts, and rejected otherwise. On the other hand, by ignoring the input neuron (and thus starting from a predefined input configuration) we can define *generative* SN P systems.

In [8] it was shown that generative SN P systems are universal, that is, can generate any recursively enumerable set of natural numbers. Moreover, a characterization of semilinear sets was obtained by spiking neural P systems with a bounded number of spikes in the neurons. These results can also be obtained with even more restricted forms of spiking P systems; for example, [7] shows that at least one of these features can be avoided while keeping universality: time delay (refractory period) greater than 0, forgetting rules, outdegree of the synapse graph greater than 2, and regular expressions of complex form. Finally, in [20] the behavior of spiking neural P systems on infinite strings and the generation of infinite sequences of 0 and 1 was investigated, whereas in [2] spiking neural P systems were studied as language generators (over the binary alphabet $\{0, 1\}$).

The rest of this paper is organized as follows. In section 2 we give some mathematical preliminaries, and we define the standard version of SN P systems (as found in [9]) as well as a slightly extended version. In section 3 we show how the **NP**–complete problems Subset Sum and 3-SAT can be solved in constant time by exploiting nondeterminism in our extended SN P systems. In section 4 we turn our attention to deterministic systems, and we show how to simulate them by using deterministic Turing machines. Section 5 concludes the paper and gives some directions for future research.

2 Preliminaries

Let us start by recalling the standard definition of a spiking neural P system, taken from [9]. A *spiking neural membrane system* (SN P system, for short), of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, with $1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^c \rightarrow a; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$ are integer numbers; if $E = a^c$, then it is usually written in the following simplified form: $a^c \rightarrow a; d$;
      (2) $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a; d$ of type (1) from R_i, we have $a^s \notin L(E)$ (where $L(E)$ denotes the regular language defined by E);
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons;
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and the *output* neurons of Π.

The rules of type (1) are called *firing* (also *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains $k \geq c$ spikes, and $a^k \in L(E)$, then the rule $E/a^c \to a; d \in R_i$ can be applied. The execution of this rule removes c spikes from σ_i (thus leaving $k - c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i, j) \in syn$. If $d = 0$, then the spike is immediately emitted, otherwhise it is emitted afted d computation steps of the system. (Observe that, as usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.) If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed*, so that it cannot receive new spikes (if a neuron has a synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost), and cannot fire new rules. In the step $t + d$, the neuron spikes and becomes again open, so that it can receive spikes (which can be used starting with the step $t + d + 1$) and select rules to be fired.

Rules of type (2) are called *forgetting* rules, and are applied as follows: if the neuron σ_i contains *exactly* s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i. Note that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. Since two firing rules, $E_1 : a^{c_1} \to a; d_1$ and $E_2 : a^{c_1} \to a; d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron. In such a case, only one of them is chosen nondeterministically. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

The *initial configuration* of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron, with all neurons being open. During the computation, a configuration is described by both the number of spikes present in each neuron and by the state of each neuron, which can be expressed as the number of steps to count down until it becomes open (this number is zero if the neuron is already open). A *computation* in a system as above starts in the initial configuration. In order to compute a function $f : \mathbb{N}^k \to \mathbb{N}$, we introduce k natural numbers $n_1, n_2, \ldots, n_k$ in the system by "reading" from the environment a binary sequence $z = 0^b 10^{n_1} 10^{n_2} 1 \ldots 10^{n_k} 10^g$, for some $b, g \geq 0$; this means that the input neuron of Π receives a spike in each step corresponding to a digit 1 from the string z. Note that we input exactly $k + 1$ spikes. The result of the computation is also encoded in the distance between two spikes: we impose to the system to output exactly two spikes and halt (sometimes after the second spike) hence producing a train spike of the form $0^{b'} 10^r 10^{g'}$, for some $b', g' \geq 0$ and with $r = f(n_1, n_2, \ldots, n_k)$.

If we use an SN P system in the *generative* mode, then no input neuron is considered, hence no input is taken from the environment; we start from the initial configuration, and the distance between the first two spikes of the output neuron (or other numbers, see the discussion in [9]) is the result of the computation.

Dually, we can ignore the output neuron, and if the computation halts, then the number is accepted.

We define the *description size* of an SN P system Π as the number of bits which are necessary to describe it. Since the alphabet O is fixed, no bits are necessary to define it. In order to represent *syn* we need at most m^2 bits, whereas we can represent the values of *in* and *out* by using $\log m$ bits each. Every neuron σ_i requires to specify a natural number n_i, and a set R_i of rules. Each rule requires to specify its type (firing or forgetting), which can be done with 1 bit, and in the worst case it requires to specify a regular expression and two natural numbers. If we denote by N the maximum natural number that appears in the definition of Π, R the maximum number of rules which occur in its neurons, and S the maximum size required by the regular expressions that occur in Π (more on this later), then we need a maximum of $\log N + R(1 + S + 2\log N)$ bits to describe every neuron of Π. Hence, to describe Π we need a total of $m^2 + 2\log m + m\big(\log N + R(1+S+2\log N)\big)$ bits. Note that this quantity is polynomial with respect to m, R, S and $\log N$. Since the regular languages determined by the regular expressions that occur in the system are *unary* languages, the strings of such languages can be bijectively identified by their lengths. Hence, when writing the regular expression E, instead of writing unions, concatenations and Kleene closures among strings we can do the same by using the lengths of such strings. In this way we obtain a representation of E which is exponentially more compact than the usual representation of regular expressions. As we will see in section 4, this compact representation will yield some difficulties when we will simulate a deterministic accepting SN P system by a deterministic Turing machine.

In what follows it will be convenient to consider also a slightly extended version of SN P systems. Precisely, we will allow rules of the type $E/a^c \to a^p; d$, where $c \geq 1$, $p \geq 0$ and $d \geq 0$ are integer numbers. The semantics of this kind of rules is as follows: if the contents of the neuron matches the regular expression E, then the rule can be applied. When the rule is applied, c spikes are removed from the contents of the neuron and p spikes are prepared to be delivered to all the neurons which are directly connected (through an arc of *syn*) with the current neuron. If $d = 0$, then these p spikes are immediately sent, otherwise the neuron becomes closed for the next d computation steps, after which the p spikes will be sent. As before, a closed neuron does not receive spikes from other neurons, and does not apply any rule. If $p = 0$, then we obtain a forgetting rule as a particular case of our general rules.

Also in the extended SN P systems it may happen that, given two rules $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$, if $L(E_1) \cap L(E_2) \neq \emptyset$ then for some contents of the neuron both the rules can be applied. In such a case, we nondeterministically choose one of them. Note that we do not require that forgetting rules are applied only when no firing rule can be applied. We say that the system is *deterministic* if, for every neuron that occurs in the system, any two rules $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$ in the neuron are such that $L(E_1) \cap L(E_2) = \emptyset$. This means

that, for any possible contents of the neuron, at most one of the rules that occur in the neuron may be applied.

By using an *input neuron* and an *output neuron*, we have SN P systems that compute functions of the kind $f : \mathbb{N} \to \mathbb{N}$, and hence we cover both the generative and the accepting cases. If $out = 0$, then it is understood that the output is sent to the environment (as the number of spikes produced by the system, as the distance between the first two spikes, etc.). As usual, to use an SN P system in the generative mode we do not consider the input neuron, and thus no input is taken from the environment; we start from the initial configuration, and the distance between the first two spikes of the output neuron (or the number of spikes contained into the output neuron at the end of the computation, as discussed above) is the result of the computation. Note that generative SN P systems are inherently nondeterministic, otherwise they would always reproduce the same sequence of computation steps, and hence the same output. Dually, we can ignore the output neuron to obtain an accepting SN P system. We input a number in the system as the distance between two spikes entering the input neuron (or the number of spikes that occur in the input neuron in the initial configuration) and, if the computation halts, then the number is accepted.

The *description size* of an extended SN P system is defined exactly as we did for standard systems, the only difference being that now we require (at most) three natural numbers to describe a rule.

3 Solving NP–complete Problems with Extended Spiking Neural P Systems

In this section we show that *nondeterministic* SN P systems are very powerful computing devices, at least in the extended version defined in the previous section: in fact, they are able to solve **NP**–complete problems in a *constant* number of computation steps.

3.1 Solving the Subset Sum problem

Let us first consider the Subset Sum problem, which can be stated as follows.

Problem 1. Name: Subset Sum.

- Instance: a (multi)set $V = \{v_1, v_2, \ldots, v_n\}$ of positive integer numbers, and a positive integer number S
- Question: is there a subset $B \subseteq V$ such that $\sum_{b \in B} b = S$?

If we allow to nondeterministically choose among the rules which occur in the neurons, then the extended SN P system depicted in Figure 1 solves any given instance of Subset Sum in a constant number of steps. We emphasize the fact

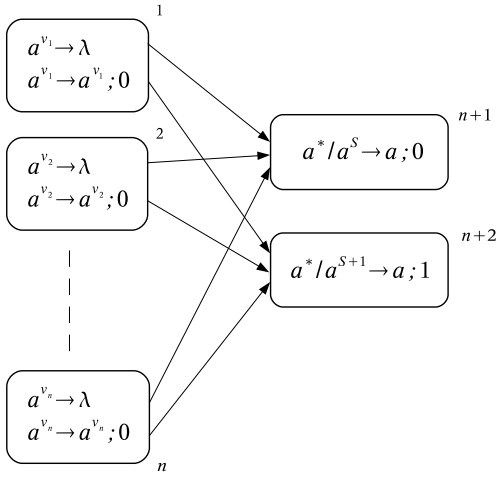


Fig. 1. A nondeterministic extended SN P system that solves the SUBSET SUM problem in constant time

that such a solution occurs in the *semi-uniform* setting, that is, for every instance of SUBSET SUM we build an SN P system that specifically solves that instance.

Let $(V = \{v_1, v_2, \ldots, v_n\}, S)$ be the instance of SUBSET SUM to be solved, and let $B \subseteq V$. In the initial configuration of the system, the leftmost neurons contain (from top to bottom) $v_1, v_2, \ldots, v_n$ spikes, respectively, whereas the rightmost neurons contain zero spikes each. In the first step of computation, in each of the leftmost neurons of the SN P system depicted in Figure 1 it is nondeterministically chosen whether to include or not the element v_i in B; this is accomplished by nondeterministically choosing among one rule that forgets v_i spikes (in such a case, $v_i \notin B$) and one rule that propagates v_i spikes to the rightmost neurons. At the beginning of the second step of computation a certain number N of spikes, that corresponds to the sum of the v_i which have been chosen, occurs in the rightmost neurons. We have three possible cases:

- $N < S$: in this case neither the rule $a^S \to a; 0$ nor the rule $a^{S+1} \to a; 1$ (which occur in the neuron at the top and at the bottom of the second layer, respectively) fire, and thus no spike is emitted to the environment;
- $N = S$: only the rule $a^S \to a; 0$ fires, and emits a single spike to the environmnent. No further spikes are emitted;
- $N > S$: both the rules $a^S \to a; 0$ and $a^{S+1} \to a; 1$ fire. The first rule immediately sends one spike to the environment, whereas the second rule sends another spike at the next computation step (due to the delay associated with the rule).

Hence, by counting the number of spikes emitted to the environment at the second and third computation steps we are able to read the solution of the given instance of Subset Sum: the instance is positive if and only if a single spike is emitted.

The formal definition of the extended (generating) SN P system depicted in Figure 1 is as follows:

$$\Pi = (\{a\}, \sigma_1, \ldots, \sigma_{n+2}, syn, out)\,,$$

where:

- $\sigma_i = (v_i, \{a^{v_i} \to \lambda,\ a^{v_i} \to a^{v_i}; 0\})$ for all $i \in \{1, 2, \ldots, n\}$;
- $\sigma_{n+1} = (0, \{a^S \to a; 0\})$;
- $\sigma_{n+2} = (0, \{a^{S+1} \to a; 1)$;
- $syn = \bigcup_{i=1}^{n} \{(i, n+1), (i, n+2)\}$;
- $out = 0$ indicates that the output is sent to the environment.

However, here we are faced with a problem that we have already encountered in [10], and that we will encounter again in the rest of the paper. In order to clearly expose the problem, let us consider the following algorithm that solves Subset Sum using the well-known Dynamic Programming technique [3]. In particular, the algorithm returns 1 on positive instances, and 0 on negative instances.

```
Subset Sum({v_1, v_2, ..., v_n}, S)
for j ← 0 to S
    do M[1, j] ← 0
M[1, 0] ← M[1, v_1] ← 1
for i ← 2 to n
     do for j ← 0 to S
              do M[i, j] ← M[i − 1, j]
                 if j ≥ v_i and M[i − 1, j − v_i] > M[i, j]
                    then M[i, j] ← M[i − 1, j − v_i]
return M[n, S]
```

In order to look for a subset $B \subseteq V$ such that $\sum_{b \in B} b = S$, the algorithm uses an $n \times (S+1)$ matrix M whose entries are from $\{0, 1\}$. It fills the matrix by rows, starting from the first row. Each row is filled from left to right. The entry $M[i, j]$ is filled with 1 if and only if there exists a subset of $\{v_1, v_2, \ldots, v_i\}$ whose elements sum up to j. The given instance of Subset Sum is thus a positive instance if and only if $M[n, S] = 1$ at the end of the execution.

Since each entry is considered exactly once to determine its value, the time complexity of the algorithm is proportional to $n(S+1) = \Theta(nS)$. This means that the difficulty of the problem depends on the value of S, as well as on the magnitude of the values in V. In fact, let $K = max\{v_1, v_2, \ldots, v_n, S\}$. If K is polynomially bounded with respect to n, then the above algorithm works in polynomial time. On the other hand, if K is exponential with respect to n, say $K = 2^n$, then the above algorithm works in exponential time and space. This behavior is usually

referred to in the literature by telling that SUBSET SUM is a *pseudo–polynomial* **NP**–complete problem.

The fact that in general the running time of the above algorithm is not polynomial can be immediately understood by comparing its time complexity with the instance size. The usual size for the instances of SUBSET SUM is $\Theta(n \log K)$, since for conciseness every "reasonable" encoding is assumed to represent each element of V (as well as S) using a string whose length is $O(\log K)$. Here all logarithms are taken with base 2. Stated differently, the size of the instance is usually considered to be the number of bits which must be used to represent in binary S and all the integer numbers which occur in V. If we would represent such numbers using the unary notation, then the size of the instance would be $\Theta(nK)$. But in this case we could write a program which first converts the instance in binary form and then uses the above algorithm to solve the problem in polynomial time with respect to the new instance size. We can thus conclude that the difficulty of a numerical **NP**–complete problem depends also on the measure of the instance size we adopt.

The problem we mentioned above about the SN P system depicted in Figure 1 is that the rules $a^{v_i} \to \lambda$ and $a^{v_i} \to a^{v_i}; 0$ which occur in the leftmost neurons, as well as those that occur in the rightmost neurons, check for the existence of a number of spikes which is exponential with respect to the usually agreed instance size of SUBSET SUM. Moreover, to initialize the system the user has to place a number of objects which is also exponential. This is not fair, because it means that the SN P system that solves the **NP**–complete problem has an exponential size with respect to the binary string which is used to describe it; an exponential effort is thus needed to build the system, that easily solves the problem by working in unary notation (hence in polynomial time with respect to the size of the system, but not with respect to its *description size*). This problem is in some aspects similar to what has been described in [10], concerning traditional P systems that solve **NP**–complete problems.

3.2 The 3-SAT problem

In this section we show that SN P systems are also able to solve non-numerical **NP**–complete problems. Such problems are inherently *strongly* **NP**–complete, that is, they remain **NP**–complete even if the numbers eventually contained into the instance are expressed in unary form. Specifically, we first propose a simple extended SN P system that solves 3-SAT, and then we show that also standard SN P systems are able to solve this problem.

We start by recalling some well known definitions, in order to settle the notation. A *boolean* variable is a variable which can assume one of two possible truth values: TRUE and FALSE. As usually done in the literature, we will denote TRUE by 1 and FALSE by 0. A *literal* is either a directed or a negated boolean variable. A *clause* is a disjunction of literals, whereas a 3-*clause* is a disjunction of exactly three literals. Given a set $X = \{x_1, x_2, \ldots, x_n\}$ of boolean variables, an *assignment* is a mapping $a : X \to \{0, 1\}$ that associates to each variable a truth value. The

number of all possible assignments to the variables of X is 2^n. We say that an assignment *satisfies* the clause C if, assigned the truth values to all the variables which occur in C, the evaluation of C (considered as a boolean formula) gives 1 as a result.

The 3-SAT decision problem is defined as follows.

Problem 2. NAME: 3-SAT.

- INSTANCE: a set $C = \{C_1, C_2, \ldots, C_m\}$ of 3-clauses, built on a finite set $\{x_1, x_2, \ldots, x_n\}$ of boolean variables;
- QUESTION: is there an assignment of the variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in C?

It is well known [5] that 3-SAT is an **NP**–complete problem. In what follows we will equivalently say that an instance of 3-SAT is a boolean formula ϕ_n, built on n free variables and expressed in conjunctive normal form, with each clause containing exactly three literals. The formula ϕ_n is thus the conjunction of the above clauses. Notice that the number m of possible 3-clauses is polynomially bounded with respect to n: in fact, since each clause contains exactly three literals, we can have at most $(2n)^3 = 8n^3$ clauses.

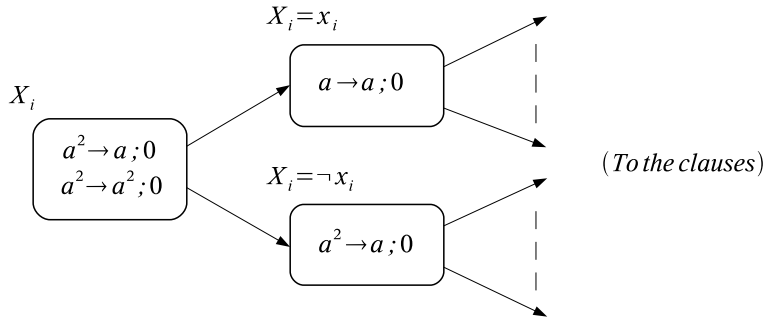


Fig. 2. A nondeterministic extended SN P system that solves the 3-SAT problem in constant time: the module used to build the first two layers (one for each boolean variable)

The extended SN P system that solves 3-SAT is depicted in Figures 2 (the module used to build the first and the second layer) and 3 (third and fourth layers). It is composed by four layers of neurons, the first three of which correspond to the variables, the literals and the clauses of ϕ_n, respectively; the fourth layer is composed by a single neuron, that gathers the spikes produced by the neurons of the third layer. Every neuron in the first layer is connected with its corresponding one or two neurons in the second layer, depending upon whether the literal appears only directed/negated in ϕ_n, or both. Similarly, the neurons of the second layer are connected with those of the third layer according to what literals appear in each clause. Since we are dealing with 3-SAT, every neuron in the third layer will

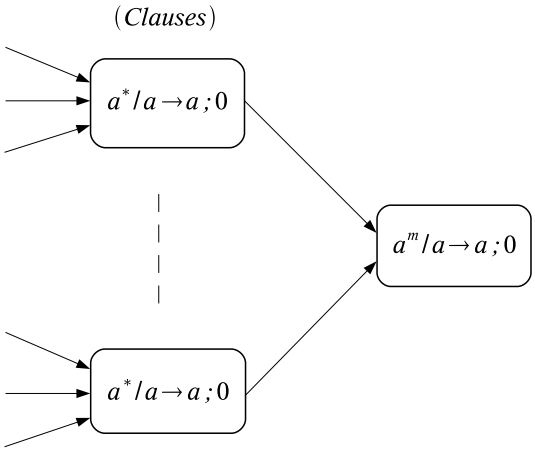


Fig. 3. A nondeterministic extended SN P system that solves the 3-SAT problem in constant time: third and fourth layer (clauses and output neuron)

have exactly three input lines coming from the second layer. Finally, every neuron of the third layer is connected with the output neuron.

During the computation, spikes move from the first to the fourth layer, and then (eventually) are expelled to the environment. In the initial configuration, every neuron in the first layer (which is bijectively associated with one of the n variables of ϕ_n) contains two spikes, whereas all the other neurons are empty. In the first step of the computation, in each neuron of the first layer it is nondeterministically chosen whether to assign 1 or 0 to the corresponding variable, that is, whether to assign 1 to the directed or to the negated literal. This choice is made by nondeterministically choosing between two rules: one that sends two spikes to the next layer, and one that sends a single spike. In the former case, only the neuron that corresponds to the negated literal will fire during the next computation step; in the latter case, only the neuron that corresponds to the directed literal will fire. Since literals are directly connected to the clauses in which they appear, every neuron associated to a clause will receive one spike if and only if at least one of its literals is satisfied. In such a case, one spike is sent to the neuron of the fourth layer, that in this way will contain as many spikes as the number of satisfied clauses. The rule contained in this neuron will fire if and only if there are m spikes, that is, if and only if all the clauses are verified. We can thus conclude that the instance ϕ_n of 3-SAT is positive if and only if (at least) one spike is emitted to the environment.

As mentioned above, it is not necessary to use *extended* SN P systems to solve 3-SAT. In Figure 4 we can see a module which contains only standard rules, whose behavior is almost equivalent to that of the extended module depicted in Figure 2. Neuron number 1 initially contains one spike, which is delivered to all the neurons of the second layer during the first step of computation; note that this neuron is

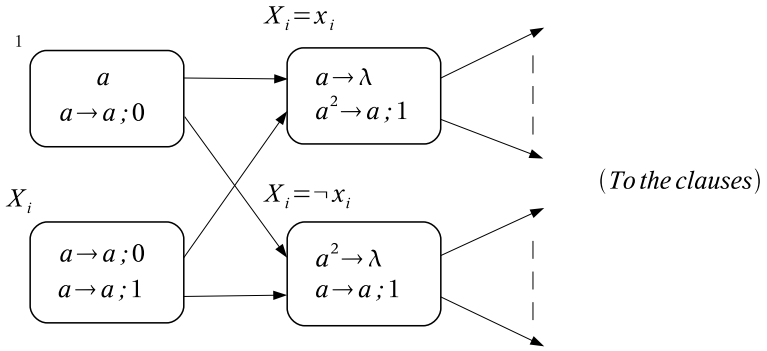


Fig. 4. A nondeterministic standard SN P system that solves the 3-SAT problem in constant time: the module used to build the first two layers (one for each boolean variable). Neuron 1 occurs only one time in the system, and is connected with every neuron of the second layer

not repeated for all modules, but appears only once in the system. All the other neurons of the system are connected exactly in the same way as the corresponding neurons of the extended SN P system described above.

During the first step of computation, every neuron labelled with X_i (in the first layer) nondeterministically chooses whether to immediately emit one spike, or to emit it after a delay of one time step. In the former case, at the end of the first computation step both the neurons of the second layer connected with X_i (corresponding to the directed and to the negated variable, respectively) will contain two spikes. However, these spikes will be removed in the second neuron by applying the rule $a^2 \to \lambda$, whereas in the first neuron they will produce a single spike that will propagate to the third layer after a delay of one time step. Hence, the nondeterministic choice of immediately emitting a spike from the first to the second layer corresponds to assigning 1 to variable x_i. On the other hand, if the spike is emitted from neuron X_i only after a delay of one time step, then at the end of the first computation step both the neurons corresponding to the directed and to the negated variable will contain one spike. This time, the application of the rule $a \to \lambda$ will remove such a spike from the first neuron, whereas the second neuron will emit one spike towards the third layer. Hence, the choice to emit a spike from the first to the second layer after one time step corresponds to assigning 0 to the variable x_i.

Note that the spikes are emitted from the neurons of the second layer with a delay of one computation step; this is done in order to keep such neurons closed during the second computation step, so that the spike (eventually) emitted from the first layer will get lost. However, if desired, we can avoid using delays in the second layer by looking at the answer produced by the system during the fourth computation step, and ignoring what happens afterwards.

4 Simulating Deterministic SN P Systems with Deterministic Turing Machines

Now that we have seen how powerful can be nondeterministic SN P systems, let us turn to *deterministic* SN P systems. In this section we consider a slight extension of the universal SN P system Π defined in [2], and we show that any t steps of computation of Π can be simulated by a deterministic Turing machine in a time which is polynomial both with respect to t and with respect to the *description size* of Π. With respect to the universal SN P system defined in [2], our extension allows to send a predefined number q of spikes to adjacent neurons, instead of sending just one spike. Clearly this modification does not affect universality, and thus also our extended SN P systems are universal.

As we will see, it is possible to simulate these systems in polynomial time mainly because the regular expressions used in the simulated SN P system are of a very restricted form. On the other hand, we will show that if the use of unrestricted regular expressions is allowed, then it is possible to solve the SUBSET SUM **NP**–complete problem by exploting the implicit mechanism used by SN P systems to decide whether a rule can be applied or not.

Theorem 1. *Consider a deterministic accepting SN P system Π, of degree $m \geq 1$, in which:*

- *P and Q are the maximum numbers of spikes that appear in the left and in the right side of the rules, respectively;*
- *D is the maximum delay that appears in the rules;*
- *all the regular expressions are of the following forms: a^i, with $i \leq 3$, or $a(aa)^+$.*

Then, t steps of computation of Π can be simulated by a deterministic Turing machine in a polynomial time with respect to t and to the description size of Π.

Proof. Consider a deterministic accepting SN P system

$$\Pi = (\{a\}, \sigma_1, \ldots, \sigma_m, syn, in)$$

having the characteristics mentioned in the statement of the theorem. We build a deterministic Turing machine M with multiple tapes, such that t steps of computation of Π can be simulated by M in a number of steps which is polynomial with respect to t and to the description size of Π.

To simulate Π with M, we basically need to keep track of the state (number of spikes and number of steps after which the neuron will become open) of each neuron. In general, the simulation of a single open neuron proceeds by first looking for the (unique) rule that can fire, among all the rules of the neuron. Once such rule has been found, we remove all the spikes consumed by the rule and we communicate (after a specified number of steps) the produced spikes to all adjacent neurons, provided that they are open.

In order to formalize this simulation we consider a deterministic Turing machine M with $m + 1$ tapes: m tapes are used to simulate the activity of each neuron,

while the remaining tape is used to store the description of Π. In the i-th tape, used to simulate neuron σ_i, we write the triplet $N_i = (n_i, o_i, t_i)$, where:

- n_i indicates the number of spikes contained into neuron σ_i;
- o_i stores the number of spikes produced by the last rule that fired, and that are ready to be sent to adjacent neurons (zero if the neuron is open, but no rule can be applied);
- t_i denotes the number of steps during which the neuron will remain closed (zero if the neuron is open).

With a little abuse of notation, in what follows we will refer to neuron σ_i by the triplet N_i (that contains the dynamical information about the state of σ_i) and by its set of rules R_i.

We simulate each step of computation of Π in two macro substeps: in the first substep we simulate the *firing phase* (i.e., the consumption of spikes) of each open neuron; then, in the second substep we simulate the *spiking phase* (i.e., the receipt of spikes) from adjacent neurons.

The simulation of a step of Π can be illustrated using the following algorithm, given in pseudocode:

```
SIMULATE-COMPUTATION-STEP(N, R)
// Remark: N = (N_1, N_2, ..., N_m), where N_i = (n_i, o_i, t_i)   ∀ i ∈ {1, ..., m}
//         R = (R_1, R_2, ..., R_m), where R_i is the set or rules of neuron σ_i

// Firing phase
for each neuron N_i
    do if t_i = 0                    // if the neuron is open then fire
       then (p; q; t) = SELECT-RULE-TO-APPLY(N_i, R_i)
            n_i ← n_i − p            // remove the spikes used to fire
            o_i ← q                  // prepare the buffer for emitting spikes
            t_i ← d                  // set closing time
       else t_i ← t_i − 1            // else decrease closing time

// Spiking phase
for each neuron N_i
    do if t_i = 0                    // if the neuron is open then spike
       then for each neuron N_j
            if (i, j) ∈ syn and t_j = 0      // if σ_j is connected to
                                             //    σ_i and is open
               then n_j ← n_j + o_i          // send the spikes
                    o_i ← 0                  // once spiked, clear the buffer


SELECT-RULE-TO-APPLY(N_i, R_i)
// Remark: N_i = (n_i, o_i, t_i)
//         R_i = {r_{i,1}, r_{i,2}, ..., r_{i,k}}, where r_{i,h} = E_{i,h}/a^{p_{i,h}} → a^{q_{i,h}}; d_{i,h}
```

```
//                                          for all h ∈ {1, 2, ..., k}
// Select the rule to be applied in neuron N_i
for each rule r_{i,h} ∈ R_i
    if n_i ∈ L(E_{i,h}) and n_i ≥ p_{i,h}        // if the rule is applicable
       then return (p_{i,h}; q_{i,h}; d_{i,h})      // set the parameters to be used
                                                    // in the simulation
return (0; 0; 0)          // if no rule can be applied, set to 0 all parameters
```

The time complexity of this algorithm can be determined as follows.

The first (firing) phase requires to check for each neuron if it is open and, in such a case, to select a rule and simulate it. Let Z_i denote the time required to select the rule to be applied in neuron σ_i, and let Z be the maximum of all Z_i. We will return later to this value, due to its importance in the time complexity of the overall simulation. Once the rule to be applied has been selected, we need to update the number of spikes in the neuron. Thus, we first subtract a certain number p of spikes (at most P) from n_i. The time required by this operation depends on the number of binary digits which are needed to represent p and n_i. In general, the largest among the two numbers will be the latter. In fact assume that, at the first step of computation, n_i is initialized with a certain number w of spikes; moreover, assume that the neuron does not consume any spike until the current step, and that all the other neurons send to σ_i the maximum possible amount of spikes (P) per computation step in all the previous steps. Then, after t steps of computation, the neuron σ_i will contain $w + tQm$ spikes. This means that the above subtraction requires a time which is $O(\log(w + tQm))$, where w, m and Q are polynomial with respect to the description size. The time required to prepare the buffer for the emission of spikes is $O(\log Q)$, while the time required to set the closure time is $O(\log D)$. Thus, the time required to simulate the firing of a single neuron is $O(Z + \log(w + tQm) + \log Q + \log D)$, and for m neurons is $O(m\,(Z + \log(w + tQm) + \log Q + \log D))$.

The second phase, the spiking, requires to check for each neuron σ_i if it is open and, in such a case, to check for any other neuron σ_j if it is connected to σ_i. If it is connected, then we need to add to n_j the number of spikes emitted by σ_i. In the worst case, each neuron is connected to all other neurons, and all neurons deliver Q spikes to every other neuron. This means that for each neuron σ_i (that is, m times) we have to check whether it is open; this operation requires $O(\log D)$ steps. If σ_i is open, then for every other neuron σ_j we have to:

- check if neurons σ_i and σ_j are connected: the time needed to execute this operation is proportional to m^2 (here we assume that the Turing machine is able to move only to the next or to the previous cell during a computation step);
- check if neuron σ_j is open: the time needed is $O(\log D)$;
- if both the previous conditions hold, then add the spikes emitted by σ_i to n_j. This sum involves numbers of at most $O(\log(w + tQm))$ bits.

Thus, a step of the spiking phase requires $O\big((m \log D) \cdot m \cdot (m^2 + \log D + \log(w + tQm))\big) = O\big((m^2 \log D)(m^2 + \log D + \log(w + tQm))\big)$ steps.

The total time required to simulate t steps of Π is thus t times the time needed to perform the two phases, that is, $t \cdot \big(O(m\,(Z + \log(w + tQm) + \log Q + \log D)) + O\big((m^2 \log D)(m^2 + \log D + \log(w + tQm))\big)\big)$.

To show that this time is polynomial with respect to the description size of the system Π, we need to explicit the time Z required to select which rule has to be applied in neuron σ_i. We stress the fact that, since the system Π is deterministic, at each computation step there is at most one rule in σ_i which can fire. In order to select such a rule, we need to check whether there are enough spikes in the neuron (and clearly this can be done in polynomial time), as well as to check if $n_i \in L(E_i)$. In general, this last operation cannot be done in polynomial time, as it will be proved in the next proposition. Nonetheless, in [9] it is shown that universality can be obtained by using SN P systems where the regular expressions associated with each rule are of very simple forms: a^i, with $i \leq 3$, or $a(aa)^+$. Considering such systems, it is easy to see that the time required to check if the contents of σ_i is in the regular set defined by the regular expression can be done in polynomial time, since in the former case it suffices to check if the number is equal to i (time proportional to $\log n_i$), whereas in the latter case it suffices to check the last bit of n_i.

Thus, the time required to select the rule to apply depends on the number of rules in each neuron. That is, $Z_i = O(|R_i|) = O(r)$, where $r = \max_{1 \leq i \leq m} |R_i|$ is the maximum number of rules which can be contained in a neuron.

As a consequence, the total time required to simulate t steps of Π is $t \cdot \big(O(m\,(r + \log(w + tQm) + \log Q + \log D)) + O\big((m^2 \log D)(m^2 + \log D + O(\log(w + tQm)))\big)\big)$.

We conclude this section by stressing that the above simulation could be performed in polynomial time because the regular expressions used in the rules are of a very restricted form. Indeed, there is a large amount of computational power hidden into the implicit mechanism that SN P systems use to decide whether a given rule can be applied or not, as proved in the following proposition. The difficulty of checking whether the contents of a neuron is into the regular set determined by the regular expression E of the rule is induced by the fact that we are dealing with unary languages. As told above, in these languages a string is uniquely determined by its length. Hence, a compact representation of the string is obtained by writing (in binary) its length, rather than by writing the string itself. This is an exponentially smaller representation, and consequently all the problems defined upon this representation become harder, as it happens for all "succint" representations (see [15], chapter 20). Concerning the succint version of the membership problem (is a given string into the language generated by E?) we can prove the following proposition.

Proposition 1. *Let Π be an SN P system having a single neuron, that contains the rule $E : a^c \to a; d$, where $c \geq 1$ and $d \geq 0$ are natural numbers, and E is any regular expression (for a unary language defined on the alphabet $\{a\}$). If E*

is described in a succint form, then deciding whether this rule can be applied is at least **NP***–complete.*

Proof. Let us show a polynomial time reduction from Subset Sum to this problem. Let $(V = \{v_1, v_2, \ldots, v_n\}, S)$ be an instance of Subset Sum. If we put $K = \max\{v_1, v_2, \ldots, v_n, S\}$, then the instance size is $\Theta(nK)$, as discussed in section 3.1. Consider the regular expression $E = E_1 \circ E_2 \circ \ldots \circ E_n$, where $E_i = (\lambda \cup \{a^{v_i}\})$, with λ the symbol that represents the empty word. Since we are dealing with unary languages in succint form, every string of $L(E)$ can be uniquely determined by its length, and thus we can write $L(E_i) = \{0, v_i\}$. $L(E)$ is just obtained by performing a language theoretic concatenation among the languages $L(E_i)$: $L(E) = L(E_1) \circ L(E_2) \circ \ldots \circ L(E_n)$. Clearly, the regular expression E can be represented using a string whose length is polynomial with respect to the size of the given instance of Subset Sum. It is immediately verified that $L(E)$ contains 2^n elements, bijectively associated with the subsets of V. Moreover, $a^S \in L(E)$ if and only if there exists a set $B \subseteq V$ such that $\sum_{b \in B} b = S$. Hence, checking whether a^S is in $L(E)$ or not is equivalent to solve the Subset Sum problem on the given instance (V, S).

5 Conclusions and Directions for Future Research

In this paper we have started to study the computational power of spiking neural P systems. In particular, by slightly extending the original definition given in [8] and [9] we have shown that by exploiting nondeterminism it is possible to solve **NP**–complete problems such as Subset Sum and 3-SAT.

Concerning deterministic systems, we have shown that the universal deterministic SN P systems defined in [8, 9] can be simulated by deterministic Turing machines with a polynomial slowdown. Surprisingly, this was possible only because the universal P systems described in [8, 9] use regular expressions of a very restricted form. In fact, it was shown that if we allow the use of general regular expressions then we can exploit the mechanism used by SN P systems to decide whether the contents of a neuron matches a regular expression to solve the **NP**–complete problem Subset Sum.

Further research is needed to fully understand the power of accepting SN P systems. In particular, by encoding their inputs as the distance between subsequent spikes we are limiting ourselves to use numbers expressed in unary notation. A more compact way to encode a k-bit natural number n would be to send a sequence of spikes during a prefixed sequence of k computation steps: the presence of a spike indicates a 1, its absence indicates a 0. It is not currently known whether in this way SN P systems can still solve numerical **NP**–complete problems such as Subset Sum, or whether a subsystem that converts integer numbers from binary to unary notation can be designed, as it was made in [10] for traditional P systems.

Acknowledgments

We gratefully thank Gheorghe Păun for introducing the authors to the stimulating subject of spiking neural P systems, and for asking us a "Milano theorem" (in the spirit of [22]) about their computational power, during the Fifth Brainstorming Week on Membrane Computing, held in Seville from January 29$^{\text{th}}$ to February 2$^{\text{nd}}$, 2007. Moreover, we are really indebted with him for suggesting us the standard nondeterministic SN P system that solves 3-SAT, exposed in section 3.

References

1. A. Alhazov, R. Freund, M. Oswald: Cell/symbol complexity of tissue P systems with symport/antiport rules. *Intern. J. Found. Computer Sci.*, 17, 1 (2006), 3–26.
2. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In: M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds., *Fourth Brainstorming Week on Membrane Computing*, Vol. I RGCN Report 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 169–194.
3. T.H. Cormen, C.H. Leiserson, R.L. Rivest: *Introduction to Algorithms*. MIT Press, Boston, 1990.
4. R. Freund, Gh. Păun, M.J. Pérez-Jiménez: Tissue-like P systems with channel states. *Theoretical Computer Science*, 330 (2004), 101–116.
5. M.R. Garey, D.S. Johnson: *Computers and Intractability. A Guide to the Theory on* **NP***–Completeness*. W. H. Freeman and Company, 1979.
6. W. Gerstner, W. Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.
7. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth: Normal forms for spiking neural P systems. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.
8. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
9. M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: traces and small universal systems. In C. Mao, T. Yokomori, eds., *DNA Computing,* 12$^{\text{th}}$ *International Meeting on DNA Computing, DNA12*, Seoul, Korea, June 5-9, 2006, Revised Selected Papers. LNCS 4287, Springer, 2006, 1–16.
10. A. Leporati, C. Zandron, M.A. Gutiérrez-Naranjo: P systems with input in binary form. *International Journal of Foundations of Computer Science*, 17, 1 (2006), 127–146.
11. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
12. W. Maass, C. Bishop (eds.). *Pulsed Neural Networks*, MIT Press, Cambridge (MA), 1999.
13. C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón: A new class of symbolic abstract neural nets: Tissue P systems. In *Proceedings of COCOON 2002*, Singapore, LNCS 2387, Springer-Verlag, Berlin, 290–299.
14. M. Oswald: Independent agents in a globalized world modelled by tissue P systems. *Conf. Artificial Life and Robotics*, 2006.

15. C.H. Papadimitriou: *Computational Complexity*, Addison-Wesley, 1994.
16. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143. See also Turku Centre for Computer Science — TUCS Report No. 208, 1998. Available at: *http://www.tucs.fi/Publications/techreports/TR208.php*
17. Gh. Păun: Computing with membranes. An Introduction. *Bulletin of the EATCS*, 67 (2/1999), 139–152.
18. Gh. Păun: *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, 2002.
19. Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen*, 60 (2002), 635–660.
20. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted for publication.
21. G. Păun, G. Rozenberg: A guide to membrane computing. *Theoretical Computer Science*, 287, 1 (2002), 73–100.
22. C. Zandron, C. Ferretti, G. Mauri: Solving **NP**–complete problems using P systems with active membranes. In I. Antoniou, C.S. Calude, M.J. Dinneen, eds., *Unconventional Models of Computation*, Springer-Verlag, London, 2000, 289–301.
23. The P systems Web page: http://psystems.disco.unimib.it/

Some Mathematical Methods and Tools for an Analysis of Harmony-Seeking Computations

Adam Obtułowicz

Institute of Mathematics, Polish Academy of Sciences
Śniadeckich 8, P.O.B. 21, 00-956 Warsaw, Poland
A.Obtulowicz@impan.gov.pl

Summary. A general review of some topic concepts and methods of membrane computing [15], [19] which can be useful in an analysis of harmony-seeking computations [1] is presented. Then an application of a certain particular method of membrane computing in a discussion of mobility of some systems considered in city planning [2] is described in some details. A conclusion of the discussion states that systems of hierarchical organization in a form of a tree are mobile by means of massively parallel (local) moves of the parts of the systems, where the moves are related to the process capabilities of moves of ambients in [7].

1 Introduction

An idea of a harmony-seeking computation was introduced and discussed in [1] aiming, among others, to improve design and planning in architecture in order to achieve that internal coherence (harmony) between the designed and planned objects of various scales (from the rooms in buildings, through buildings themselves, the districts of cities, and to cities themselves) which natural living system possesses. Therefore the discussion of harmony-seeking computations in [1] expands far beyond the methods of design and planning in architecture and concerns also better understanding of the phenomena of life with a regard to a geometric adaptation.

The paper [1] contains general postulates for mathematical modeling of harmony-seeking computations.

The aim of the present paper is to propose and review some known mathematical methods and tools which may serve for modeling harmony-seeking computations according to those postulates.

The methods focus on modeling those evolution processes of natural (living) systems which may realize massively parallel computations themselves or inspire a planning of devices realizing these computations, where an aspect of geometric

adaptation is respected (on topological level) by considering certain possible transformations of hierarchical organization of systems during their evolution processes. A hierarchical organization is meant here as determined by a nesting relation of the less complex parts of a system in the more complex parts of a system.

The proposed and reviewed in the paper (Section 2) mathematical methods and tools are mainly those already applied in membrane computing, a branch of natural computing initiated by Gh. Păun and described in [19] (see also the P systems page [15]), where the underlying structure of a system (called a membrane system) with respect to nesting of subsystems or its parts is a tree and this underlying structure may be transformed during an evolution process.

In Section 3 and Appendix we present in some details a concrete application of membrane computing methods for an analysis of mobility (with respect to massively parallel moves) of certain systems considered in city planning and discussed in [2]. Section 3 together with Appendix are self-contained.

The author thanks Krzysztof Wodiczko for long discussions about architecture and mathematics.

2 Membrane Systems and Their Evolution Rules; A General Review

In [1] a harmony-seeking computation is identified with its underlying process whose steps are wholeness-extending-transformations (briefly W-E-transformations), where each W-E-transformation operates on one wholeness to produce another wholeness which is illustrated as follows:

$$W_1 -^{WE_1}\!\!\rightarrow W_2 -^{WE_2}\!\!\rightarrow W_3 -^{WE_3}\!\!\rightarrow W_4 -^{WE_4}\!\!\rightarrow \dots .$$

The character of W-E-transformations is established in [1] by five postulates A1–A5 about the structure of wholeness and three postulates B1–B3 about the definition of W-E-transformations themselves.

In Postulate A5 each wholeness is identified (defined as) with a system of configurations, where according to Postulate A2 the subconfigurations may be spatially nested, or overlapping, or disjoint.

One can see that membrane systems, the basic tools of membrane computing, meant as finite trees with nodes labeled by multisets are appropriate candidates to model (the structure of configurations identified with) a wholeness because the trees (and their Venn diagram presentation, cf. [19]) well describe the spatial nesting. Moreover, the description of evolution processes of membrane systems by using evolution rules in membrane computing also well suits to model the harmony-seeking computations. Namely, the evolution rules of membrane systems are similar to production rules generating languages and they can be simultaneously applied to membrane systems in a similar way like production rules for L-systems can be simultaneously applied to the processed expressions. We point out here

that a simulation of harmony-seeking process of tree growth by using a context-free L-system is discussed in [1].

The known applications of membrane systems and their evolution rules for modeling processes in system biology presented among others in the recent papers contained in (Pre-)Proceedings of Workshops and Brainstorming Weeks on Membrane Computing (cf. [13], [14], [11], see the papers by D. Bezzossi, N. Busi and C. Zandron, L. Cardelli and Gh. Păun, V. Manca) show that it is worth to apply membrane computing methods and tools to model harmony-seeking computations.

The most remarkable are those applications which concern fractals generation by P systems presented in [12], because fractals represent geometry of dynamic systems with a regard to similarities in various scales which is an important aspect of fractals applications in architecture, see [20].

3 Semilattices of Subsets, Trees of Subsets, hereditarily finite sets, and Their Mobility

The paper [2] contains a discussion of a thesis that a city structure of a form of a semilattice of subsets is better (topologically) adapted or more fit to live in than a structure of a form of a tree of subsets.

In this section we introduce a representation of those semilattices and trees by certain hereditarily finite sets and then we show that this representation makes possible to transfer from [18] some results concerning mobile ambients and mobile membranes into the area of trees and semilattices of subsets and then into the realm of city planning.

We quote from [2] the definitions of semilattices and trees of subsets.

The semilattice axiom goes like this: *A collection of sets forms a semilattice if and only if, when two overlapping sets belong to the collection, the set of elements common to both also belongs to the collection.*

The tree axiom states: *A collection of sets forms a tree if and only if, for any two sets that belong to the collection, either one is wholly contained in the other, or else they are wholly disjoint.*

We use the following notion which is a generalization of the above defined concepts.

We define a [*finite*] *nesting structure* to be an ordered pair $\mathfrak{N} = (U_{\mathfrak{N}}, \mathcal{N}_{\mathfrak{N}})$ such that $U_{\mathfrak{N}}$ is a [finite] set, called the *underlying set of* $\mathfrak{N}$, and $\mathcal{N}_{\mathfrak{N}}$ is a collection of nonempty subsets of $U_{\mathfrak{N}}$ with $U_{\mathfrak{N}}$ belonging to $\mathcal{N}_{\mathfrak{N}}$. The elements of $\mathcal{N}_{\mathfrak{N}}$ are called the *parts in* $\mathfrak{N}$.

For two parts $\boldsymbol{n}, \boldsymbol{n}'$ in a nesting structure $\mathfrak{N}$ we define that $\boldsymbol{n}$ is *an immediate part of* $\boldsymbol{n}'$ *in* $\mathfrak{N}$ (and write $\boldsymbol{n} \prec \boldsymbol{n}'$) if $\boldsymbol{n} \subsetneq \boldsymbol{n}'$ and for every part $\boldsymbol{m}$ in $\mathfrak{N}$ if $\boldsymbol{n} \subseteq \boldsymbol{m} \subseteq \boldsymbol{n}'$, then $\boldsymbol{m} = \boldsymbol{n}$ or $\boldsymbol{m} = \boldsymbol{n}'$.

We recall now the notion of a hereditarily finite set used in [18].

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

$$\begin{aligned} \mathrm{HF}_0 &= \varnothing, \\ \mathrm{HF}_{i+1} &= \text{the set of nonempty finite subsets of } L \cup \mathrm{HF}_i. \end{aligned}$$

The elements of the union $\mathrm{HF} = \bigcup\{\mathrm{HF}_i \,|\, i \geq 0\} \cup \{\varnothing\}$ are called *hereditarily finite sets over* L or *hereditarily finite sets with urelements* in L, or simply *hereditarily finite sets* if there is no risk of confusion.

For $x \in \mathrm{HF}$ we define its weak transitive closure $\mathrm{WTC}(x)$ and support $\mathrm{supp}(x)$ by

$$\begin{aligned} \mathrm{WTC}(x) &= \bigcup\big\{\mathrm{WTC}(y) \,|\, y \in x \text{ and } y \in \mathrm{HF}\big\} \cup \{x\} \\ \mathrm{supp}(x) &= (x \cap L) \cup \bigcup\{\mathrm{supp}(y) \,|\, y \in x \text{ and } y \in \mathrm{HF}\}, \end{aligned}$$

and the *depth of* x is defined to be the smallest natural number i for which $x \in \mathrm{HF}_i$.

The notion of a hereditarily finite set is applied in [10] to give a general characterization of physical computing devices. The characterization is improved in [4], [21], [22], and examples are given in [9]. Membrane computing applications of hereditarily finite sets are discussed in [16], [17], [18].

For a finite nesting structure $\mathfrak{N} = (U_{\mathfrak{N}}, \mathcal{N}_{\mathfrak{N}})$ we define *its hereditarily finite set* $\mathrm{hfs}(\mathfrak{N})$ by

$$\mathrm{hfs}(\mathfrak{N}) = \big(U_{\mathfrak{N}} - \bigcup\{\boldsymbol{n} \in \mathcal{N}_{\mathfrak{N}} \,|\, \boldsymbol{n} \prec U_{\mathfrak{N}}\}\big) \cup \{\mathrm{hfs}(\mathfrak{N}(\boldsymbol{n})) \,|\, \boldsymbol{n} \in \mathcal{N}_{\mathfrak{N}} \text{ and } \boldsymbol{n} \prec U_{\mathfrak{N}}\},$$

where for a part $\boldsymbol{n} \in \mathcal{N}_{\mathfrak{N}}$ we write $\mathfrak{N}(\boldsymbol{n})$ to denote a nesting structure whose underlying set $U_{\mathfrak{N}(\boldsymbol{n})}$ is $\boldsymbol{n}$ itself and the set $\mathcal{N}_{\mathfrak{N}(\boldsymbol{n})}$ of parts in $\mathfrak{N}(\boldsymbol{n})$ is the set $\{\boldsymbol{n}' \in \mathcal{N}_{\mathfrak{N}} \,|\, \boldsymbol{n}' \subseteq \boldsymbol{n}\}$.

For a hereditarily finite set x we define *its nesting structure* $\mathfrak{N}^x$ by

$$U_{\mathfrak{N}^x} = \mathrm{supp}(x), \quad \mathcal{N}_{\mathfrak{N}^x} = \{\mathrm{supp}(y) \,|\, y \in \mathrm{WTC}(x)\}.$$

A characterization of hereditarily finite sets of finite nesting structures is formulated in the following theorem which one can treat as a representation theorem of nesting structures by hereditarily finite sets.

Theorem 1. *For every finite nesting structure* $\mathfrak{N}$*, if* x *is the hereditarily finite set of* $\mathfrak{N}$*, i.e.* $x = \mathrm{hfs}(\mathfrak{N})$*, then the following conditions hold*:

0) $\mathfrak{N}^x = \mathfrak{N}$,
1) $\mathrm{supp}(y) = \mathrm{supp}(y')$ *implies* $y = y'$ *for all* y, y' *with* $\{y, y'\} \subseteq \mathrm{WTC}(x)$,
2) $y \in y'$ *if and only if* $\mathrm{supp}(y) \prec \mathrm{supp}(y')$ *for all* y, y' *with* $\{y, y'\} \subseteq \mathrm{WTC}(x)$,
3) $y \cap \bigcup\{\mathrm{supp}(y') \,|\, y' \in y\} = \varnothing$ *for every* $y \in \mathrm{WTC}(x)$.

For every hereditarily finite set x*, if it satisfies the conditions* 1)–3)*, then* $\mathrm{hfs}(\mathfrak{N}^x) = x$.

Proof. One proves by induction on the number of elements of $\mathcal{N}_{\mathfrak{N}}$ that $x = \mathrm{hfs}(\mathfrak{N})$ implies that the conditions 0)–3) hold for x. One proves by induction on the depth of x that if the conditions 1)–3) hold for x, then $\mathrm{hfs}(\mathfrak{N}^x) = x$.

Corollary 1. *For a finite nesting structure* $\mathfrak{N}$ *the set* $\mathrm{WTC}(\mathrm{hfs}(\mathfrak{N}))$ *ordered by the membership relation* $\in$ *forms a structure which is isomorphic to that structure which is given by the set* $\mathcal{N}_{\mathfrak{N}}$ *of parts in* $\mathfrak{N}$ *ordered by* $\prec$.

Proof. The corollary is a consequence of Theorem 1. By 1) a mapping from $\mathrm{WTC}(\mathrm{hfs}(\mathfrak{N}))$ into $\mathcal{N}_{\mathfrak{N}}$ given by $y \mapsto \mathrm{supp}(y)$ is a bijection which preserves the ordering by 2), where 1) and 2) are the conditions from Theorem 1 which hold for $x = \mathrm{hfs}(\mathfrak{N})$.

Example 1. The set x of the form

$$\Big\{\Big\{\Big\{1,\{2,\{3\}\}\Big\},\Big\{\{2,\{3\}\},\{\{3\},\{4\}\}\Big\},\Big\{\{\{3\},\{4\}\},\{5,\{4\}\}\Big\}\Big\},\Big\{6,\Big\{\{\{3\},\{4\}\},\{5,\{4\}\}\Big\}\Big\}\Big\}$$

is a hereditarily finite set such that $\mathrm{hfs}(\mathfrak{N}^x) = x$, where $\mathfrak{N}^x$ is a semilattice illustrated in Fig. 0.

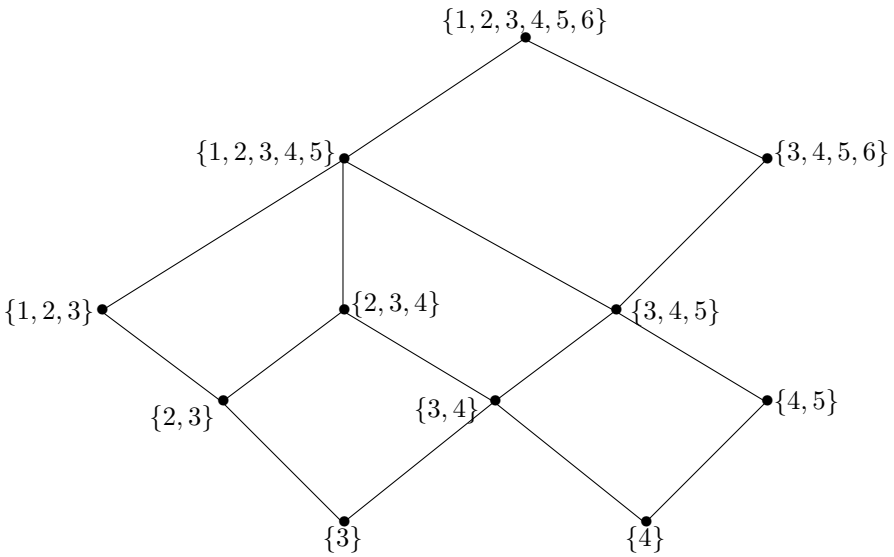


Fig. 0.

The representation of finite nesting structures by hereditarily finite sets described in Theorem 1 and Corollary 1 provides that already defined constructions and proved properties of hereditarily finite sets x satisfying $\mathrm{hfs}(\mathfrak{N}^x) = x$ can be transferred or interpreted in the class of finite nesting structures. In particular, the basic concepts and constructions describing mobile systems[1] modeled by hereditarily finite sets, see [18] and Appendix in the present paper, can be transferred to the class of finite nesting structures via the representation. The following theorem is a starting point of this transfer.

Theorem 2. *Let x be a hereditarily finite set such that* $\mathrm{hfs}(\mathfrak{N}^x) = x$. *Then the following conditions hold*:

$\mathbb{C}_1$) *if $y \in x$ with $y \in \mathrm{HF}$, then for $u_1 = (x - \{y\}) \cup y$ the condition $\mathcal{N}_{\mathfrak{N}^{u_1}} = \mathcal{N}_{\mathfrak{N}^x} - \{\mathrm{supp}(y)\}$ holds,*

$\mathbb{C}_2$) *if $\{y, z\} \subseteq x \cap \mathrm{HF}$ with $y \neq z$, then for $u_2 = (x - \{y, z\}) \cup \{z \cup \{y\}\}$ the condition $\mathcal{N}_{\mathfrak{N}^{u_2}} = (\mathcal{N}_{\mathfrak{N}^x} - \{\mathrm{supp}(y)\}) \cup \{\mathrm{supp}(y) \cup \mathrm{supp}(z)\}$ holds,*

$\mathbb{C}_3$) *if $z \in y \in x$ with $z \in \mathrm{HF}$, then for $u_3 = (x - \{y\}) \cup \{y - \{z\}, z\}$ the condition $\mathcal{N}_{\mathfrak{N}^{u_3}} = (\mathcal{N}_{\mathfrak{N}^x} - \{\mathrm{supp}(y)\}) \cup \{\mathrm{supp}(y - \{z\})\}$ holds.*

Moreover, if $\mathfrak{N}^x$ is such that $\mathcal{N}_{\mathfrak{N}^x}$ is a tree, then for every $i \in \{1, 2, 3\}$ the set $\mathcal{N}_{\mathfrak{N}^{u_i}}$ is also a tree and $\mathrm{hfs}(\mathfrak{N}^{u_i}) = u_i$.

Proof. We prove the theorem by induction on the depth of x.

We interpret Theorem 2 in the following way. The conditions $\mathbb{C}_1$), $\mathbb{C}_2$), $\mathbb{C}_3$) correspond to the following process capabilities discussed in [7]:

- condition $\mathbb{C}_1$) corresponds to the capability "can open an ambient",
- condition $\mathbb{C}_2$) corresponds to the capability "can enter an ambient",
- condition $\mathbb{C}_3$) corresponds to the capability "can exit out an ambient".

The above capabilities are capabilities of some "spatial" moves of parts of systems with respect to hierarchical organization of systems determined by nesting relation of parts.

A mathematical description of the mentioned capabilities for systems modeled by hereditarily finite sets x (with $\mathrm{WTC}(x)$ meant as a collection of parts of x) is contained in conditions $\mathbb{C}_1$), $\mathbb{C}_2$), $\mathbb{C}_3$), where for every $i \in \{1, 2, 3\}$ the conditions written between "if" and "then" in $\mathbb{C}_i$) are (pre)conditions which provide a realization of a move and the equation defining u_i written after "then" in $\mathbb{C}_i$) is a (post)condition describing the result u_i of the move.

For a hereditarily finite set x modeling a system with parts represented by elements of $\mathrm{WTC}(x)$ the capabilities of moves can be applied (or referred) to the elements of $\mathrm{WTC}(x)$ and these applications are called *local moves in x*. The local moves are described in terms of Gh. Păun's evolution rules and their applications in [18], see also Appendix of the present paper, where a local action is a mathematical description of a local move.

[1] Related to mobile ambient systems in [7].

A collisionless set of simultaneous local moves in a given x (more than one local move in a unit of time) is described in [18] and Appendix as a proper set of local actions over x.

An inductive formula for assembly of a whole system from the results of local moves belonging to a collisionless set of simultaneous local moves in a hereditarily finite set x is given in [18], see also the inductive definition of $\mathrm{Ap}(\mathcal{A}, x)$ in Appendix, where $\mathrm{Ap}(\mathcal{A}, x)$ is the result of assembly for a proper set $\mathcal{A}$ of local actions over x. Theorem 3 in Appendix is the final and concluding step of the discussed transfer of the basic concepts and constructions describing mobile systems modeled by hereditarily finite sets into the area of nesting structures.

Remark. For x given in Example,

$$y = \Big\{\Big\{1, \{2, \{3\}\}\Big\}, \Big\{\{2, \{3\}\}, \{\{3\}, \{4\}\}\Big\}, \Big\{\{\{3\}, \{4\}\}, \{5, \{4\}\}\Big\}\Big\},$$

and

$$z = \Big\{\{2, \{3\}\}, \{\{3\}, \{4\}\}\Big\}$$

we have that $z \in y \in x$ which means that preconditions in $\mathbb{C}_3$) hold for these x, y, z. Then for $u_3 = (x - \{y\}) \cup \{y - \{z\}, z\}$ we have that $\mathfrak{N}^{u_3} = \mathfrak{N}^x$ (because $\mathrm{supp}(y - \{z\}) = \mathrm{supp}(y)$ in this case) which means that capability "can exit out an ambient" does not lead to any real move leaving $\mathfrak{N}^x$ unchanged.

Conclusion

By virtue of Theorem 2 and Theorem 3 in Appendix every finite nesting structure $\mathfrak{N}$ with $\mathcal{N}_{\mathfrak{N}}$ being a tree is mobile with respect to simultaneous (massively parallel) local moves determined by process capabilities "can open an ambient", "can enter an ambient", and "can exit out an ambient". The case discussed in Remark shows that mobility of some nesting structures $\mathfrak{N}$ with $\mathcal{N}_{\mathfrak{N}}$ being a semilattice is problematic.

Appendix

We consider those evolutive transformations of hereditarily finite sets into hereditarily finite sets which are determined by evolution rules written in Păun's manner as the parenthesis expressions, cf. [19]:

R_1) [ ] $\rightarrow$ (*dissolution rule*),
R_2) [ ][ ] $\rightarrow$ [[ ]] (*in-rule*),
R_3) [[ ]] $\rightarrow$ [ ][ ] (*out-rule*),

The single applications from the top of the above rules to hereditarily finite sets are described in the following way:

- if $y \in x \cap \mathrm{HF}$, then the dissolution rule $[\,] \to$ *can be applied to* x and the result of its application is a new hereditarily finite set of the form

$$(x - \{y\}) \cup y,$$

- if $\{y, z\} \subseteq x \cap \mathrm{HF}$, and $z \neq y$, then the in-rule $[\,][\,] \to [[\,]]$ *can be applied to* x and the result of its application is a new hereditarily finite set of the form

$$(x - \{y, z\}) \cup \{z \cup \{y\}\},$$

- if $z \in y \in x \in \mathrm{HF}$, $z \in \mathrm{HF}$, and $y - \{z\} \neq \varnothing$, then the out-rule $[[\,]] \to [\,][\,]$ *can be applied to* x and the result of its application is a new hereditarily finite set of the form

$$(x - \{y\}) \cup (\{y - \{z\}, z\} - \{\varnothing\}).$$

The above described single applications of evolution rules R_1), R_2), R_3) from the top determine evolutive transformations of hereditarily finite sets into new hereditarily finite sets from the top. One sees that these rules are related to process capabilities "can open an ambient", "can enter an ambient", "can exit out an ambient", introduced in [6]. The evolution rules may describe forces in patterns, cf. [3]. We describe by using $\cup$, $-$, and $\{?\}$ a more complicated case of evolutive transformations of hereditarily finite sets, where these transformations are determined by simultaneous applications of different rules to many different elements of $\mathrm{WTC}(x)$ for a hereditarily finite set x to be transformed.

The evolutive transformations of hereditarily finite sets considered above can be "transferred" to nesting structures by using the construction of $\mathfrak{N}^x$ (see Theorem 1 and Corollary 1) to define evolutive transformations of nesting structures themselves.

We restrict our considerations to *tree-like hereditarily finite sets* which are defined to be such that $\mathrm{hfs}(\mathfrak{N}^x) = x$ and $\mathfrak{N}^x$ is a tree.

Let x be a tree-like hereditarily finite set. By a *local action over* x we mean an ordered pair $\mathfrak{a} = (P^{\mathfrak{a}}, R^{\mathfrak{a}})$, where $P^{\mathfrak{a}}$ is a bijection from $\mathrm{dom}(\mathfrak{a})$ into $\mathrm{scope}(\mathfrak{a})$ with $\mathrm{scope}(\mathfrak{a}) \subset \mathrm{WTC}(x)$ and $R^{\mathfrak{a}}$ is an evolution rule such that

a_1) if $R^{\mathfrak{a}}$ is a dissolution rule $[\,] \to$, then $\mathrm{dom}(\mathfrak{a}) = \{0, 1\}$ and $P^{\mathfrak{a}}(1) \in P^{\mathfrak{a}}(0)$,
a_2) if $R^{\mathfrak{a}}$ is an in-rule $[\,][\,] \to [[\,]]$, then $\mathrm{dom}(\mathfrak{a}) = \{0, 1, 2\}$ and $\{P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(2)\} \subset P^{\mathfrak{a}}(0)$,
a_3) if $R^{\mathfrak{a}}$ is an out-rule $[[\,]] \to [\,][\,]$, then $\mathrm{dom}(\mathfrak{a}) = \{0, 1, 2\}$ and $P^{\mathfrak{a}}(2) \in P^{\mathfrak{a}}(1) \in P^{\mathfrak{a}}(0)$.

For a local action $\mathfrak{a}$ over x the bijection $P^{\mathfrak{a}}$ is meant as a *place of application* of the rule $R^{\mathfrak{a}}$, where it will be seen later than one can interpret $\mathrm{scope}(\mathfrak{a})$ as the scope of the local transformation of x according to the rule $R^{\mathfrak{a}}$.

Let $\mathcal{A}$ be a set of local actions over x. For a set $y \in \mathrm{WTC}(x)$ and a set $z \subseteq y$ we write $\mathcal{A} \restriction (y - z)$ to denote the set of local actions $\mathfrak{a}$ over $y - z$ such that $\mathfrak{a} \in \mathcal{A}$ or $P^{\mathfrak{a}}(0) = y - z$ with $\mathfrak{a}^* = (P^{\mathfrak{a}^*}, R^{\mathfrak{a}}) \in \mathcal{A}$ for $P^{\mathfrak{a}^*} : \mathrm{dom}(\mathfrak{a}) \to (\mathrm{scope}(\mathfrak{a}) - \{y - z\}) \cup \{y\}$ with $P^{\mathfrak{a}^*}(i) = P^{\mathfrak{a}}(i)$ for all $i \in \mathrm{dom}(\mathfrak{a}) - \{0\}$. If $z = \varnothing$,

then $\mathcal{A} \restriction (y - z) = \mathcal{A} \restriction y$ is simply the set of those local actions over y which belong to $\mathcal{A}$. If $z = y$, then $\mathcal{A} \restriction (y - z) = \mathcal{A} \restriction \varnothing = \varnothing$.

For a set $\mathcal{A}$ of local actions over x we adopt the following notation

$$\mathcal{A}_\alpha = \{\mathfrak{a} \in \mathcal{A} \,|\, R^{\mathfrak{a}} \text{ is an } \alpha\text{-rule}\} \quad \text{for} \quad \alpha \in \{\text{in}, \text{out}\},$$
$$\mathcal{A}_{\text{diss}} = \{\mathfrak{a} \in \mathcal{A} \,|\, R^{\mathfrak{a}} \text{ is a dissolution rule}\}.$$

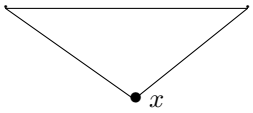
We define now a property of sets $\mathcal{A}$ of local actions over tree-like hereditarily finite sets x such that if $\mathcal{A}$ has this property, then one can construct the result of transformation of x with respect to $\mathcal{A}$ in a consistent (unambiguous) way, where x is transformed according to simultaneous application of the rules $R^{\mathfrak{a}}$ in places $P^{\mathfrak{a}}$, respectively for all $a \in \mathcal{A}$.

A set $\mathcal{A}$ of local actions over x is called a *proper set of local actions over* x if for all local actions $\mathfrak{a}, \mathfrak{a}'$ in $\mathcal{A}$ if $\mathfrak{a} \neq \mathfrak{a}'$, then $\text{scope}(\mathfrak{a}) \cap \text{scope}(\mathfrak{a}') = \varnothing$ or the disjunction of the following conditions holds:

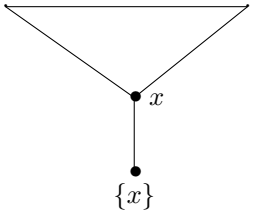
(C_1) $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(0)$ and $(\text{scope}(\mathfrak{a}) - \{P^{\mathfrak{a}}(0)\}) \cap (\text{scope}(\mathfrak{a}') - \{P^{\mathfrak{a}'}(0)\}) = \varnothing$,
(C_2) if $\{\mathfrak{a}, \mathfrak{a}'\} \subseteq \mathcal{A}_{\text{diss}}$, then $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(1)$,
(C_3) if $\{\mathfrak{a}, \mathfrak{a}'\} \subseteq \mathcal{A}_{\text{in}}$, then $P^{\mathfrak{a}}(1) = P^{\mathfrak{a}'}(1)$ or $P^{\mathfrak{a}'}(0) \in \{P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(2)\}$,
(C_4) if $\{\mathfrak{a}, \mathfrak{a}'\} \subseteq \mathcal{A}_{\text{out}}$, then $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(2)$
or $\{P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(2)\} \cap \{P^{\mathfrak{a}'}(0), P^{\mathfrak{a}'}(1)\} = \{P^{\mathfrak{a}}(1)\}$,
(C_5) if $\mathfrak{a} \in \mathcal{A}_{\text{diss}}$ and $\mathfrak{a}' \in \mathcal{A}_{\text{in}}$, then $P^{\mathfrak{a}}(1) = P^{\mathfrak{a}'}(0)$
or $P^{\mathfrak{a}}(0) \in \{P^{\mathfrak{a}'}(1), P^{\mathfrak{a}'}(2)\}$,
(C_6) if $\mathfrak{a} \in \mathcal{A}_{\text{diss}}$ and $\mathfrak{a}' \in \mathcal{A}_{\text{out}}$, then $P^{\mathfrak{a}}(1) = P^{\mathfrak{a}'}(0)$ or $\{P^{\mathfrak{a}}(0), P^{\mathfrak{a}}(1)\} \cap \{P^{\mathfrak{a}'}(1), P^{\mathfrak{a}'}(2)\} = \{P^{\mathfrak{a}}(0)\}$,
(C_7) if $\mathfrak{a} \in \mathcal{A}_{\text{in}}$ and $\mathfrak{a}' \in \mathcal{A}_{\text{out}}$, then $P^{\mathfrak{a}}(1) = P^{\mathfrak{a}'}(1)$ or $P^{\mathfrak{a}'}(0) \in \{P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(2)\}$
or $\text{scope}(\mathfrak{a}) \cap \{P^{\mathfrak{a}'}(1), P^{\mathfrak{a}'}(2)\} = \{P^{\mathfrak{a}}(0)\}$.

We adopt the following conventions to explain and illustrate the notion of a proper set of local actions.

For a tree-like non-empty hereditarily finite set x whose content is not specified (or is not important for considerations) we illustrate x by a drawing given by a triangle below whose bottom vertex is labeled by x.

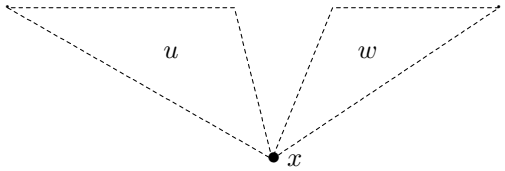


For a tree-like non-empty hereditarily finite set x whose content is not specified we illustrate one-element set $\{x\}$ by a drawing given by a triangle with an arrow glued to the bottom vertex of the triangle as below



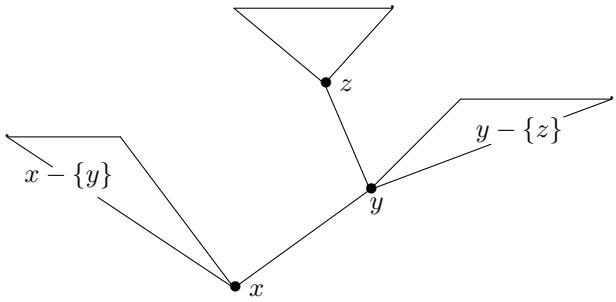
where the bottom vertex of the drawing is that vertex which is labeled by $\{x\}$.

If a tree-like hereditarily finite set x is such that $x = u \cup w$ for hereditarily finite sets u, w with $(\mathrm{HF} \cap u) \cap (\mathrm{HF} \cap w) = \varnothing$ such that there are given the drawings used for illustrations of u and w, respectively, then we illustrate x by a drawing below



where the meta-triangles labeled by u and w contain the drawing used to illustrate u and the drawing used to illustrate w, respectively. In the above drawing which illustrates $x = u \cup w$ the bottom vertex labeled by x is the result of gluing of the bottom vertex of the drawing used to illustrate u and the bottom vertex of the drawing used to illustrate w. Here the intersection of the set of vertices of the drawing for u and the set of vertices of the drawing for w is the one-element set containing the result of gluing described above, which is the vertex labeled by x.

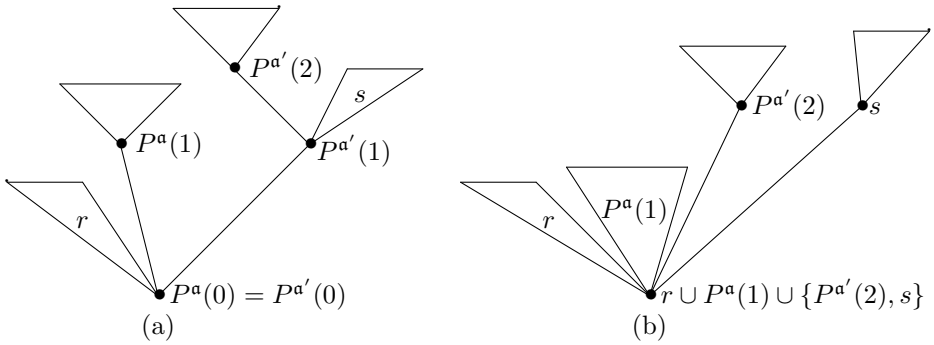
Thus for tree-like hereditarily finite sets x, y, z such that $z \in y \in x$ one can illustrate x by the drawing



where the contents of $x - \{y\}$, $y - \{z\}$, and z are not specified.

We explain and illustrate the conditions (C_1)–(C_7).

Ad (C_1). For two different local actions $\mathfrak{a} \in \mathcal{A}_{\mathrm{diss}}$ and $\mathfrak{a}' \in \mathcal{A}_{\mathrm{out}}$ satisfying (C_1) the places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ are illustrated in Fig. 1(a). The result of simultaneous application of the rules $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$, respectively, is illustrated in Fig. 1(b), where $P^{\mathfrak{a}}(1)$ is "dissolved" in $P^{\mathfrak{a}}(0)$ and $P^{\mathfrak{a}'}(2)$ is "sent out" of $P^{\mathfrak{a}'}(1)$

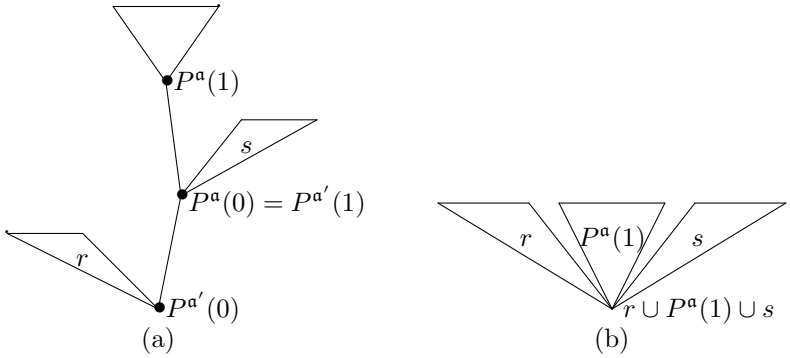


where $r = P^{\mathfrak{a}}(0) - \{P^{\mathfrak{a}}(1), P^{\mathfrak{a}'}(1)\}$ and $s = P^{\mathfrak{a}'}(1) - \{P^{\mathfrak{a}'}(2)\}$

Fig. 1.

into $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(0)$. The remaining cases of $\mathfrak{a}$ and $\mathfrak{a}'$ satisfying (C_1) are explained and illustrated in a similar way.

Ad (C_2). For two different local actions $\mathfrak{a}, \mathfrak{a}'$ belonging to $\mathcal{A}_{\text{diss}}$ with $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(1)$ the places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ are illustrated in Fig. 2(a). The result of simultaneous application of the rules $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$, respectively, is illustrated in Fig. 2(b), where both $P^{\mathfrak{a}}(1)$ and $P^{\mathfrak{a}'}(1) - \{P^{\mathfrak{a}}(1)\}$ are "dissolved" simultaneously in $P^{\mathfrak{a}'}(0)$.



where $r = P^{\mathfrak{a}'}(0) - \{P^{\mathfrak{a}}(0)\}$ and $s = P^{\mathfrak{a}}(0) - \{P^{\mathfrak{a}}(1)\}$

Fig. 2.

Ad (C_3). For two different local actions $\mathfrak{a}, \mathfrak{a}'$ belonging to $\mathcal{A}_{\text{in}}$ with $P^{\mathfrak{a}}(1) = P^{\mathfrak{a}'}(1)$ the places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ are illustrated in Fig. 3(a). The result of simultaneous application of the rules $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in these places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$, respectively, is illustrated in Fig. 3(b), where both $P^{\mathfrak{a}}(2)$ and $P^{\mathfrak{a}'}(2)$ are "sent into" $P^{\mathfrak{a}}(1) = P^{\mathfrak{a}'}(1)$ simultaneously. We point out that for all two different local actions $\mathfrak{a}$ and $\mathfrak{a}'$

with $\mathrm{scope}(\mathfrak{a}) \cap \mathrm{scope}(\mathfrak{a}') \neq \varnothing$ the condition (C_3) implies $P^{\mathfrak{a}}(1) \neq P^{\mathfrak{a}'}(2)$, which excludes the case such that simultaneous application of $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ is ambiguous. The remaining cases of $\mathfrak{a}$ and $\mathfrak{a}'$ satisfying (C_3) are explained and illustrated in a similar way.

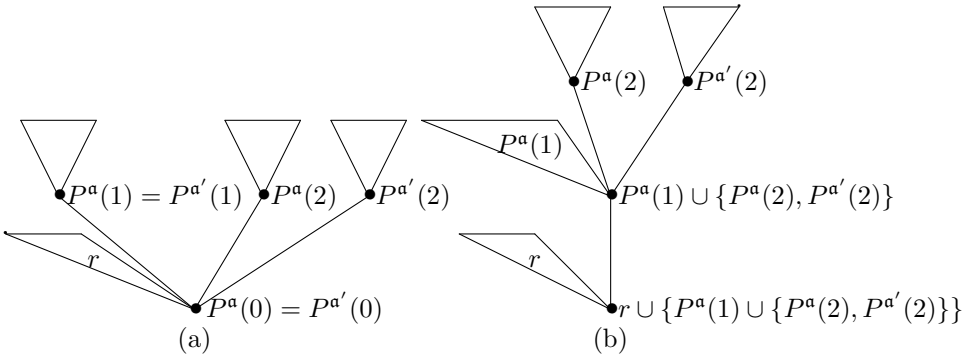


Fig. 3.

Ad (C_4). For two different local actions $\mathfrak{a}, \mathfrak{a}'$ belonging to $\mathcal{A}_{\mathrm{out}}$ we explain the case of $\{P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(2)\} \cap \{P^{\mathfrak{a}'}(0), P^{\mathfrak{a}'}(1)\} = \{P^{\mathfrak{a}}(1)\}$ which is equivalent to the disjunction of the following two conditions:

i) $P^{\mathfrak{a}'}(0) = P^{\mathfrak{a}}(1)$ and $P^{\mathfrak{a}'}(1) \neq P^{\mathfrak{a}}(2)$,
ii) $P^{\mathfrak{a}}(1) = P^{\mathfrak{a}'}(1)$.

The places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ for the case i) are illustrated in Fig. 4(a). The result of simultaneous application of $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in these places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$, respectively, is illustrated in Fig. 4(b), where $P^{\mathfrak{a}}(2)$ and $P^{\mathfrak{a}'}(2)$ are simultaneously "sent out" of $P^{\mathfrak{a}}(1)$ into $P^{\mathfrak{a}}(0)$ and of $P^{\mathfrak{a}'}(2)$ into $P^{\mathfrak{a}'}(0) = P^{\mathfrak{a}}(1)$, respectively. The condition $P^{\mathfrak{a}'}(1) \neq P^{\mathfrak{a}}(2)$ in i) excludes the case such that simultaneous application of $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ is ambiguous. The case ii) and the remaining cases in (C_4) are explained and illustrated in a similar way.

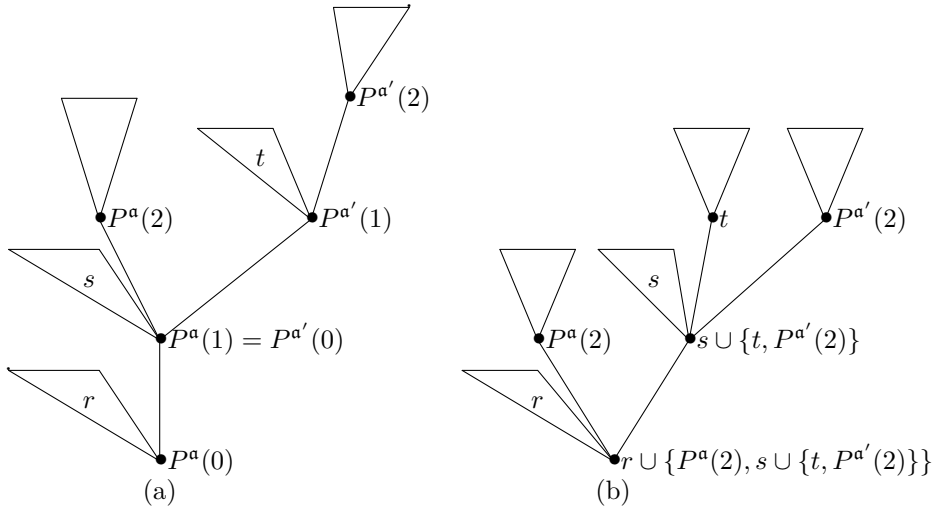
Ad (C_5). One explains and illustrates this condition in a way similar to (C_1) and (C_3).

Ad (C_6). One explains and illustrates this condition in a way similar to (C_4). We point out here that for two different local actions $\mathfrak{a} \in \mathcal{A}_{\mathrm{diss}}$ and $\mathfrak{a}' \in \mathcal{A}_{\mathrm{out}}$ with $\mathrm{scope}(\mathfrak{a}) \cap \mathrm{scope}(\mathfrak{a}') \neq \varnothing$ the condition

$$\{P^{\mathfrak{a}}(0), P^{\mathfrak{a}}(1)\} \cap \{P^{\mathfrak{a}'}(1), P^{\mathfrak{a}'}(2\} = \{P^{\mathfrak{a}}(0)\}$$

is equivalent to the disjunction of the following two conditions:

iii) $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(1)$ and $P^{\mathfrak{a}}(1) \neq P^{\mathfrak{a}'}(2)$,



where $r = P^{\mathfrak{a}}(0) - \{P^{\mathfrak{a}}(1)\},\ s = P^{\mathfrak{a}}(1) - \{P^{\mathfrak{a}}(2), P^{\mathfrak{a}'}(1)\}$
and $t = P^{\mathfrak{a}'}(1) - \{P^{\mathfrak{a}'}(2)\}$

Fig. 4.

iv) $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(2)$.

The condition $P^{\mathfrak{a}}(1) \neq P^{\mathfrak{a}'}(2)$ in iii) excludes the case such that simultaneous application of $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in the places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ is ambiguous.

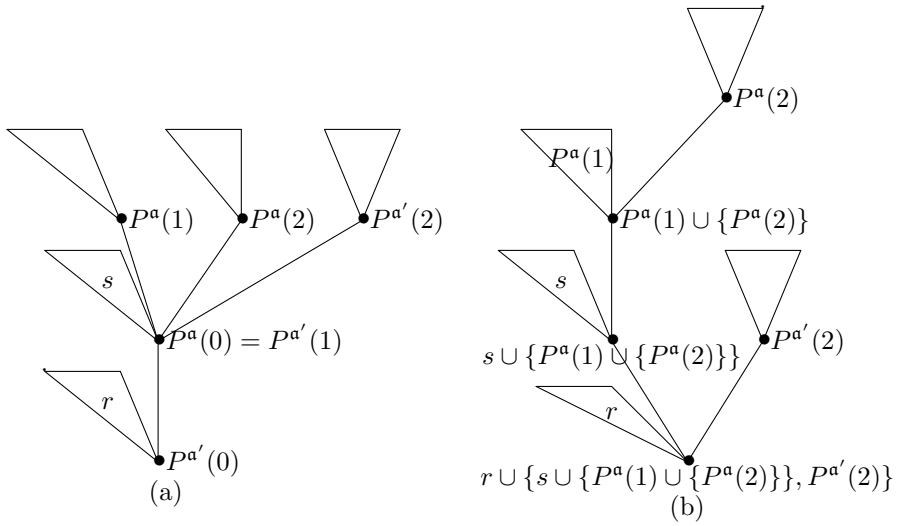
Ad (C_7). For two different local actions $\mathfrak{a} \in \mathcal{A}_{\text{in}}$ and $\mathfrak{a}' \in \mathcal{A}_{\text{out}}$ we explain the case of $\text{scope}(\mathfrak{a}) \cap \{P^{\mathfrak{a}'}(1), P^{\mathfrak{a}'}(2)\} = \{P^{\mathfrak{a}}(0)\}$ which is equivalent to the disjunction of the following two conditions:

v) $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(1)$ and $P^{\mathfrak{a}'}(2) \notin \{P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(2)\}$,
vi) $P^{\mathfrak{a}}(0) = P^{\mathfrak{a}'}(2)$.

The places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ in the case v) are illustrated in Fig. 5(a). The result of simultaneous application of $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in these places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$, respectively, is illustrated in Fig. 5(b), where $P^{\mathfrak{a}}(2)$ is "sent into" $P^{\mathfrak{a}}(1)$ and $P^{\mathfrak{a}'}(2)$ is "sent out" of $P^{\mathfrak{a}'}(1) = P^{\mathfrak{a}}(0)$ into $P^{\mathfrak{a}'}(0)$ simultaneously. The condition $P^{\mathfrak{a}'}(2) \notin \{P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(2)\}$ in v) excludes the case such that simultaneous application of $R^{\mathfrak{a}}$ and $R^{\mathfrak{a}'}$ in the places $P^{\mathfrak{a}}$ and $P^{\mathfrak{a}'}$ is ambiguous. The case vi) and the remaining cases in (C_7) are explained and illustrated in a similar way.

Let $\mathcal{A}$ be a proper set of local actions over a tree-like hereditarily finite set x. By the *result of evolutive transformation of x with respect to $\mathcal{A}$* we mean a set, denoted by $\text{Ap}(\mathcal{A}, x)$, which is defined inductively (with respect to the number of elements of $\mathcal{A}$ and the depth of x) by the following equations:

1) $\text{Ap}(\varnothing, x) = x$ and $\text{Ap}(\varnothing, \varnothing) = \varnothing$,



where $r = P^{\mathfrak{a}'}(0) - \{P^{\mathfrak{a}'}(1)\}$ and $s = P^{\mathfrak{a}}(0) - \{P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(2), P^{\mathfrak{a}'}(2)\}$

Fig. 5.

2) if $\mathcal{A} \neq \varnothing$, then $\mathrm{Ap}(\mathcal{A}, x) = (L \cap x) \cup \mathrm{Ap}^{\bullet}(\mathcal{A}, x)$ for

$$\mathrm{Ap}^{\bullet}(\mathcal{A}, x) = \bigcup_{1 \leq i \leq 4} \mathrm{Ap}_i(\mathcal{A}, x),$$

where

- $\mathrm{Ap}_1(\mathcal{A}, x) = \{\mathrm{Ap}(\mathcal{A} \restriction y, y) \,|\, y \in x \cap \mathrm{HF} \text{ and } y \notin \bigcup\{\mathrm{scope}(\mathfrak{a}) \,|\, P^{\mathfrak{a}}(0) = x \text{ and } \mathfrak{a} \in \mathcal{A}\}\}$,
- $\mathrm{Ap}_2(\mathcal{A}, x) = \bigcup\{\mathrm{Ap}^{\bullet}(\mathcal{A} \restriction P^{\mathfrak{a}}(1), P^{\mathfrak{a}}(1)) \,|\, P^{\mathfrak{a}}(0) = x \text{ and } \mathfrak{a} \in \mathcal{A}_{\mathrm{diss}}\}$,
- $\mathrm{Ap}_3(\mathcal{A}, x) = \{\mathrm{Ap}^{\bullet}(\mathcal{A} \restriction P^{\mathfrak{a}}(2), P^{\mathfrak{a}}(2)) \,|\, P^{\mathfrak{a}}(0) = x \text{ and } \mathfrak{a} \in \mathcal{A}_{\mathrm{out}}\}$,
- $\mathrm{Ap}_4(\mathcal{A}, x) = \{\mathrm{Ap}^{\bullet}(\mathcal{A} \restriction (y - \boldsymbol{P}^y), y - \boldsymbol{P}^y) \cup \boldsymbol{Q}^y \,|\, y \in \mathrm{INOUT}^x_{\mathcal{A}}\}$ for

$$\mathrm{INOUT}^x_{\mathcal{A}} = \{P^{\mathfrak{a}}(1) \,|\, P^{\mathfrak{a}}(0) = x \text{ and } \mathfrak{a} \in \mathcal{A}_{\mathrm{in}} \cup \mathcal{A}_{\mathrm{out}}\},$$
$$\boldsymbol{P}^y = \{P^{\mathfrak{a}}(2) \,|\, P^{\mathfrak{a}}(1) = y \text{ and } \mathcal{A} \in \mathcal{A}_{\mathrm{out}}\},$$
$$\boldsymbol{Q}^y = \{\mathrm{Ap}^{\bullet}(\mathcal{A} \restriction P^{\mathfrak{a}}(2), P^{\mathfrak{a}}(2)) \,|\, P^{\mathfrak{a}}(1) = y \text{ and } \mathfrak{a} \in \mathcal{A}_{\mathrm{in}}\}.$$

The result $\mathrm{Ap}(\mathcal{A}, x)$ of evolutive transformation of x with respect to $\mathcal{A}$ is the result of simultaneous application of the rules $R^{\mathfrak{a}}$ in places $P^{\mathfrak{a}}$, respectively for $\mathfrak{a} \in \mathcal{A}$, such that $\mathrm{Ap}(\mathcal{A}, x)$ inherits some basic properties of x which are described in the following theorem.

Theorem 3. *Let x be a tree-like hereditarily finite set and let $\mathcal{A}$ be a proper set of local action over x. Then $\mathrm{Ap}(\mathcal{A}, x)$ is a tree-like hereditarily finite set.*

Proof. One proves the theorem by induction on the number of elements of $\mathcal{A}$ and the depth of x. Theorem 2 in Section 3 provides the first inductive step.

References

1. Ch. Alexander: Harmony-Seeking Computations. *International Journal of Unconventional Computation*, to appear; see also `http://www.livingneighborhoods.org`.
2. Ch. Alexander: A city is not a tree. *Architectural Forum*, 122, 1 (1965), 58–61; 2 (1965), 58–62; `http://www2.rudi.net/bookshelf/classics/city`.
3. Ch. Alexander: *Pattern Languages*. Oxford Univ. Press, New York, 1977.
4. J. Byrnes, W. Sieg: An abstract model for parallel computations: Gandy's Thesis. *The Monist*, 82, 1 (1999), 150–164.
5. L. Cardelli: Brane calculi. Interactions of biological membranes. In *Proc. Computational Methods in System Biology*, 2004, Springer, Berlin.
6. L. Cardelli, A.D. Gordon: Mobile ambients. In *Foundations of Software Science and Computation Structures*, Lecture Notes in Comput. Sci. 1378, Springer, Berlin, 1998, 140–155.
7. L. Cardelli, A.D. Gordon: Mobile ambients. Coordination. *Theoret. Comput. Sci.*, 240 (2000), 177–213.
8. L. Cardelli, Gh. Păun: An universality result for (mem)brane calculus based on mate/drip operations. In *Proc. Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop, January 31–February 2, 2005 (M.A. Gutiérrez-Naranjo et al., eds.), 75–94.
9. N. De Pisapia: *Gandy Machines: an Abstract Model for Parallel Computations, for Turing Machines, the Game of Life, and Artificial Neural Networks*. M.S. Thesis, Carnegie Mellon Univ., Pittsburgh 2000, `http://artsci.wustl.edu/~ndepisap`.
10. R. Gandy: Church's thesis and principles for mechanisms. In *The Kleene Symposium* (J. Barwise et al., eds.), North-Holland, Amsterdam, 1980, 123–148.
11. C. Graciani Diaz et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing*, Sevilla, January 30–February 3, 2006, vol. I and II, Sevilla, 2006.
12. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Fractals and P systems. In *Proceedings of Fourth Brainstorming Week on Membrane Computing*, Sevilla, January 30–February 3, 2006 (C. Graciani Diaz et al., eds.), Sevilla 2006, vol. II, 65–86.
13. M.A. Gutiérrez-Naranjo et al., eds.: *Proc. Cellular Computing* (*Complexity Aspects*). ESF PESC Exploratory Workshop, January 31–February 2, 2005, Sevilla, 2005.
14. H.J. Hoogeboom et al., eds.: *Pre-proceedings of the 7th Workshop on Membrane Computing WMC7*. 17–21 July 2006, Lorenz Center, Leiden, Leiden, 2006.
15. Membrane computing web page `http://psystems.disco.unimib.it`.
16. A. Obtułowicz: Mathematical (denotational) semantics of some reducts of Ambient Calculus and Brane Calculi. *Romanian Journal of Information Science and Technology*, to appear.
17. A. Obtułowicz: Gandy's principles for mechanisms and membrane computing. In *Proc. Cellular Computing (Complexity Aspects)*, ESF PESC Exploratory Workshop, January 31–February 2, 2005 (M. A. Gutiérrez-Naranjo et al., eds.), Sevilla, 2005, 267–276.
18. A. Obtułowicz: Relational membrane systems. In *Membrane Computing, 6th International Workshop, WMC 2005,* Vienna, Austria, July 18–21, 2005 (R. Freund et al., eds.), Lecture Notes in Comput. Sci. 3850, Springer, Berlin, 2006, 342–355.
19. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
20. N.A. Salingaros: *Fractals in New Architecture*. Katarxis 3, September 2004, `http://www.katarxis3.com/Salingaros-Fractals.htm`.

21. W. Sieg: *Computability Theory*. Seminar Lectures, University of Bologna, November 2004,
http://www.phil.cmu.edu/summerschool/2006/Sieg/computability_theory.pdf.
22. W. Sieg: Calculations by man and machine: conceptual analysis. *Lecture Notes in Logic*, 15 (2002), 390–409.

Twenty Six Research Topics About Spiking Neural P Systems

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucharest, Romania
and
Department of Computer Science and AI
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es

1 Foreword

To continue the tradition of the previous brainstorming weeks on membrane computing, I am collecting here a series of open problems and research topics, not about membrane computing in general, but about one of the directions of research which were pretty much investigated in the last year: spiking neural P systems. In general, one mentions issues which look of a broader nature, but also some precise problems are formulated. As usual with such lists of problems, the selection is subjective, by no means exhaustive.

Of course, choosing only problems related to spiking neural P systems does not mean that there are no longer enough problems waiting to be solved in the general framework of membrane computing – on contrarily (e.g., separate lists can refer to computational complexity issues, to dynamical systems approaches, etc.), but such problems tend to become rather specialized and technical at the present stage of the development of membrane computing. Instead, the membrane computing models with a neural inspiration are at the beginning of a systematic exploration, and, as claimed below, this area of research looks very promising.

2 Forecast

It is obvious that the (human) brain structure and functioning, from neurons, astrocytes, and other components to complex networks and complex (chemical, electrical, informational) processes taking place in it, should be – and only partially is – a major source of inspiration for informatics (I choose this more general term rather that the restrictive, but usual, "computer science", in order to stress

that I have in mind both mathematics *per se* and practice, both the theory of computability and the use of computing machineries). If biology is such a rich source of inspiration for informatics as natural computing proves, then the brain should be the "golden mine" of this intellectual enterprise. Risking a forecast, I believe that *if something really great is to appear in informatics in the near future, then this "something" will be suggested by the brain (and this will probably be placed at the level of "strategies" of computing, not at the "tactic" level* – just in balance with the two computing devices already learned from the brain activity and which can be considered the most central notions in informatics, the Turing machine and the finite automaton).

The previous statements do not intend to suggest that spiking neural P systems are the answer to this learning-from-brain challenge, but only to call (once again) the attention to this challenge. Becoming familiar with brain functioning, in whatever reductionistic framework (as spiking neural P systems investigation is), can however be useful. After all, "the road of one thousand miles starts with the first step", Lao Tze said... Let us make from spiking neural P systems "the first step".

3 Some (Neural) Generalities

The neuron is a highly specialized cell, at the same time intricate and simple, robust and fragile, like any other cell, but having the particularity of being involved (in general) in huge networks by means of the synapses established with partner neurons. It is not at all the intention of these lines to give any biological information from this area, but only to point out some of the peculiarities related to neurons and the brain: the functioning of each neuron assumes chemical, electrical, and informational processing at the same time; the axon is not a simple transmitter of impulses, but an information processor; in the communication between neurons the spiking activity plays a central role (which means that the distance in time between consecutive spikes is used to carry information, that is, time is a support of information); the neurons are not cooperating only through synapses, but their relationships are also regulated through the calcium waves controlled by the astrocytes, "eavesdroppers" of axons playing an important role in the neural communication; the brain displays a general emergent behavior which, to my knowledge and to my understanding, cannot be explained only in terms of neuron interrelationships (something is still missing in this picture, maybe of a quantum nature – as Penrose suggests, maybe related to the organization of parts, maybe of a still subtler or even unknown nature). Some of these ideas (especially spiking) are supposed to lead to "neural computing of the third generation", which suggests that already computer scientists are aware of the possibility of major progresses to be made (soon) on the basis of progresses in neuro-biology.

The bibliography of this note contains several titles, both from the general biology of the cell [1], general neurology [41], and from neural computing based on

spiking [3], [29], [16], [26], [27], [28], about the axon as an information processor [39], astrocytes and their role in the brain functioning [37], [40]. Of course, these titles are only meant to be initial "dendrites" to the huge bibliography related to (computer science approaches to) brain functioning.

4 Spiking Neural P Systems – Informal Presentation

Spiking neural P systems (SN P systems, for short) were introduced in [23] in the precise (and modest: trying to learn a new "mathematical game" from neurology, not to provide models to it) aim of incorporating in membrane computing ideas specific to spiking neurons; the intuitive goal was to have (1) a tissue-like P system with (2) only one (type of) object(s) in the cells – the *spike*, with (3) specific rules for evolving populations of spikes, and (4) making use of the time as a support of information.

In what follows, I briefly describe several classes of SN P systems investigated so far, as well as some of the main types of results obtained in this area.

In short, an SN P system (of the basic form – later called a *standard* SN P system) consists of a set of *neurons* placed in the nodes of a directed graph and sending signals (*spikes*, denoted in what follows by the symbol a) along the arcs of the graph (they are called *synapses*). The objects evolve by means of *spiking rules*, which are of the form $E/a^c \to a; d$, where E is a regular expression over $\{a\}$ and c, d are natural numbers, $c \geq 1, d \geq 0$. The meaning is that a neuron containing k spikes such that $a^k \in L(E), k \geq c$, can consume c spikes and produce one spike, after a delay of d steps. This spike is sent to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are *forgetting rules*, of the form $a^s \to \lambda$, with the meaning that $s \geq 1$ spikes are removed, provided that the neuron contains exactly s spikes.

An extension of theses type of rules was considered (with a mathematical motivation) in [30], [14]: rules of the form $E/a^c \to a^p; d$, with the meaning that when using the rule, c spikes are consumed and p spikes are produced (one assumes that $c \geq p$, not to produce more than consuming). Because p can be 0 or greater than 0, we obtain a generalization of both spiking and forgetting rules, while forgetting rules also have a regular expression associated with them.

An SN P system (with standard as well with extended rules) works in the following way. A global clock is assumed and in each time unit each neuron which can use a rule should do it (the system is synchronized), but the work of the system is sequential locally: only (at most) one rule is used in each neuron. One of the neurons is considered to be the *output neuron*, and its spikes are also sent to the environment. The moments of time when a spike is emitted by the output neuron are marked with 1, the other moments are marked with 0. This binary sequence is called the *spike train* of the system – it might be infinite if the computation does not stop.

With a spike train we can associate various numbers, which can be considered as *computed* (we also say *generated*) by an SN P system. For instance, in [23] only the distance between the first two spikes of a spike train was considered, then in [33] several extensions were examined: the distance between the first k spikes of a spike train, or the distances between all consecutive spikes, taking into account all intervals or only intervals that alternate, all computations or only halting computations, etc.

An SN P system can also work in the *accepting* mode: a neuron is designated as the *input neuron* and two spikes are introduced in it, at an interval of n steps; the number n is accepted if the computation halts.

Two main types of results were obtained: computational completeness in the case when no bound was imposed on the number of spikes present in the system, and a characterization of semilinear sets of numbers in the case when a bound was imposed.

Another attractive possibility is to consider the spike trains themselves as the result of a computation, and then we obtain a device generating a (binary) language. We can also consider input neurons and then an SN P system can work as a transducer. Such possibilities were investigated in [34]. Languages – even on arbitrary (i.e., not only binary) alphabets – can be obtained also in other ways: following the path of a designated spike across neurons, as proposed in [12], or using rules of the extended form mentioned above. Specifically, with a step when the system sends out i spikes, we associate a symbol b_i, and thus we get a language over an alphabet with as many symbols as the number of spikes simultaneously produced. This case was investigated in [14], where representations or characterizations of various families of languages were obtained. (An essential difference was found between the case when zero spikes sent out is interpreted as a symbol b_0 and the case when this is interpreted as inserting λ, the empty string, in the result.)

Other extensions were proposed in [21] and [20], where several output neurons were considered, thus producing vectors of numbers, not only numbers. A detailed typology of systems (and of sets of vectors generated) is investigated in the two papers mentioned above, with classes of vectors found in between the semilinear and the recursively enumerable ones.

The proofs of all computational completeness results known up to now in this area are based on simulating register machines. Starting the proofs from small universal register machines, as those produced in [25], one can find small universal SN P systems (working in the generating mode, as sketched above, or in the computing mode, i.e., having both an input and an output neuron and producing a number related to the input number). This idea was explored in [30] and the results are as follows: there are universal computing SN P systems with 84 neurons using standard rules and with only 49 neurons using extended rules. In the generative case, the best results are 79 and 50 neurons, respectively.

In the initial definition of SN P systems several ingredients are used (delay, forgetting rules), some of them of a general form (unrestricted synapse graph, unrestricted regular expressions). As shown in [19], several normal forms can be

found, in the sense that some ingredients can be removed or simplified without losing the computational completeness. For instance, the forgetting rules or the delay can be avoided, and the outdegree of the synapse graph can be bounded by 2, while the regular expressions from firing rules can be of very restricted forms. The dual problem, of the indegree bounding, was solved (affirmatively) in [35].

Besides using the rules of a neuron in the sequential mode introduced above, it is possible to also use the rules in a parallel way. A possibility was considered in [24]: when a rule is enabled, it is used as many times as possible, thus exhausting the spikes it can consume in that neuron. As proved in [24], SN P systems with the exhaustive use of rules are again universal, both in the accepting and the generative cases.

In the proof of these results the synchronization plays a crucial role, but both from a mathematical point of view and from a neuro-biological point of view it is rather natural to consider non-synchronized systems, where the use of rules is not obligatory: even if a neuron has a rule enabled in a given time unit, this rule is not obligatorily used, the neuron may remain still, maybe receiving spikes from the neighboring neurons; if the unused rule may be used later, it is used later, without any restriction on the interval when it has remained unused; if the new spikes made the rule non-applicable, then the computation continues in the new circumstances (maybe other rules are enabled now). This way of using the rules applies also to the output neuron, hence now the distance in time between the spikes sent out by the system is no longer relevant. That is why, for non-synchronized SN P systems we take as a result of a computation the total number of spikes sent out; this, in turn, makes necessary considering only halting computations (the computations never halting are ignored, they provide no output). Non-synchronized SN P systems were introduced and investigated in [7], where it is proved that SN P systems with extended rules are still equivalent with Turing machines (as generators of sets of natural numbers).

5 Some (More) Formal Definitions

To make clearer some of the subsequent formulations, I recall here the definition of central classes of SN P systems, but more details should be found in the papers mentioned in the bibliography. No general notions or notations from language or automata theory, computability, complexity, computer science in general, or membrane computing, are recalled.

A *spiking neural P system* (in short, an SN P system), of degree $m \geq 1$, is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);

2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i), 1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained by the neuron;
   b) R_i is a finite set of *rules* of the following general form:

$$E/a^c \to a^p; d,$$

   where E is a regular expression with a the only symbol used, $c \geq 1$, and $p, d \geq 0$, with $c \geq p$; if $p = 0$, then $d = 0$, too.
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$ with $(i, i) \notin syn$ for $1 \leq i \leq m$ (*synapses*);
4. $out \in \{1, 2, \ldots, m\}$ indicates the *output neuron*.

A rule $E/a^c \to a^p; d$ with $p \geq 1$ is called a *firing* (we also say *spiking*) *rule*; a rule $E/a^c \to a^p; d$ with $p = d = 0$ is written in the form $E/a^c \to \lambda$ and is called a *forgetting rule*. If $L(E) = \{a^c\}$, then the rules are written in the simplified form $a^c \to a^p; d$ and $a^c \to \lambda$. A system having only rules of the forms $E/a^c \to a; d$ and $a^c \to \lambda$ is said to be *restricted* (we also use to say that such a system is a *standard* one).

The rules are applied as follows: if the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \to a^p; d \in R_i$ (with $p \geq 1$) is enabled and it can be applied; applying it means that c spikes are consumed, only $k - c$ remain in the neuron, the neuron is fired, and it produces p spikes after d time units. If $d = 0$, then the spikes are emitted immediately, if $d = 1$, then the spikes are emitted in the next step, and so on. In the case $d \geq 1$, if the rule is used in step t, then in steps $t, t + 1, t + 2, \ldots, t + d - 1$ the neuron is *closed*, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends spikes along it, then the spikes are lost). In step $t + d$, the neuron spikes and becomes again open, hence can receive spikes (which can be used in step $t + d + 1$). The p spikes emitted by a neuron σ_i are replicated and they go to all neurons σ_j such that $(i, j) \in syn$ (each σ_j receives p spikes). If the rule is a forgetting one, hence with $p = 0$, then no spike is emitted (and the neuron cannot be closed, because also $d = 0$).

In the synchronized mode, considered up to now in all SN P systems investigations except [7], a global clock is assumed, marking the time for all neurons, and in each time unit, in each neuron which can use a rule, a rule must be used. Because two rules $E_1/a^{c_1} \to a^{p_1}; d_1$ and $E_2/a^{c_2} \to a^{p_2}; d_2$ can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron, and then one of them is chosen non-deterministically. Note that the neurons work in parallel (synchronously), but each neuron processes sequentially its spikes, using only one rule in each time unit.

The initial configuration of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron. During the computation, a configuration is described by both the number of spikes present in each neuron and by the state of the neuron, more precisely, by the number of steps to count down until it becomes open (this number is zero if the neuron is already open). Thus, $\langle r_1/t_1, \ldots, r_m/t_m \rangle$ is the configuration where neuron $\sigma_i, i = 1, 2, \ldots, m$ contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps; with this notation, the initial configuration is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$ (see an example in Figure 2).

Using the rules as suggested above, we can define transitions among configurations. Any sequence of transitions starting in the initial configuration is called a *computation*. A computation halts if it reaches a configuration where all neurons are open and no rule can be used. With any computation, halting or not, we associate a *spike train*, a sequence of digits 0 and 1, with 1 appearing in positions which indicate the steps when the output neuron sends spikes out of the system (we also say that the system itself spikes at that time). With any spike train we can associate various numbers, which are considered as computed (generated) by the system; in the spirit of spiking neural computing, the distance between certain spikes are usually taken as the result of a computation (e.g., the distance between the first two spikes). Because of the non-determinism in using the rules, a given system computes in this way a set of numbers. An SN P system can be also used in the accepting mode: a number n is introduced in the system in the form of the distance between two spikes entering a specified neuron, and this number is accepted if the computation eventually halts.

We denote by $N_{gen}(\Pi)$ the set of numbers generated (in the synchronized way) by a system Π in the form of the number of steps elapsed between the first two spikes of a spike train. Then, by $Spik_2SP_m(rule_k, cons_p, forg_q, del_d)$ we denote the family of such sets of numbers generated by systems with at most m neurons, each of them containing at most k rules, all of them of the standard form, and each rule consuming at most p spikes, forgetting at most q spikes, and having the delay at most d. When using extended SN P systems, we use $Spik_2EP_m(rule_k, cons_p, prod_q, del_d)$ to denote the family of sets $N_{gen}(\Pi)$ generated by systems with at most m neurons, each of them containing at most k rules (of the extended form), each spiking rule consuming at most p spikes, producing at most q spikes, and having the delay at most d. When any of the parameters m, k, p, q, d is not bounded, it is replaced by $*$. When using the rules in the exhausting or the non-synchronized mode, we write $N^{ex}_{gen}(\Pi), N^{nsyn}_{gen}(\Pi)$, respectively, and the superscripts ex and $nsyn$ are also added to $Spik$ in the families notation.

The notations should be changed when dealing with other sets of numbers than the distance between the first two spikes, with accepting systems, when generating or accepting languages, but I do not enter here into details. Instead, I close this section by introducing two important tools in presenting SN P systems, namely, the graphical representation and the transition diagram.

Figures 1, 2 are recalled from [9]. The graphical representation of an SN P system is rather intuitive: the neurons are represented by membranes, placed in the nodes of a directed graph whose arrows represent the synapses; an arrow also exits from the output neuron, pointing to the environment; in each neuron we specify the rules and the spikes present in the initial configuration.

Figure 1 represents the initial configuration of a system Π. We have three neurons, labeled with 1, 2, 3, with neuron σ_3 being the output one. Each neuron contains two rules, with neurons σ_1 and σ_2 having the same rules (firing rules which can be chosen in a non-deterministic way, the difference between them being in the delay from firing to spiking), and neuron σ_3 having one firing and one forgetting

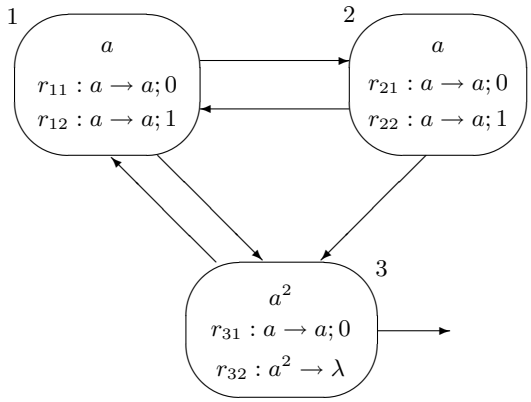


Fig. 1. The initial configuration of the SN P system Π

rule. In the figure, the rules are labeled, and these labels are useful below, in relation with Figure 2.

This figure can be used for analyzing the evolution of the system Π: because the system is finite, the number of configurations reachable from the initial configuration is finite, too, hence, we can place them in the nodes of a graph, and between two nodes/configurations we draw an arrow if and only if a direct transition is possible between them. In Figure 2 there are also indicated the rules used in each neuron, with the following conventions: for each r_{jk} we have written only the subscript jk, with **31** being written in bold face, in order to indicate that a spike is sent out of the system at that step; when a neuron $\sigma_j, j = 1, 2, 3$ uses no rule, we have written $j0$, and when it spikes (after being closed for one step), we write js.

The functioning of the system, both as a number generator and as a string generator, can easily be followed on this diagram.

6 Open Problems and Research Topics

The following list of problems should be read with the standard precautions: it is not meant to be exhaustive, there is no ordering of the problems (according to their significance/interest), some problems are very general, others are much more particular, in many cases the formulation is preliminary/informal and addressing the problem should start with a precise/suitable formulation, in many cases related results exist in the literature, and so on. Most problems are stated in a short way, with reference to the discussion from Section 4 and the definitions from Section 5.

A. Let us start with a general and natural idea: linking the study of SN P systems with neural computing. This can be a rich source of ideas, based on trans-

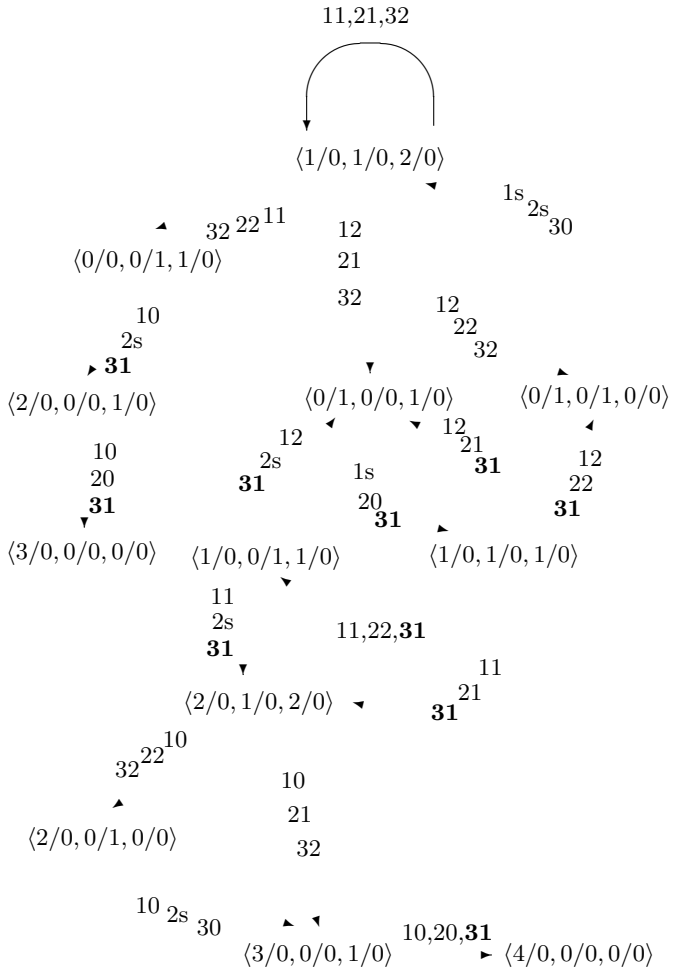


Fig. 2. The transition diagram of system Π from Figure 1

ferring from an area to the other one research topics which make sense also in the destination framework. What means, for instance, training (in general, learning, adaptation, evolving) in terms of SN P systems? More elementary: what means solving a problem by using an SN P system, implicitly, what means to solve a problem in a better way? Maybe the starting point should not be (only) neural computing, which is already an abstract, specialized, reductionistic framework, but (also) from neurology, from learning in the general psycho-pedagogical sense.

B. This problem is related to another general, natural, and important one: bringing more ingredients from neurology. Just a few quick ideas: considering an energy associated with firing/spiking; taking into considerations the antiport pro-

cesses which are performed in synapses; introducing circadian periodicity in the functioning of neurons and of nets of neurons, with "tiredness", "resting periods", etc.

C. In particular, the recent discoveries related to the role of astrocytes in the functioning of the brain need to be examined and formalized. Astrocytes are a class of cells that form a supporting and insulating structure for the neurons, but also participate in the process of communication between neurons. They "listen" the spikes passing along axons and accordingly regulate the release of neurotransmitters from the nerve terminals, thus relating in an intricate way the functioning of different neighboring axons. The regulation is either excitatory or inhibitory, and it is done by means of calcium waves. I refer to [37] and [40] for further details – and further references. How can astrocytes be considered in an SN P system and with what consequences?

D. The neuron-astrocyte coupling is based on signaling pathways of a kind which reminds the controlling pathways which were recently modeled and simulated in terms of P systems in many papers, and this suggests the next general research challenge: applications (in neurology). This is perhaps a too ambitious goal at this stage of the development of the study of SN P systems and it is first necessary to have answers to the previous two problems, but it is important to keep in mind the possibility of applications when devising new classes of SN P systems. It is difficult to forecast which would be the most promising types of applications – looking for conceptual clarifications, for analytical results, for computer experiments and simulations, for all these intertwined? Of course, the cooperation with a biologist/neurologist would be very important in this respect.

E. Making a step from neurobiology to mathematics, the problem appears to consider systems using more than one type of spikes. At the first sight, this is against the spirit of spiking neural computing, and can lead to standard membrane systems. Still, the question makes sense in various setups. For instance, neurology deals both with excitatory and inhibitory impulses, both in neurons and at the level of astrocytes. How inhibitory spikes can be defined and used?

F. Then, there are features of SN P systems which were not considered for general P systems. Using a regular expression for enabling a rule looks like controlling the application of rules by means of promoters, inhibitors, activators, but a notion of delay does not exits in membrane computing. Can it be of any interest also for usual P systems? Then, defining the result of a computation in a P system in terms of the time elapsed between two specified events, in particular, sending a given object outside, was briefly investigated in [5], but this issue deserves further research efforts.

G. Conversely, there are many ingredients of usual P systems which were not considered for SN P systems and might make sense also in this area, at least at a mathematical level. Of a particular interest can be tools to exponentially increase the working space in a polynomial (if possible, even linear) time, for

instance, by operations similar to cell division and cell creation in P systems with active membranes. How new neurons can be created (added to a system) in such a way to make possible polynomial solutions to computationally hard (typically, **NP**-complete) problems? The brain is supposed to be a very efficient computing device – how SN P systems can be made efficient from this point of view?

H. This touches a more general issue, that of considering SN P systems with a dynamical structure. The dynamism can be achieved both in terms of neurons and synapses, or only for synapses. From birth to maturity, the brain essentially evolves at the level of synapses, learning means establishing new synapses, cutting them, making them more stable/fast when used frequently, and so on and so forth. How this can be incorporated in SN P systems? A related idea is to associate a duration to each synapse (which is not of interest when the duration is constant), and to vary it in time, according to the intensity of using that synapse, and this looks rather motivated from a learning point of view.

I. Making synapses to have a duration or a length, depending on their use, can be related to a similar idea [8] at the level of spikes: considering a duration of life also for spikes, in the form of a decaying constant associated with them (at the level of the whole system, or locally, for each neuron). If a spike is not used a number of steps larger than the decaying threshold, then it is removed (a sort of forgetting rules are thus implicitly acting, depending on the age of each spike).

J. Moving further to theoretical issues, let us consider an idea related both to "classic" membrane computing and to the efficiency issue: using the rules in a parallel manner. This has been already considered in [24], in the particular form of using the rules in the exhaustive mode: if a neuron contains $kn + r$ spikes and has a rule $E/a^n \to a; d$ such that $a^{kn+r} \in L(E)$ and $k \geq 1, 0 \leq r < n$, then the rule is enabled and it is applied k times; kn spikes are consumed, r remain unused, and k are produced. Besides continuing the research from [24] (where it is only proved that SN P systems with an exhaustive use of rules are Turing complete both in the generative and the accepting modes), several other problems remain to be investigated. Actually, most problems usually considered for SN P systems with a sequential use of rules can be formulated also for the exhaustive mode: generating or accepting languages, translating strings of infinite sequences, looking for small universal systems, etc.

K. Then, the problem arises to consider other forms of parallelism, at the level of each neuron or at the level of the whole system. What about using several rules at the same time, in the same way as the rules of a usual P system are applied in the maximally parallel manner? Variants inspired from grammar systems area can also be considered, thus obtaining a bounded parallelism: at least k, at most k, exactly k rules to be used at a time. This last idea can be transferred also at the level of neurons: in each step, only a prescribed number of neurons, non-deterministically chosen, to be active. Finally, one can borrow to this area the idea of minimal parallelism from [15]: when a neuron can use at least one rule, then

at least one must be used, without any restriction about how many. A significant non-determinism is introduced in this way in the functioning of the system.

L. When the number of rules to be used in each neuron is "at least zero" (and this is equivalent with making evolve "at least zero" neurons at a time), we get the rather natural idea of a non-synchronized functioning of an SN P system. In such a case, in each time unit, any neuron is free to use a rule or not.

I have described the functioning of such a system in the end of Section 4. I only recall that, because now "the time does not matter", the spike train can have arbitrarily many occurrences of 0 between any two occurrences of 1, hence the result of a computation can no longer be defined in terms of the steps between two consecutive spikes, but as the total number of spikes sent into the environment by (or contained in) the output neuron. In this way, only halting computations can be considered as successful.

In [7] it is proved that SN P systems with extended rules are Turing equivalent even in the non-synchronized case, but the problem was left open whether this is true also for systems using standard rules. The conjecture is that this does not happens, hence that synchronization plays a crucial role in this case.

Similar to the exhaustive mode of using rules, also the non-synchronization can be investigated in relation with many types of problems usual in the SN P systems area: handling languages, looking for small universal systems, etc.

M. A related issue is to consider the class of systems for which the synchronization does not matter, i.e., they generate/accept the same set of numbers in both modes. Furthermore, time-free, clock-free, time-independent systems can be considered, in the same way as in [4], [6], [38].

N. Several times so far, the idea of efficiency was invoked, with the need to introduce new ingredients in the area of SN P systems in such a way to make possible polynomial solutions to intractable problems. Actually, such a possibility was already considered in [10]: making use use of arbitrarily large pre-computed resources. The framework is the following: an arbitrarily large net of neurons is given, of a regular form (as the synapse graph) and with only a few types of neurons (as contents and rules) repeated indefinitely; the problem to be solved is plug-in by introducing a polynomial number of spikes in certain neurons (of course, polynomially many), then the system is left to work autonomously; in a polynomial time, it activates an exponential number of neurons, and, after a polynomial time, it outputs the solution to the problem. The problem considered in [10] was the `SAT` problem.

This strategy is attractive from a natural computing point of view (we may assume that the brain is arbitrarily large with respect to the small number of neurons currently used, the same with the cells in liver, etc.), but it has no counterpart in the classic complexity theory. A formal framework for defining acceptable solutions to problems by making use of pre-computed resources needs to be formulated and investigated. What kind of pre-computed workspace is acceptable, i.e., how much information may be provided for free there, what kind of net of neurons and what

kind of neurons? (We have to prevent "cheating" by already placing the answer to the problem in the given resources and then "solving" the problem just by visiting the right place where the solution waits to be read.) What means introducing a problem in the existing device? (Only spikes, also rules, or maybe also synapses?) Defining complexity classes in this case remains as an interesting research topic.

O. Coming back to the initial definitions, there are several technical issues which are worth clarifying (most probably, for universality and maybe also for efficiency results, they do not matter, but it is also possible to exist other situations where these details matter). For instance, the self-synapses are not allowed in the synapse graph. However, a neuron with a rule $a \to a$ and a self-synapse can work forever, hence it can be used for rejecting a computation in the case when successful computations should halt. Similarly, (in the initial definition from [23]) the forgetting rules $a^s \to \lambda$ were supposed to have $a^s \notin L(E)$ for all spiking rules $E/a^c \to a; d$ from the same neuron, while in extended rules $E/a^c \to a^p; d$ it was assumed that $c \geq p$. Is there any situation where these restrictions make a difference? Then, in [19] it was shown that some of the ingredients used in the definition of SN P systems with standard rules can be avoided. This is the case with the delay, the forgetting rules, the generality of regular expressions. Can these normal forms be combined, thus avoiding at the same time two of the mentioned features?

P. What then about using a kind of rules of a more general form, namely $E/a^n \to a^{f(n)}; d$, where f is a partial function from natural numbers to natural numbers (maybe with the property $f(n) \leq n$ for all n for which f is defined), and used as follows: if the neuron contains k spikes such that $a^k \in L(E)$, then c of them are consumed and $f(c)$ are created, for $c = \max\{n \in \mathbf{N} \mid n \leq k$, and $f(n)$ is defined$\}$; if f is defined for no n smaller than or equal to k, then the rule cannot be applied. This kind of rules looks both adequate from a neurobiological point of view (the sigmoid excitation function can be captured) and powerful from a mathematical point of view (arbitrarily many spikes can be consumed at a time, and arbitrarily many produced).

Q. A standard problem when dealing with accepting devices concerns the difference between deterministic and non-deterministic systems. Are they different in power, does determinism imply a decrease of the computing power? Up to now, all computability completeness proofs for the accepting version of SN P systems of various types were obtained for deterministic systems. Are there classes (maybe non-universal) for which the determinism matters?

Actually, the problem can be refined. The determinism is defined usually in terms of non-branching during computations: a computation is deterministic if for every configuration there is (at most) one next configuration. A first subtle point: is this requested for *all* possible configurations or only for all configurations which are *reachable* from the initial one?

Maybe more interesting for SN P systems is the possibility to define a *strong determinism*, in terms of rules: an SN P system is said to be strongly deterministic

if $L(E) \cap L(E') = \emptyset$ for all rules $E/a^c \to a; d$ and $E'/a^{c'} \to a; d'$ from any neuron. Obviously, such a system is deterministic also when defining this notion in terms of branching (even for arbitrary configurations, not only for the reachable ones).

Is any class of SN P systems for which these types of determinism are separated?

R. Different from the case of general P systems, where finding infinite hierarchies on the number of membranes was a long awaited result, for SN P systems one can easily find such hierarchies, based on the characterization of semilinear sets of numbers (by means of systems with a bounded number of spikes in their neurons): if for each finite automaton with n states (using only one symbol) one can find an equivalent SN P system with $g(n)$ neurons, and, conversely, for each SN P system with m neurons one can find an equivalent (i.e., generating strings over an one-letter alphabet whose lengths are numbers generated/accepted by the SN P system) with $h(m)$ states, then, because there is an infinite hierarchy of regular one-letter languages in terms of states, we get an infinite hierarchy of sets of numbers with respect to the number of neurons. Still, several problems arise here. First, not always the characterization of semilinear sets of numbers is based on proving the equivalence of bounded SN P systems with the finite automata. Then, this reasoning only proves that the hierarchy is infinite, not also that it is "dense" (*connected* is the term used in classic descriptional complexity: there is n_0 such that for each $n \geq n_0$ there is a set Q_n whose neuron-complexity is exactly n). Finally, what about finding classes intermediate between semilinear and Turing computable for which the hierarchy on the number of neurons is infinite (maybe connected)?

S. The previous question directly suggests two others. The first one is looking for small universal SN P systems (here "universal" is understood in the sense of "programmable" – the existence of a fixed system which can simulate any particular system after introducing a code of the particular system in it – not in the sense of "Turing complete", although there is a direct connection between these two notions). This question is considered in [30] for SN P systems with standard and with extended rules, working either in the computing mode or in the generating mode. For standard rules, 84 and 76 neurons were used, while for extended rules 49 and 50 neurons were used, respectively. Are these results optimal? A negative answer is expected (however, a significant improvement is not very probable). What about universal SN P systems of other types – in particular, with exhaustive or non-synchronized use of rules?

T. Problem **R** also suggests to look for classes of SN P systems which are not equivalent with Turing machines, but also not computing only semilinear sets of numbers, hence equivalent in power with finite automata. This does not look as an easy question, but it is rather interesting, in view of the possibility of finding classes of systems with decidable properties, but (significantly) more powerful than bounded SN P systems. Such a class would be attractive also from the point of

view of applications, because of the possibility of finding properties of the modeled processes by analytical, algorithmic means.

U. Again in a direct continuation with the previous issue, there appears the need to find characterizations of classes of languages, other than finite, regular, and recursively enumerable, in terms of SN P systems. The investigations from [9], [12], [14] have left open these questions, and this fits with the general situation in membrane computing (as well as in DNA computing): the Chomsky hierarchy seems not to have a counterpart in nature, families like those of linear, context-free, and context-sensitive languages do not have (easy) characterizations in bio-inspired computing models. The same challenge appears for families of languages generated by L systems (sometimes, with the exception of ET0L languages).

V. L systems can be related with SN P systems also at the level of infinite sequences: both by iterating morphisms (D0L systems) and by taking infinite spike trains we can get classes of infinite sequences. Directly as spike trains we have binary sequences, but, for extended rules (and for SN P systems with a parallel use of rules) we can get as an output of a computation a string or an infinite sequence over an arbitrary alphabet. A preliminary examination of the binary case was done in [34], but many problems were left open, starting with the comparison of SN P systems as tools for handling infinite sequences (of bits) with other tools from language and automata theory (with ω-languages computed by finite automata, Turing machines, etc.) and with known infinite sequences, e.g., those from [42].

A particular problem from [34] is the following. SN P systems cannot compute arbitrary morphisms, but only length preserving morphisms (codes). An extension of these latter functions are the so-called *k-block morphisms*, which are functions $f : \{0,1\}^k \longrightarrow \{0,1\}^k$ (for a given $k \geq 1$) prolonged to $f : \{0,1\}^\omega \longrightarrow \{0,1\}^\omega$ by $f(x_1x_2\ldots) = f(x_1)f(x_2)\ldots$. In [34] it is only shown that 2-block morphisms can be computed by SN P systems, and the conjecture was formulated that this is true for any k.

In general, more should be found about the use of SN P systems as tools for transducing strings and infinite sequences.

W. Maybe useful in addressing the previous problem – and interesting also from other points of view (e.g., if starting investigations in terms of process algebra), is the issue of compositionality: looking for ways to pass from given systems to more complex systems, for instance, to systems generating/accepting the result of an operation between the sets of numbers or the languages generated/accepted by the initial systems. Morphisms were mentioned also above, but there are many other set-theoretic or language-theoretic operations to consider, as well as serial and parallel composition, embedding as a subsystem, etc. Of course, a central point in such operations is that of synchronization. It is expected that the case of non-synchronized systems is much easier (maybe, instead, less interesting theoretically).

X. I have mentioned at the beginning of these notes that the axon is not a simple transmitter of spikes, but a complex information processor. This suggests

considering computing models based on the axon functioning (Ranvier nodes amplification of impulses, and other processes) and a preliminary investigation was carried out in [13]. Many questions remain to be clarified in this area (see also the questions formulated in [13]), but a more general and probably more interesting problem appears, namely, of combining neurons and axons (as information processing units) in a global model; maybe also astrocytes can be added, thus obtaining a more complex model, closer to reality.

Y. I will conclude with two general issues, where nothing was done up to now. First, SN P systems have a direct (pictural) similarity with Petri nets, where tokens (like spikes) are moved through the net according to specific rules. Bridging the two areas looks then rather natural – with "bridging" understood as a move of notions, tools, results in both directions, from Petri nets to SN P systems and the other way round.

Z. Then, directly important for possible applications is the study of SN P systems as dynamical systems, hence not focusing on their output, but on their evolution, on the properties of the sequences of configurations reachable from each other. The whole panoply of questions from the (discrete) dynamical systems theory can be brought here, much similar to what happened in general membrane computing.

As it was the case also other times, I have to stop because of reaching the end of the alphabet... – with the hope that the reader will shorten this list by providing answers to some problems.

7 Final Remarks

Many other open problems and research topics can be found in the papers devoted to SN P systems – the interested reader can check the titles below in this respect (the bibliography contains all papers about SN P systems which I was aware of at the beginning of November 2006). On the other hand, because the research in this area is quite vivid, it is possible that some of these problems were solved at the same time or shortly after writing these notes, without being possible to mention the respective results here. That is why, the reader is advised to follow the developments in this area, for instance, through the information periodically updated at the Milano web page [43].

References

1. B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, P. Walter: *Molecular Biology of the Cell*, 4th ed. Garland Science, New York, 2002.
2. A. Alhazov, R. Freund, M. Oswald, M. Slavkovik: Extended variants of spiking neural P systems generating strings and vectors of non-negative integers. In *Pre-proc. WMC7*, Leiden, July 2006, 88–101, and [18], 123–134.

3. A. Carnell, D. Richardson: Parallel computation in spiking neural nets. Available at `http://people.bath.ac.uk/masdr/`.
4. M. Cavaliere, V. Deufemia: On time-free P systems. *Intern. J. Found. Computer Sci.*, 17, 1 (2006), 69–90.
5. M. Cavaliere, R. Freund, A. Leitsch, Gh. Păun: Event-related outputs of computations in P systems. In *Proc. Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, RGNC Report 01/2005, 107–122.
6. M. Cavaliere, D. Sburlan: Time-independent P systems. In *Membrane Computing. International Workshop WMC5, Milano, Italy, 2004*, LNCS 3365, Springer, 2005, 239–258.
7. M. Cavaliere, M. Ionescu, Gh. Păun: Asynchronous spiking neural P systems. Submitted, 2007.
8. M. Cavaliere, M. Ionescu: SN P systems with decaying spikes. Personal communication, 2006.
9. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In [17], Vol. I, 169–194.
10. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. In [17], Vol. I, 195–206, and *Proc. 8th Intern. Conf. on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
11. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules: Universality and languages. *Natural Computing*, to appear.
12. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. In [17], Vol. I, 207–224, and *Proc. Eighth International Workshop on Descriptional Complexity of Formal Systems* (DCFS 2006), June 21-23, 2006, Las Cruces, New Mexico, USA, 94–105.
13. H. Chen, T.-O. Ishdorj, Gh. Păun: Computing along the axon. In [17], Vol. I, 225–240, and *Pre-proc. BIC-TA*, Wuhan, 2006, 60–70.
14. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. In [17], Vol. I, 241–265.
15. G. Ciobanu, L. Pan, Gh. Păun, M.J. Pérez-Jiménez: P systems with minimal parallelism. *Theoretical Computer Sci.*, to appear.
16. W. Gerstner, W Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
17. M.A. Gutiérrez-Naranjo et al., eds.: *Proceedings of Fourth Brainstorming Week on Membrane Computing*, Febr. 2006, Fenix Editora, Sevilla, 2006.
18. H.J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Membrane Computing, International Workshop, WMC7, Leiden, The Netherlands, 2006, Revised, Selected, and Invited Papers*. LNCS 4361, Springer, Berlin, 2006.
19. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. In [17], Vol. II, 105–136, and *Theoretical Computer Sci.*, 372, 2-3 (2007), 196–217.
20. O.H. Ibarra, S. Woodworth: Characterizations of some restricted spiking neural P systems. In *Pre-proc. WMC7*, Leiden July 2006, 387–396, and [18], 424–442.
21. O.H. Ibarra, S. Woodworth, F. Yu, A. Păun: On spiking neural P systems and partially blind counter machines. In *Proc. UC2006*, York, LNCS 4135, Springer, 2006, 113–129.
22. M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems. In *Proc. DNA12* (C. Mao, Y.

Yokomori, B.-T. Zhang, eds.), Seoul, June 2006, 32–42.
23. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
24. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with exhaustive use of rules. *Intern. J. Unconventional Computing*, to appear.
25. I. Korec: Small universal register machines. *Theoretical Computer Science*, 168 (1996), 267–301.
26. W. Maass: Computing with spikes. *Special Issue on Foundations of Information Processing of TELEMATIK*, 8, 1 (2002), 32–36.
27. W. Maass: Paradigms for computing with spiking neurons. In *Models of Neural Networks. Early Vision and Attention* (J.L. van Hemmen, J.D. Cowen, E. Domany, eds.), Springer, Berlin, 2002, 373–402.
28. W. Maass, C. Bishop, eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, 1999.
29. C. O'Dwyer, D. Richardson: Spiking neural nets with symbolic internal state. Available at `http://people.bath.ac.uk/masdr/`.
30. A. Păun, Gh. Păun: Small universal spiking neural P systems. In [17], Vol. II, 213–234, and *BioSystems*, in press.
31. Gh. Păun: *Membrane Computing – An Introduction*. Springer, Berlin, 2002.
32. Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. In *Proc. of Developments in Language Theory Conference, DLT 2006*, Santa Barbara, CA, June 2006, LNCS 4036, Springer, Berlin, 2006, 20–35.
33. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17, 4 (2006), 975–1002.
34. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Submitted, 2006.
35. Gh. Păun, M.J. Perez-Jimenez, A. Salomaa: Bounding the indegree of spiking neural P systems. *TUCS Technical Report* 773, 2006.
36. Gh. Păun, M.J. Pérez-Jiménez, A. Salomaa: Spiking neural P systems. An early survey. *Intern. J. Fund. Computer Sci.*, 2007.
37. G. Perea, A. Araque: Communication between astrocytes and neurons: a complex language. *J. Physiol. – Paris*, 96 (2002), 199–207.
38. D. Sburlan: *Promoting and Inhibiting Contexts in Membrane Computing*. PhD Thesis, Univ. Sevilla, Spania, 2006.
39. I. Segev, E. Schneidman, Axons as computing devices: basic insights gained from models. *J. Physiol. (Paris)*, 93 (1999), 263–270.
40. X. Shen, P. De Wilde: Long-term neuronal behavior caused by two synaptic modification mechanisms. *Neurocomputing*, to appear.
41. G.M. Shepherd: *Neurobiology*. Oxford University Press, NY Oxford, 1994.
42. N.J.A. Sloane, S. Plouffe: *The Encyclopedia of Integer Sequences*. Academic Press, New York, 1995.
43. The P Systems Web Page: `http://psystems.disco.unimib.it`.

Membrane Computing Schema Based on String Insertions

Mario J. Pérez-Jiménez[1], Takashi Yokomori[2]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

[2] Department of Mathematics
Faculty of Education and Integrated Arts and Sciences
Waseda University, 1-6-1 Nishiwaseda, Shinjuku-ku
Tokyo 169-8050, Japan
yokomori@waseda.jp

Summary. In this note we introduce the notion of a membrane computing schema for string objects. We propose a computing schema for a membrane network (i.e., tissue-like membrane system) where each membrane performs unique type of operations at a time and sends the result to others connected through the channel. The distinguished features of the computing models obtained from the schema are:

1. only *context-free insertion operations* are used for string generation,
2. some membranes assume filtering functions for *structured objects(molecules)*,
3. the generating model and accepting model are obtained in the *same schema*, and both are computationally universal,
4. several known rewriting systems with universal computability can be reformulated in terms of membrane computing schema in a uniform manner.

The first feature provides the model with a simple uniform structure which facilitates a biological implementation of the model, while the second feature suggests further feasibility of the model in terms of DNA complementarity.

Through the third and fourth features, one may have a unified view of a variety of existing rewriting systems with Turing computability in the framework of membrane computing paradigm.

1 Introduction

In the theory of bio-inspired computing models, membrane systems (or P systems) have been widely studied from various aspects of the computability such as the optimal system designs, the functional relations among many ingredients in different levels of computing components, the computational complexity and so forth.

Up to the present, major concerns are focused on the computational capability of multisets of certain objects in a membrane structure represented by a rooted tree, and there are a relatively limited amount of works in the membrane structure of other types (like a network or graph) on string objects and their languages; those are, for example, in the context of P system on graph structure ([15]), of the tissue P systems ([10]) and of spiking neural P systems ([2, 6]).

On the other hand, in DNA computing theory, a string generating device called insertion-deletion system has been proposed and investigated from the uniqueness of non-rewriting nature in generating string objects. Among others, string insertion operation with no context is of our particular interests, because of the relevance to biological feasibility in terms of DNA sequences.

In this paper, we are concerned with tissue-like membrane systems with string insertion operations and investigate the computational capability of those systems. By using the framework of tissue-like membrane systems, however, our major focus is on studying the new aspects of the computational mechanisms used in a variety of existing models based on string rewriting.

To this aim, we propose the notion of a membrane computing schema which provides a unified view and framework to investigate new aspects of the variety of computational mechanisms. That is, by a membrane computing schema Π, we represent a *core* structure of the computing model M at issue. At the same time, we also consider an interpretation I to Π which specifies the *details* of M. Then, we have M that is embodied as a tissue-like membrane system $I(\Pi)$ with string insertion operation.

The advantages of this schematic approach to computing are the following: (1) High transparency of the computing mechanism is obtained from separating the skeletal (core) part from other detailed specificity of the computation. (2) Structural modularity of the computing model facilitates our better understanding of the computing mechanism.

With this framework we will present not only new results of the computing models with universal computability but also a unified view of those models from the framework of tissue-like membrane system with string insertion operations.

2 Preliminaries

We assume the reader to be familiar with all formal language notions and notations in standard use. For unexplained details, consult, e.g., [14, 16].

For a string x over an alphabet V (i.e., x in V^*), $lg(x)$ denotes the length of x. For the empty string, we denote it by λ. For an alphabet V, $\overline{V} = \{\overline{a} \mid a \in V\}$. A binary relation ρ over V is called an *involution* if ρ is injective and ρ^2 is an identity (i.e., for any $a \in V$, if we write $\rho(a) = \overline{a}$, then it holds that $\rho(\overline{a}) = a$). A *Dyck* language D_k over $V \cup \overline{V}$ is a language generated by a context-free grammar $G = (\{S\}, V, P, S)$, where $P = \{S \to SS, S \to \lambda\} \cup \{S \to aS\overline{a} \mid a \in V\}$ and k is the cardinality of V.

An *insertion system* ([9]) is a triple $\gamma = (V, P, A)$, where V is an alphabet, A is a finite set of strings over V called axioms, and P is a finite set of insertion rules. An *insertion rule* is of the form (u, x, v), where $u, x, v \in V^*$. We define the relation $\mapsto$ on V^* by $w \mapsto z$ iff $w = w_1uvw_2$ and $z = w_1uxvw_2$ for some insertion rule $(u, x, v) \in P$, $w_1, w_2 \in V^*$. As usual $\mapsto^*$ denotes the reflexive and transitive closure of $\mapsto$. The *insertion language* generated by γ is defined as follows: $L(\gamma) = \{w \in V^* | \ s \mapsto^* w, \ s \in A\}$. An insertion rule of the form (λ, x, λ) is said to be *context-free*, and we denote it by $\lambda \to x$.

We denote by RE, CF, LIN and RG the families of recursively enumerable languages, of context-free languages, of linear languages, and of regular languages, respectively.

A *matrix grammar with appearance checking* is a construct $G = (N, T, S, M, F)$, where N, T are disjoint alphabets, $S \in N$, M is a finite set of sequences of the form $(A_1 \to x_1, \ldots, A_n \to x_n)$, $n \geq 1$, of context-free rules over $N \cup T$ (with $A_i \in N, x_i \in (N \cup T)^*$, in all cases), and F is a set of occurrences of rules in M (we say that N is the nonterminal alphabet, T is the terminal alphabet, S is the axiom, while the elements of M are called matrices).

For $w, z \in (N \cup T)^*$ we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1, \ldots, A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*, 1 \leq i \leq n+1$, such that $w = w_1, z = w_{n+1}$, and, for all $1 \leq i \leq n$, either $w_i = w'_i A_i w''_i, w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or $w_i = w_{i+1}$, A_i does not appear in w_i, and the rule $A_i \to x_i$ appears in F. (The rules of a matrix are applied in order, possibly skipping the rules in F if they cannot be applied; we say that these rules are applied in the *appearance checking* mode.) If $F = \emptyset$, then the grammar is said to be without appearance checking (and F is no longer mentioned).

We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$.

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by MAT_{ac}. When we use only grammars without appearance checking, then the obtained family is denoted by MAT. It is known that $MAT \subset MAT_{ac} = RE$.

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

1. $(S \to XA)$, with $X \in N_1, A \in N_2$,
2. $(X \to Y, A \to x)$, with $X, Y \in N_1, A \in N_2, x \in N_2 \cup N_2^2 \cup T \cup \{\lambda\}$,
3. $(X \to Y, A \to \#)$, with $X, Y \in N_1, A \in N_2$,
4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2$, and $x \in T \cup \{\lambda\}$.

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

For each matrix grammar (with appearance checking) there effectively exists an equivalent matrix grammar (with appearance checking) in the binary normal

form. (Note that the definition of the binary normal form presented here is a variant of the one in [4].)

A *random context grammar* is a construct $G = (N, T, S, P)$, where N, T are disjoint alphabets, $S \in N$, P is a finite set of rules of the form $(A \to x, Q, R)$, where $A \to x$ is a context-free rule $(A \in N, x \in (N \cup T)^*)$, Q and R are subsets of N.

For $\alpha, \beta \in (N \cup T)^*$ we write $\alpha \Longrightarrow \beta$ iff $\alpha = uAv$, $\beta = uxv$ for some $u, v \in (N \cup T)^*$, $(A \to x, Q, R) \in P$, all symbols of Q appear in uv, and no symbol of R appears in uv. We denote by $\Longrightarrow^*$ the reflexive and transitive closure of the relation $\Longrightarrow$.

The language generated by G is defined by $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$. The family of languages of this form is denoted by RC. It is known that $RC = RE$ ([4]).

3 Membrane Computing Schema, Interpretation and Languages

We now introduce the notion of a membrane computing schema in a general form, then we will present a restricted version, from which a variety of specific computing models based on insertion operations and filtering can be obtained in the framework of a tissue-like membrane computing. That is, a membrane computing schema is given as a skeletal construct consisting of a number of *membranes* connected with synapses (or channels) whose structure may be taken as a kind of *tissue P systems* (e.g., [10]).

3.1 Membrane Computing Schema

A *membrane computing schema* of degree (k, p) is a construct

$$\Pi = (V, T, Syn, Com, Ope, Fil, i_s, Out),$$

where:

(i) V is a finite alphabet with an involution relation ρ called the *working alphabet*.
(ii) T is a subset of V called the *terminal alphabet*.
(iii) $Syn \subseteq (Com \times Ope) \cup (Ope \times Com) \cup (Com \times (SubFil \cup \{SF\})) \cup (\{FF\} \times Com) \cup (SubFil \times Ope) \cup \{(SF, FF), (FF, Out)\}$,
where $Com = \{Com_1, \cdots, Com_p\}$ called a set of *communication cells*,
$Ope = \{Ope_1, \cdots, Ope_k\}$ called a set of *operation cells*,
$Fil = \{SF, FF\} \cup SubFil$ called a set of *filtering cells*,
where $SubFil = \{FF_1, \cdots, FF_t\}$ called a set of *subfilter cells*.
Syn determines the tissue membrane structure of Π. (See Figure 1. In the figure, the thick arrow indicates multiple arrows of one-way or two-way directions.)

(iv) Each cell Com_i serves as a communication channel. That is, any string x in the cell Com_i is sent out to all the cells indicated by $Syn(Com_i)$.
(v) Each cell Ope_j consists of a finite number of rules $\{\sigma_{j1}, \cdots, \sigma_{js}\}$, where each σ_{ji} is a string insertion operation of the form : $\lambda \rightarrow u$, where $u \in V^*$.
(vi) SF and FF, called *structured filter* and *final filter*, are associated with two languages L_{SF} and L_{FF} over V, respectively. Further, each cell FF_i, called *subfilter*, is also associated with a language L_{FF_i}.
(vii) $i_s(= Com_1)$ is the distinguished cell (to designate some specific role).
(viii) Out is the *output cell* (for obtaining the outputs).

3.2 Interpretation of Π

In order to embody a membrane computing schema Π, we need to give more information about Π which specifies each operation in Ope, and materializes SF and FF. Let us consider such an interpretation I to the schema Π which enables us to have a computing model $I(\Pi)$ that is feasible in a practical sense.

[Notation] For any x in $Com \cup SubFil \cup \{FF\}$, let $Syn(x) = \{\ j\ \mid\ (x, y_j) \in Syn\}$ and for any y in $Ope \cup SupFil \cup \{SF\}$, let $Syn^{-1}(y) = \{\ i\ \mid\ (x_i, y) \in Syn\}$. (Note that both $Syn(x)$ and $Syn^{-1}(y)$ refer to sets of *indices*.)

Formally, an *interpretation* I to Π is a construct $I = (\{R_1, \cdots, R_k\}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \leq i \leq t\})$, where

(i) R_i specifies a set of insertion operations used in Ope_i (for $i = 1, \cdots, k$)
(ii) L_{SF} (L_{FF}) materializes a concrete specification about the function of SF (FF, respectively). In practical operational phases (described below), we assume the following:
• In the cell SF, each string is assumed to form a certain *structure* (e.g., structured molecule based on hybridization in terms of H-bonds via minimal energy principle). SF takes as input a string u over V and allows it to filter through if it is in L_{SF} (otherwise, the string u is lost). Then, after building up a structured form $s(u)$ of u, SF removes all parts of structures from $s(u)$ and produces as output the concatenation of all remaining strings. The output v is sent out to the cell FF.
• FF receives as input a string v over V (from SF). A string v filters through if it is in L_{FF} and is sent out to Out. Otherwise, it is sent out to all cells indicated by $Syn(FF)$.
• Each FF_i receives as input a string u over V. Then, a string v filters through if it is in L_{FF_i} and is sent out to all cells indicated by $Syn(FF_i)$. Otherwise, it is lost.
• Filtering applies simultaneously to all strings in the filtering cell.

Note that in the case $SubFil$ is empty in a given Π, an interpretation to Π is simply written as $I = (\{R_1, \cdots, R_k\}, L_{SF}, L_{FF})$.

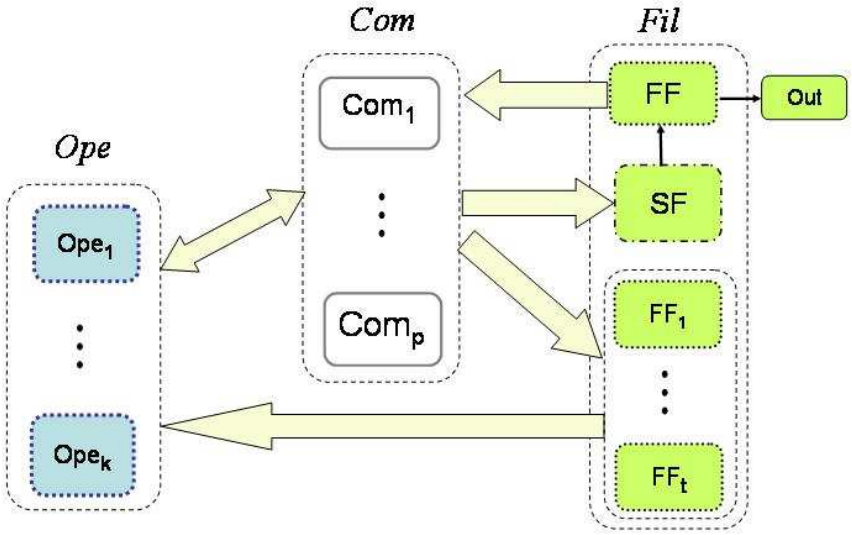


Fig. 1. Modular Structure of Membrane Network in Π

3.3 Transitions and Languages

Given a schema Π and its interpretation I, we now have a membrane system $I(\Pi)$ based on string insertions. In what follows, we define a transition sequence of $I(\Pi)$ and the language associated with $I(\Pi)$.

The $(p+1)$-tuple of languages over V represented by $(L_1, \ldots, L_p, L_{out})$ constitutes a *configuration* of the system, where each L_i represents the set of all strings in the cell Com_i (for all $i = 1, \cdots, p$), and L_{out} is the set of strings presented in the output cell at some time instance.

Let $C_1 = (L_1, \ldots, L_p, L_{out})$ and $C_2 = (L'_1, \ldots, L'_p, L'_{out})$ be two configurations of the system.

We define one transition from C_1 to C_2 in the following steps:

(0) **Pre-checking Step**: For each $\ell = 1, \cdots, t$, let $Syn^{-1}(FF_\ell) = \{\ell_1, \cdots, \ell_q\}$ and consider $L[\ell] = \cup_{i=1}^{q} L_{\ell_i}$. Then, each cell FF_ℓ filters out all strings of $L[\ell]$ that are not in L_{FF_ℓ}, and all strings that have passed through are sent to all the cells indicated by $Syn(FF_\ell)$. (In the case of $t = 0$, this step is skipped.)

(1) **Evolution Step**: For each $j = 1, \cdots, k$, let $\sigma_{j1}, \cdots, \sigma_{js}$ be all the operations given in Ope_j.

Suppose that we apply operations $\sigma_{ji} : \lambda \to u_{ji}$ $(1 \leq i \leq s)$ to a string v which means that each σ_{ji} is applied to v *simultaneously*. Further, when we apply

σ_{ji} to v, the location in v to insert u_{ji} is nondeterministically chosen and the result $\sigma_{ji}(v)$ is considered as the set of *all possible strings* obtained from v by σ_{ji}. (Note that if two or more rules share the same location to insert, then all possible permutations of those rules are considered to apply to the location.) The result of such an application of all operations in Ope_j to v is denoted by $Ope_j(v)$.

Let $L(j) = \cup_{m=1}^{t} L_{j_m}$, where $Syn^{-1}(Ope_j) = \{j_1, \cdots, j_t\}$. Then, the total result performed by Ope_j to $L(j)$ is defined as

$$Ope_j(L(j)) = \cup_{v \in L(j)} Ope_j(v).$$

This result is then sent out to all cells indicated by $Syn(Ope_j)$ simultaneously.

(2) **Filtering Step**: For each $i = 1, \cdots, p$, let $\tilde{L}_i = \cup_{n=1}^{r} Ope_{i_n}(L(i_n))$, where $Syn(Com_i) = \{i_1, \cdots, i_r\}$. Further, let $L_e = \cup_{n=1}^{g} \tilde{L}_{i_n}$, where $Syn^{-1}(SF) = \{i_1, \cdots, i_g\}$.

SF takes as input the set L_e and produces as output a set of strings L_f. (Recall that in the cell SF, each string u is assumed to form a certain structure, and the output of SF is the reduced string by removing structural parts from u in L_{SF}.) Then, SF sends out L_f to FF. (Any element of L_e that was filtered off by SF is assumed to be lost.)

Finally, the cell FF filters out strings of L_f depending upon whether they are in L_{FF} or not. All strings in L_f that passed through FF are sent out to Out, while others are simultaneously sent to Com_i for all $i \in Syn(FF)$ or they are all lost if $Syn(FF) = \emptyset$.

Let L_{ff} be the set of all strings that were filtered off by FF. Then, we define $C_2 = (L'_1, \ldots, L'_p, L'_{out})$ by setting for each $i = 1, \cdots, p$

$$L'_i = \begin{cases} L_{ff}, \text{ if } i \in Syn(FF), \\ \tilde{L}_i, \quad \text{otherwise.} \end{cases}$$

Further, let $L'_{out} = L_{out} \cup L_f(Out)$, where $L_f(Out) = L_f - L_{ff}$ (the set of all stings that have passed through FF and been sent to Out).

Remarks.
(1) Each cell in Com not only provides a buffer for storing intermediate results in the computation process but also *transmits* them to the cells specified by Syn.
(2) The system has a global clock and counts time in such a way that every cell (including communication cells) takes one unit time to perform its task irrespective of the existence of strings in it, while a pre-checking step by a subfilter cell FF_i is assumed to be executed within the unit time for its corresponding operation cells in $Syn(FF_i)$.

Let I be an interpretation of Π. When we have a transition from C_1 to C_2 of $I(\Pi)$, we write $C_1 \Longrightarrow C_2$. A configuration $C_0 = (\{w\}, \emptyset, \cdots, \emptyset)$, where $w \in V^*$, is called the *initial configuration*. A sequence of transitions between configurations of the system $I(\Pi)$, starting from the initial configuration, is called a *computation* of $I(\Pi)$.

Let $\Longrightarrow^*$ be the reflexive and transitive closure of $\Longrightarrow$. For any $n \geq 0$, let $C_0 \Longrightarrow^n C_n = (L_{1,n}, \cdots, L_{p,n}, L_{out}^{(n)})$ be a computation of $I(\Pi)$ with n transitions from C_0. We consider two types of computing models induced from $I(\Pi)$; one is the generating model and the other the accepting model.

[*Generating Model*] In the case of a generating model, we consider all computations whose results are present in the output cell. That is, we define the language generated by $I(\Pi)$ as follows:

$$L_g(I(\Pi)) = \cup_{n \geq 0} L_{out}^{(n)}.$$

[*Accepting Model*] In the case we consider an accepting model, let w be an input string. Then, we assume that if w is recognized, then the result *Yes* or *No* is present in the output cell after a certain number of transitions from $C_0 = (\{w\}, \emptyset, \cdots, \emptyset)$. Thus, we define the language accepted by $I(\Pi)$ as follows:

$$L_a(I(\Pi)) = \{w \in T^* \mid Yes \in L_{out}^{(n)} \text{ for some } n \geq 0\}.$$

For a class of interpretations $\mathcal{I}$ to Π, we denote by $\mathcal{LMS}_x(\Pi, \mathcal{I})$ the family of all languages $L_x(I(\Pi))$ specified by those systems as above, where I is in $\mathcal{I}$. That is,

$$\mathcal{LMS}_x(\Pi, \mathcal{I}) = \{L_x(I(\Pi)) \mid I \in \mathcal{I}\},$$

where x is in $\{a, g\}$.

4 Characterizations by Membrane Schema Π_0

The structure of the membrane computing schema Π introduced in the previous section seems to be general enough to induce a computing device with universal computability power by finding an appropriate interpretation. In what follows, we will show that such a universal computability can be realized by much simpler schemas together with appropriate interpretations of moderately simple filtering cells.

First we consider the following simple membrane computing schema:

$$\Pi_0 = (V, T, Syn, Com, Ope, Fil, i_s, Out),$$

where:

(1) V, T, Ope, i_s and Out are the same as in Π,
(2) $Fil = \{SF, FF\}$ (i.e., $SubFil$ is empty),
(3) $Com = \{Com_1, Com_2\}$,
(4) $Syn = \{Com_1, Ope_i), (Ope_i, Com_2) \mid 1 \leq i \leq k\}$
$\cup\{(FF, Com_1), (Com_2, SF), (SF, FF), (FF, Out)\}$ (see (a) of Figure 2).

We are now in a position to present our first result.

Theorem 4.1 *There exists* $\mathcal{I}_G$ *such that* $RE = \mathcal{LMS}_g(\Pi_0, \mathcal{I}_G)$.

Proof. We prove only the inclusion $\subseteq$. (The opposite inclusion is a consequence of the Turing-Church thesis.)

Let L be any language in RE that is generated by a Chomsky type-0 grammar $G = (N, T, S, P)$. Then, we consider the following interpretation $I_G = (R_G, L_{SF}, L_{FF})$, where

(i) For each $r : u \to v$ in P, construct $R_r = \{\lambda \to vr,\ \lambda \to \overline{u}^R\overline{r}\}$, and let $R_G = \{\ R_r \mid r \in P\}$.

(ii) • L_{SF} is given as following language:

$$L_{mir} = V^*\{\ w\overline{w}^R \mid w \in V^*\}V^* = \{\ xw\overline{w}^R y \ \mid\ x, y, w \in V^*\ \}\ .$$

That is, a string filters through SF iff it is an element in L_{mir}, where $V = N \cup T \cup \{r \mid r \in P\}$. (Recall that, by definition of SF, any string in SF is assumed to form a structure, and we assume the hybridization by involution relation ρ over V. Specifically, SF performs two functions: it only accepts all structures of molecules containing a special hairpin $w\overline{w}$, and then it removes the portion of a hairpin from the structure and send out the rest part of the string to FF(see (b) of Figure 2). The structures rejected by SF are *all lost*.)

• L_{FF} is simply given as T^*, so that only strings in T^* can pass through FF and are sent to the output cell Out. Other strings are all sent to Com_1. (Note that (FF, Com_1) is in Syn of Π_0.)

For any $n \geq 1$, let $C_0 = (\{S\}, \emptyset, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{2,n}, L_{out}^{(n)})$ be a computation with n transitions in $I_G(\Pi_0)$. Suppose that $S \Longrightarrow^{n-1} \alpha \Rightarrow_r \beta$ in G, where $\alpha = xuy$, $\beta = xvy$ and $r : u \to v$ is used. Then, we can show that

(i) α is in $L_{1,(n-1)}$,

(ii) $\beta' = xvru\overline{u}^R\overline{r}y$ is in SF, and

(iii) after filtering by SF, the reduced string of β', i.e., a string $xvy(= \beta)$ is sent to FF.

In FF, if β is not in T^*, then it is sent to Com_1 and, therefore, in $L_{1,n}$, where $C_n = (L_{1,n}, L_{2,n}, L_{out}^{(n)})$. Otherwise, β is sent to $L_{out}^{(n)}$. That is, if β is in $L(G)$, then we have that β is in $L_g(I_G(\Pi_0))$.

Conversely, suppose that for any $n \geq 1$, α is in $L_{1,n}$. Then, there exists $\alpha' = \alpha_1 w\overline{w}^R\alpha_2$ in Com_2 such that $\alpha = \alpha_1\alpha_2$. From the way of constructing R_G, there uniquely exists $r : u \to v$ in P such that $R_r = \{\lambda \to vr, \lambda \to \overline{u}^R\overline{r}\}$ and $w = ru$. (No other $R_{r'}(r' \neq r)$ can make a substring $w\overline{w}^R$ by insertion operations because of the uniqueness of r.)

Therefore, we can write $\alpha' = \alpha_1 ru\overline{u}^R\overline{r}\alpha_2$ for some α_1, α_2. Then, there must exist α_1' such that $\alpha_1 = \alpha_1' v$ because of the rule $\lambda \to vr$. Hence, we have that $\alpha' = \alpha_1' vru\overline{u}^R\overline{r}\alpha_2$ from which we can derive that $\alpha_1' u\alpha_2$ is in $L_{1,(n-1)}$. Thus, there exists a derivation: $\alpha_1' u\alpha_2 \Longrightarrow_r \alpha_1' v\alpha_2 = \alpha$ in G. By iteratively applying the above argument, we eventually conclude that there exists a derivation $S \Longrightarrow^n \alpha$ in G.

Taking $L_{1,0} = \{S\}$ into consideration, it holds that for any $n \geq 0$, $L_{1,n} = \{\alpha \mid S \Longrightarrow^n \alpha \text{ in } G\}$. If α is in $L_g(I_G(\Pi_0))$, then it is also in $L(G)$. Thus, we have that $L(G) = L_a(I_G(\Pi_0))$, which completes the proof. □

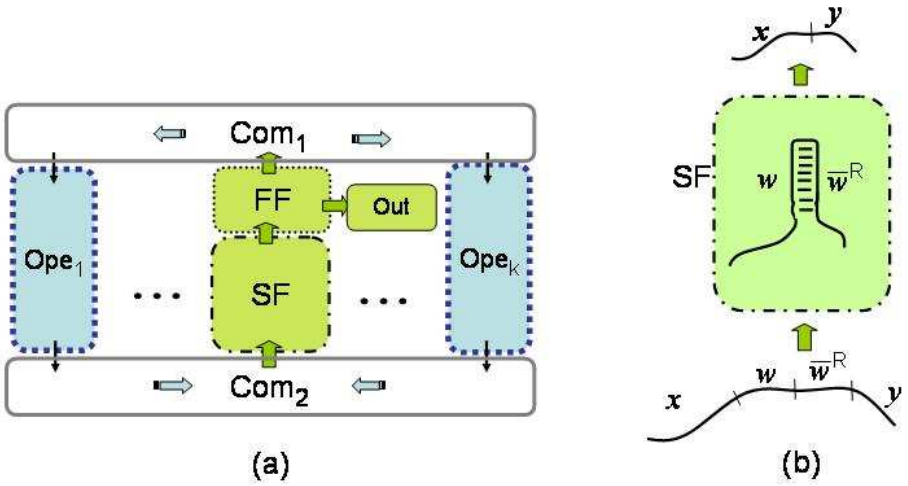


Fig. 2. Membrane Computing Schema: Π_0

Theorem 4.2 *There exists $\mathcal{I}_M$ such that $RE = \mathcal{LMS}_a(\Pi_0, \mathcal{I}_M)$.*

Proof. We use the same strategy as in Theorem 4.1, but start with a (nondeterministic) Turing machine M. That is, let L be any language in RE accepted by $M = (Q, T, U, \delta, p_0, B, F)$, where $B(\notin U)$ is a blank symbol. (Without loss of generality, we may assume that M immediately stops as soon as it enters into a final state of F.) An instantaneous description (ID) of M is represented by a string $xpay$ in U^*QU^*, where xay is the tape content ($x, y \in U^*$, $a \in U$) and M is in the state $p(\in Q)$ and the tape head is on a.

Given an input $w(\in T^*)$, M starts computing w from the state p_0, which is represented by an ID: p_0w. (We assume that the tape content has the left-boundary (the leftmost of w) and no right-boundary where blank symbols B are initially filled.) In general, suppose that a transition rule $(p, a) \to (q, c, i) \in \delta$ is applied to an ID $xbpay$. Then, we have a transition between IDs of M:

- $xbpay \Longrightarrow xbcqy$ (if $i = R$ and $a \neq B$),
- $xbpay \Longrightarrow xqbcy$ (if $i = L$ and $a \neq B$),
- $xbpa \Longrightarrow xbcq$ (if $i = R$, $a = B$ and $y = \lambda$),

- $xbpa \Longrightarrow xqbc$ (if $i = L$, $a = B$ and $y = \lambda$).

Thus, in each case one can consider a rewriting rule : for example, a rule $pa \to cq$ for the case (i). Let P_M be the set of rewriting rules obtained from δ in this manner. We define an interpretation $I_M = (R_M, L_{SF}, L_{FF})$ as follows:

(i) For each $r : u \to v$ in P_M, construct $R_r = \{\lambda \to vr,\ \lambda \to \overline{u}^R\overline{r}\}$, and let $R_M = \{\ R_r \mid r \in P_M\}$.
(ii) L_{SF} is the same as the one in I_G.
(iii) L_{FF} is given as V^*FV^*, where $V = U \cup Q \cup \{B\} \cup \{r \mid r \in P_M\}$.

Let w be any string in T^* and $n \geq 0$. Then, from the way of constructing I_M together with discussion above, it is easily seen that $p_0w \Longrightarrow^n xqy$ for some $q \in F$, $x, y \in U^*$ iff there exists $Yes \in L_{out}^{(n)}$ such that $C_0 = (\{w\}, \emptyset, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L_{2,n}, L_{out}^{(n)})$. Thus, we have that $L(M) = L_a(I_M(\Pi_0))$, which completes the proof. □

5 Further Results by Some Variants of Membrane Schema

We give now another membrane computing schema $\Pi_{1,t}$ which is a variant of Π_0 and also able to induce a family of computing devices $I(\Pi_{1,t})$ (with an appropriate interpretation I) that can characterize RE.

The membrane computing schema $\Pi_{1,t}$ is given as follows:

$$\Pi_{1,t} = (V, T, Syn, Com, Ope, Fil, i_s, Out),$$

where:

(1) V, T, Ope, i_s and Out are the same as in Π_0.
(2) $Com = \{Com_1\}$.
(3) $Fil = \{SF, FF\} \cup SubFil$, where $SubFil = \{FF_i \mid 1 \leq \forall i \leq t\}$ or $= \emptyset$ $(t = 0)$.
(4) $Syn = \{(Com_1, FF_i), (FF_i, Ope_i) \mid 1 \leq i \leq t\} \cup \{(Com_1, Ope_j) \mid t{+}1 \leq j \leq k\}$
$\cup (Ope_i, Com_1) \mid 1 \leq \forall i \leq k\} \cup \{(Com_1, SF), (SF, FF), (FF, Out)\}$.
(aee (a) of Figure 3).

Note: In (3) and (4) above, t can take any integer between 0 and k, and when $t = 0$, it means the corresponding set is empty.

The difference between $\Pi_{1,t}$ and Π_0 is that $\Pi_{1,t}$ has only one Com_1, while several Ope_is possibly require subfiltering cells FF_i for prechecking of strings.

Notation. We denote by $\Pi_{1.5}$ a schema $\Pi_{1,t}$ where $1 \leq t < k$, and write Π_1 for a schema $\Pi_{1,k}$(i.e., t coincides with k).

Theorem 5.1 *There exists $\mathcal{I}_{G_m}$ such that $RE = \mathcal{LMS}_g(\Pi_{1.5}, \mathcal{I}_{G_m})$.*

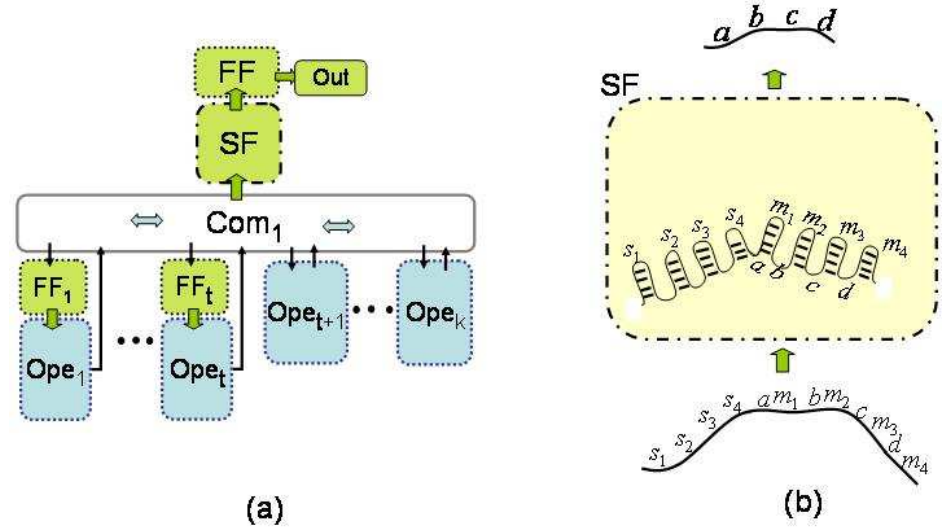


Fig. 3. Membrane Computing Schema $\Pi_{1.5}$

Proof sketch. We use a similar argument to the one in the proof of Theorem 4.1 and start with a matrix grammar. That is, let L be any language in RE generated by a matrix grammar $G_m = (N, T, S, M, F)$ with appearance checking, where $N = N_1 \cup N_2 \cup \{S, \#\}$, and we may assume that G is in the binary normal form.

We consider the following interpretation $I_{G_m} = (R_{G_m}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \leq i \leq t\})$, where

(i) (1) Let k be the cardinality of M and t be the number of appearance checking matrix rules in M. For each appearance checking rule $m_i : (X \to Y, A \to \#)$ $(1 \leq i \leq t)$, construct $R_{m_i} = \{\lambda \to Y s_{m_i},\ \lambda \to \overline{X}\overline{s}_{m_i}\}$.
(2) For other rules $m_j : (X \to Y, A \to x)$ in M (where $Y \in N_1 \cup \{\lambda\}, x \in T \cup N_2 \cup N_2^2 \cup \{\lambda\};\ t+1 \leq j \leq k$), construct $R_{m_j} = \{\lambda \to Y s_{m_j},\ \lambda \to \overline{X}\overline{s}_{m_j}, \lambda \to x r_{m_j}, \lambda \to \overline{A}\overline{r}_{m_j}\}$. Then, let $R_{G_m} = \{R_{m_1}, \cdots, R_{m_k}\}$.

(ii) • L_{SF} is given as the regular language $L_{mat} = L_s L_m$, where

$$L_s = \{s_m X \overline{X} \overline{s}_m \mid m : (X \to Y, A \to \#) \in M\}^*, \text{ and}$$
$$L_m = (T \cup \{r_m A \overline{A} \overline{r}_m \mid m : (X \to Y, A \to y) \in M\})^*.$$

• L_{FF_i} is given as follows: Let t be the number of appearance checking matrix rules in M. For each appearance checking rule $m_i : (X \to Y, A \to \#)$ $(1 \leq i \leq t)$, consider $L_{m_i} = (V \cup \overline{V})^* - (V \cup \overline{V})^* \{A\} (V \cup \overline{V})^*$, where $V = T \cup N \cup \{s_m, r_m \mid m \in M\}$. Then, L_{FF_i} is given as L_{m_i}. (Thus, FF_i performs in such a way that it allows only strings in L_{m_i} to pass through and send them to the cell Ope_i. Other strings are all lost.)

• Finally, L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent to Out.

Let $C_0 = (\{XA\}, \emptyset)$, where $(S \to XA) \in M$, and consider a transition sequence $C_0 \Longrightarrow^n C_n = (L_{1,n}, L_{out}^{(n)})$. Then, for any $\alpha \in (N \cup T)^*$ and $n \geq 0$, it holds that $\alpha \in L_{1,n}$ iff $XA \Longrightarrow^n \alpha$ in G_m. Thus, we have $L(G_m) = L_g(I_{G_m}(\Pi_{1.5}))$. □

Theorem 5.2 *There exists $\mathcal{I}_{G_r}$ such that $RE = \mathcal{LMS}_g(\Pi_1, \mathcal{I}_{G_r})$.*

Proof sketch. We use the same argument as the one in the proof of Theorem 5.1, but we start with a random context grammar $G_r = (N, T, S, P)$ generating an arbitrary recursively enumerable language L. (See Preliminary section.) Then, consider the following interpretation $I_{G_r} = (R_{G_r}, L_{SF}, L_{FF}, \{L_{FF_i} \mid 1 \leq i \leq k\})$, where

(i) Let k be the cardinality of P. For each rule $r_i : (A \to x, Q, R)$ $(1 \leq i \leq k)$, construct $R_{r_i} = \{\lambda \to xr_i,\ \lambda \to \overline{A}\overline{r_i}\}$. Then, let $R_{G_{r_i}} = \{R_{r_i} \mid r_i \in P\}$.

(ii) • L_{SF} is defined by the regular language L_m (in L_{SF} for I_{G_m}).
• L_{FF_i} is given as follows: For each rule $r_i : (A \to x, Q, R)$, if $A \in Q$, then let $L_{r_i} = (V \cup \overline{V})^*\{A\}(V \cup \overline{V})^*\{A\}(V \cup \overline{V})^* \cap \cap_{X \in Q - \{A\}}(V \cup \overline{V})^*\{X\}(V \cup \overline{V})^* \cap \cap_{X \in R}((V \cup \overline{V})^* - (V \cup \overline{V})^*\{X\}(V \cup \overline{V})^*)$. Otherwise (i.e., $A \notin Q$), let $L_{r_i} = \cap_{X \in Q}(V \cup \overline{V})^*\{X\}(V \cup \overline{V})^* \cap \cap_{X \in R}((V \cup \overline{V})^* - (V \cup \overline{V})^*\{X\}(V \cup \overline{V})^*)$, where $V = N \cup T \cup \{r \mid r \in P\}$. Then, L_{FF_i} is defined by L_{r_i}. (That is, each FF_i performs in such a way that it allows only strings in L_{r_i} to pass through and send them to the cell Ope_i. Other strings are all lost.)
• Finally, L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent out to Out.

Let $C_0 = (\{S\}, \emptyset)$, where S is the starting symbol of G_r, and consider a transition sequence $C_0 \Longrightarrow^n C_n = (L_{1,n}, L_{out}^{(n)})$. Then, for any $\alpha \in (N \cup T)^*$ and $n \geq 0$, it holds that $\alpha \in L_{1,n}$ iff $XA \Longrightarrow^n \alpha$ in G_r. Thus, we have that $L(G_r) = L_g(I_{G_r}(\Pi_1))$. □

We give yet another membrane computing schema Π_2 which is simpler than Π_0 but still able to provide the universal computability of those models induced from the schema with appropriate interpretations, but at the price of increasing the structural complexity in the filtering function SF.

The membrane computing schema Π_2 is given as follows:

$$\Pi_2 = (V, T, Syn, Com, Ope, Fil, i_s, Out),$$

where:

(1) V, T, Ope, Fil, i_s and Out are the same as in Π_0.
(2) $Com = \{Com_1\}$.
(3) $Syn = \{(Com_1, Ope_i),\ (Ope_i, Com_1) \mid 1 \leq i \leq k\}$
$\cup\{(Com_1, SF), (SF, FF), (FF, Out)\}$ (see (a) of Figure 4).

From this simpler schema Π_2, we can induce a family of computing devices $I(\Pi_2)$ (with an appropriate interpretation I) that can characterize RE. (Note that Π_2 is nothing but a schema $\Pi_{1,0}$.)

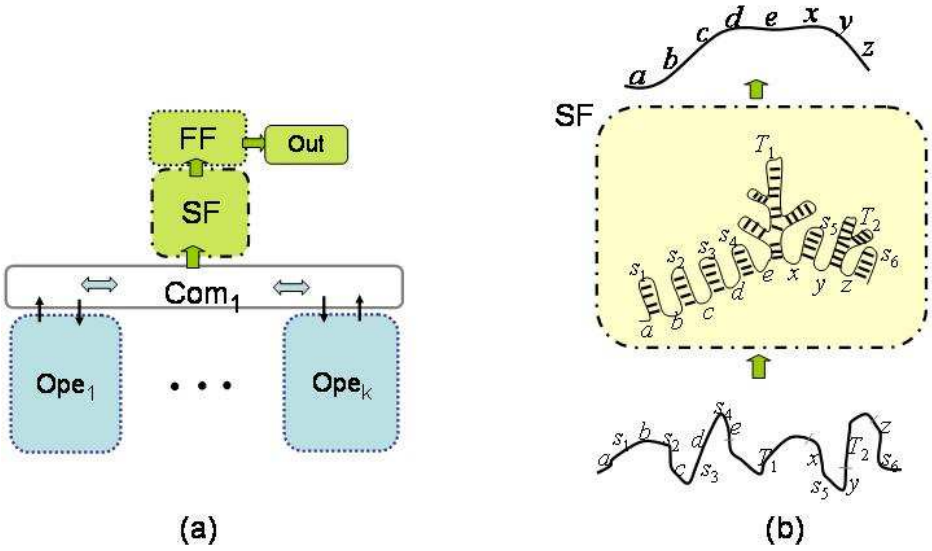


Fig. 4. Membrane Computing Schema: Π_2

Theorem 5.3 *There exists $\mathcal{I}'_G$ such that $RE = \mathcal{LMS}_g(\Pi_2, \mathcal{I}'_G)$.*

Proof sketch. Let L be any language in RE that is generated by a Chomsky type-0 grammar $G = (V, T, S, P)$. Then, consider the following interpretation $I'_G = (\{R_G\}, L_{SF}, L_{FF})$, where

(i) For each $r : u \to v$ in P, construct $R_r = \{\lambda \to vr,\ \lambda \to \overline{u}^R\overline{r}\}$, and let $R_G = \cup_{r\in P} R_r$.
(ii) L_{SF} adopts the Dyck language D_n, where $n = |N \cup T \cup \{r \mid r \in P\}|$.
(iii) L_{FF} is given as T^*, so that only strings in T^* can pass through FF and are sent out to *Out*.

Consider a transition sequence $C_0 = (\{S\}, \emptyset) \Longrightarrow^n C_n = (L_{1,n}, L^{(n)}_{out})$. Then, for any $\alpha \in (N \cup T)^*$ and $n \geq 0$, it holds that $\alpha \in L_{1,n}$ iff $S \Longrightarrow^n \alpha$ in G. Thus, we have that $L(G) = L_g(I'_G(\Pi_2))$. (The proof is based on the fact that each recursively enumerable language L can be represented in the form $L = h(L' \cap D_k))$, where L' is an insertion language, h is a projection and D_k is a Dyck language. (Theorem 3.1 in [13]). In order to understand the idea of the proof, it would be helpful to

note that R_G in Com_1 generates the insertion language L', while a pair of SF and FF plays the same role as a pair of D_k and h.) □

6 Concluding Remarks

In this paper we have introduced the notion of a membrane computing schema and showed that several known computing models with the universal computability can be reformulated in a uniform manner in terms of the framework of the schema together with its interpretation. A similar idea in the context of grammar schema has been proposed and discussed in [1, 5] to prove the computational completeness of new type of P systems based on the framework of the random context grammars for both string and multiset languages. (Note that the definition of random context in those papers is not on a string to be rewritten but on the applicability of rules to rewrite, different from the standard notion.) As for the communication by sending objects and the use of filtering function, there are several papers that have been devoted to studying the computational powers of communicating distributed H systems (e.g., [3, 11]), and of the hybrid networks of evolutionary processors (e.g., [7, 8]).

Table 1 summarizes the results we have obtained. From the table, one can have a unified view of a variety of computing models based on *string rewriting*. For example, it is seen that there exists a trade-off between the complexity of network structure in the schema and the complexity of the filtering SF.

More specifically, for new terminologies, L is a *star language* iff $L = F^*$ for some finite set F. Further, L is an *occurrence checking language* iff $L = V^*FV^*$ for some finite set F. Then, it should be noted that
(i) L_G is a finite union of occurrence checking languages,
(ii) L_s and L_m are star languages,
(iii) L_{r_i} is a finite intersection of occurrence checking languages and their complements,
(iv) L_{m_i} is the complement of an occurrence checking language.

Since Π_0 (or Π_1) is more complex than Π_2, one may see a trade-off between the complexity of the schema and that of SF, telling that L_G for single (or L_{mat} for multiple) hairpin checking is simpler than D_k for nested hairpin checking. This kind of trade-off can also be seen in complexity between a series of schemas $(\Pi_1, \Pi_{1.5}, \Pi_2)$ and the corresponding SFs (L_m, L_{mat}, D_k).

It should be remarked that if we start with a programmed context-free grammar G_{pr} ([4]), then we have the result that $RE = \mathcal{LMS}_g(\Pi_1, \mathcal{I}_{G_{pr}})$, where an interpretation $I_{G_{pr}}$ consists of L_m (for SF), T^* (for FF) and L_{m_i} (for FF_i), which suggests that programmed context-free grammars can be regarded as hybrid systems between matrix grammars with appearance checking and random context grammars.

Table 1

	Schema	SF	FF	SubFil
Chomsky type-0 Grammar	Π_0	$L_G(\in RG)$ or $L_{mir}(\in LIN)$	T^*	(N.A.)
Turing Machine	Π_0	$L_M(\in RG)$ or $L_{mir}(\in LIN)$	V^*FV^*	(N.A.)
Random Context Grammar	Π_1	$L_m(\in RG)$	T^*	$L_{r_i}(\in RG)$
Matrix$_{ac}$ Grammar	$\Pi_{1.5}$	$L_{mat} = L_s L_m(\in RG)$	T^*	$L_{m_i}(\in RG)$
Chomsky type-0 Grammar	Π_2	$D_k(\in CF)$	T^*	(N.A.)

In this paper we have just made the first step in the new direction towards understanding and characterizing the nature of the Turing computability from the novel viewpoint of *modularity* in the membrane computing schema. There remain many questions for the future works:

- It would be the most interesting to study the relation between the complexity of the language classes and that of SF within a given schema. For instance, we can show that within the schema Π_2, CF can be characterized by star (regular) languages for SF.
- Instead of insertion operations we adopted in this paper, what kind of operations can be considered for the unique operation in the cells *Ope*? What kind of different landscape of the computing mechanisms can be seen from the new schema?

References

1. M. Cavaliere, R. Freund, M. Oswald, D. Sburlan: Multiset random context grammars, checkers, and transducers. In *Fourth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), Fenix Editora, Sevilla, 2006, Vol. 1, 113–131.
2. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In *Fourth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), Fenix Editora, Sevilla, 2006, Vol. 1, 169–193.
3. E. Csuhaj-Varju, L. Kari, Gh.Păun: Test tube distributed systems based on splicing. *Computers and AI*, 15, 2-3 (1996), 211–232.
4. J. Dassow, Gh. Păun: *Regulated Rewriting in Formal Language Theory*. Springer-Verlag, Berlin, 1989.
5. R. Freund, M. Oswald: Modeling grammar systems by tissue P systems working in the sequential mode. In *Proc. of Grammar Systems Workshop*, Budapest, 2004.
6. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.

7. M. Margenstern, V. Mitrana, M.J. Pérez-Jiménez: Accepting hybrid networks of evolutionary processors. *Lecture Notes in Computer Science*, 3384, Springer-Verlag, Berlin, 2005, 235–246.
8. C. Martin-Vide, V. Mitrana, M.J. Pérez-Jiménez, F. Sancho-Caparrini: Hybrid networks of evolutionary processors. In *Proc. of GECCO, Lecture Notes in Computer Science*, 2723, Springer-Verlag, Berlin, 2003, 401–412.
9. C. Martin-Vide, Gh. Păun, A. Salomaa: Characterizations of recursively enumerable languages by means of insertion grammars. *Theoretical Computer Science*, 205 (1998), 195–205.
10. C. Martin-Vide, Gh. Păun, J. Pazos, A. Rodriguez-Paton: Tissue P systems, *Theoretical Computer Science*, 296 (2003), 295–326.
11. Gh. Păun: Distributed architectures in DNA Computing based on splicing: Limiting the size of components. In *Proc. Conf. Unconventional Models of Computation* (C.S. Calude, J. Casti, M.J. Dinneen, eds.), Springer-Verlag, Berlin, 1998, 323–335.
12. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
13. Gh. Păun, M.J. Pérez-Jiménez, T. Yokomori: Representations and characterizations of languages in Chomsky hierarchy by means of insertion-deletion systems. Submitted. 2007.
14. Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer-Verlag, Berlin, 1998.
15. Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graph of restricted forms. *Publicationes Mathematicae Debrecen*, 60 (2002), 635–660.
16. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer-Verlag, Berlin, 1997.

A Software Tool for Dealing with Spiking Neural P Systems

Daniel Ramírez-Martínez, Miguel A. Gutiérrez-Naranjo

Research Group on Natural Computing
University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
thebluebishop@gmail.com, magutier@us.es

Summary. Software simulators for P system are nowadays the main tool to carry out experiments in the field of Membrane Computing. Although the simulation of a P system is a quite complex task, current simulators have been successfully used for pedagogical purposes and also as assistant tools for researchers. In this paper we present a first software tool for dealing with Spiking Neural P Systems. This tool outputs the transition diagram of a given system in a step-by-step mode. The code is modular and flexible enough to be adapted for further research tasks.

1 Introduction

Membrane Computing was introduced by Gh. Păun in [11], starting from the assumption that the processes taking place in the compartmental structure of a living cell can be interpreted as computations[1]. Since then, a large number of variants and subvariants have been considered, concerning both the syntax and the semantics of the model. The devices of this model are called generically *P systems*.

Roughly speaking, a P system consists of a membrane structure, in the compartments of which one places multisets of objects which evolve according to given rules. The evolution of the rules is usually performed in a synchronous non-deterministic maximally parallel manner, but this can vary depending on the model. Let us recall here three of the main variants of P systems: Cell-like P systems, Tissue-like P Systems, and Spiking Neural P Systems.

In the *cell-like* model of P systems, membranes are hierarchically arranged in a tree-like structure (see [11]). The biological inspiration is the morphology of cell, where small vesicles are surrounded by larger ones. This biological structure can be abstracted into a tree-like graph, where the root of the graph represents the

[1] The foundations of Membrane Computing can be found at [12] and updated bibliography at [16].

skin of the cell (the outermost membrane), the leaves represent membranes that do not contain other ones (elementary membranes) and two nodes in the graph are connected if they represent two membranes such that one of them contains the other one.

In the *tissue P systems* ([9, 10]) we consider a general graph instead of a tree-like membrane structure, where the nodes can be considered as processors and the edges as connections among them in order to share information. This model has two biological inspirations (see [10]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules: Symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite direction.

Spiking neural P systems (SN P systems, for short) were introduced in [6]. The aim of this new model was to incorporate in membrane computing ideas specific to spiking neurons; the intuitive goal was to have (1) a tissue-like P system with (2) only one (type of) object(s) in the cells, the spike, with (3) specific rules for evolving populations of spikes, and (4) making use of the time as a support of information[2].

Irrespectively of the selected approach, it is usually a complex task to predict or to guess how a P system will behave. Moreover, as there do not exist, up to now, implementations in laboratories (neither *in vitro* nor *in vivo* nor in any electronic media), it seems natural to look for software tools that can be used as assistants that are able to simulate computations of P systems.

The first software simulator for P systems appeared in 2000. It was written by Mihaela Maliţa [8] in LPA-Prolog and, since then, more than ten software simulators have been presented (see [7] and references therein). Their common purpose is the better understanding of the computational process of P systems, for pedagogical purposes as well as assistant for researchers. Among these simulators, we can find simulators for transition P system (as Maliţa's one) or simulators able to deal with P Systems with active membranes (as Ciobanu and Paraschiv's simulator [3]). Some of them explore new architectures looking for increasing the speed of the computation like Ciobanu and Wenyuan's simulator [5] based on parallel architecture. Others put the stress on a simulation closer to biological laws, as the simulator by the *Group for Models of Natural Computing* [17] in Verona, based on the implementation of the metabolic algorithm introduced in [1].

In this paper we present a new tool which is born with the hope of becoming a tool for the creativity in Membrane Computing. It deals with SN P systems, and to the best of our knowledge, so far no other software tool has been presented for dealing with such systems. Our simulator shares the common purpose of the

[2] In the next section we will give a brief introduction of some concepts related to SN P Systems, a detailed description can be found at [13] and the references therein.

others simulators: the understanding of the computational process of P systems and becoming an assistant for researchers. It receives the description of an SN P System and outputs its transition diagram with an user friendly interface.

The paper is organized as follows. First we recall some definitions related to SN P systems. Then, the simulator is presented and some features of its implementation are given. Section 4 provides an example of how the simulator works and finally, in the last section some remarks and future work lines are presented.

2 The Model

An SN P system consists of a set of neurons placed in the nodes of a directed graph and sending signals (called *spikes*) along the arcs of the graph (called *synapses*). The objects evolve according to a set of rules (called *spiking rules*). The idea is that a neuron containing a certain amount of spikes can consume some of them and produce other ones. The produced spikes are sent (maybe with a delay of some steps) to all neurons to which a synapse exists outgoing from the neuron where the rule was applied. There also are forgetting rules. By the application of these rules no spikes are sent, simply some spikes are removed from the neuron where the rule was applied. A global clock is assumed and in each time unit each neuron which can use a rule should do it, but only (at most) one rule is used in each neuron. One of the neurons is considered to be the output neuron, and its spikes are also sent to the environment.

Formally, an *SN P system* of degree $m \geq 1$, is a construct of the form

$$Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out),$$

where:

- $O = \{a\}$ is the alphabet (the object a is called *spike*);
- $\sigma_1, \ldots, \sigma_m$ are neurons, of the form $\sigma_i = (n_i, R_i)$ with $i \in \{1, \ldots, m\}$ where:
  - $n_i \geq 0$ is the initial number of spikes contained by the neuron;
  - R_i is a finite set of rules of the form:

$$E/a^c \rightarrow a^p; d$$

    where E is a regular expression with a the only symbol used, $c \geq 1$ and $p, d \geq 0$, with $c \geq p$; if $p = 0$, then $d = 0$ too.
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, ..., m\}$ with $(i, i) \notin syn$ for $i \in \{1, \ldots, m\}$ (synapses);
- $out \in \{1, 2, \ldots, m\}$ is the output neuron. Notice that only one neuron can send spikes to the environment.

For the sake of simplicity, if $p = d = 0$, the rule is written as $E/a^c \rightarrow \lambda$ instead of $E/a^c \rightarrow a^0; 0$. This type of rules is called *forgetting rules*. If $L(E) = \{a^c\}$ then the rules are written in the simplified form $a^c \rightarrow a^p; d$ and $a^c \rightarrow \lambda$.

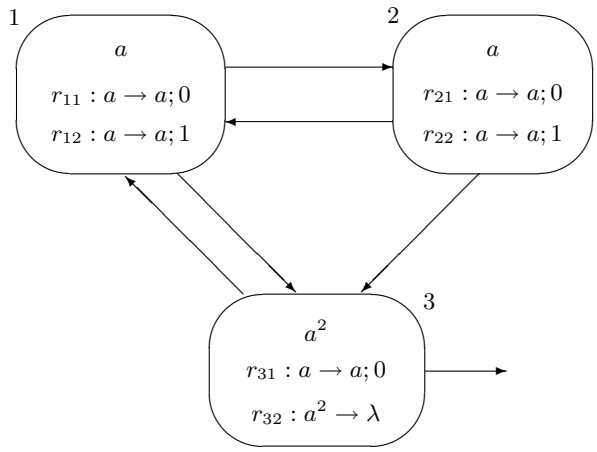


Fig. 1. An example of SN P system

With respect to the application of the rules, if the neuron σ_i contains k spikes, $a^k \in L(E)$ and $k \geq c$, then the rule $E/a^c \to a^p; d \in R_i$ (with $p \geq 1$) can be applied; applying it means that c spikes are consumed, only $k - c$ remain in the neuron and it produces p spikes after d time units. If $d = 0$, then the spikes are emitted immediately, otherwise the spikes are emitted after d steps. In the case $d \geq 1$, if the rule is used in step t, then in steps $t, t + 1, t + 2, \ldots, t + d - 1$ the neuron is closed, and it cannot receive new spikes (if a neuron has a synapse to a closed neuron and sends spikes along it, then the spikes are lost). In step $t + d$, the neuron spikes and becomes again open, hence can receive spikes. The p spikes emitted by a neuron σ_i are replicated and they go to all neurons σ_j such that $(i, j) \in syn$ (each σ_j receives p spikes). If the rule is a forgetting one, then no spike is emitted and the neuron cannot be closed.

In each time unit, in each neuron which can use a rule, a rule must be used non-deterministically chosen if several rules are applicable. Note that each neuron processes sequentially its spikes, using only one rule in each time unit. During the computation, a configuration is described by both the number of spikes present in each neuron and by the number of steps to count down until it becomes open (this number is zero if the neuron is open, for example, in the initial configuration). Any sequence of transitions starting in the initial configuration is called a computation. A computation halts if it reaches a configuration where all neurons are open and no rule can be used.

The graphical representation of an SN P system is rather intuitive: the neurons are represented by membranes, placed in the nodes of a directed graph whose arrows represent the synapses. Figure 1 shows an example of the graphical representation of an SN P System taken from [2].

In the graphical representation of our software tool, the environment is also represented by a membrane and an arrow exits from the output neuron, pointing

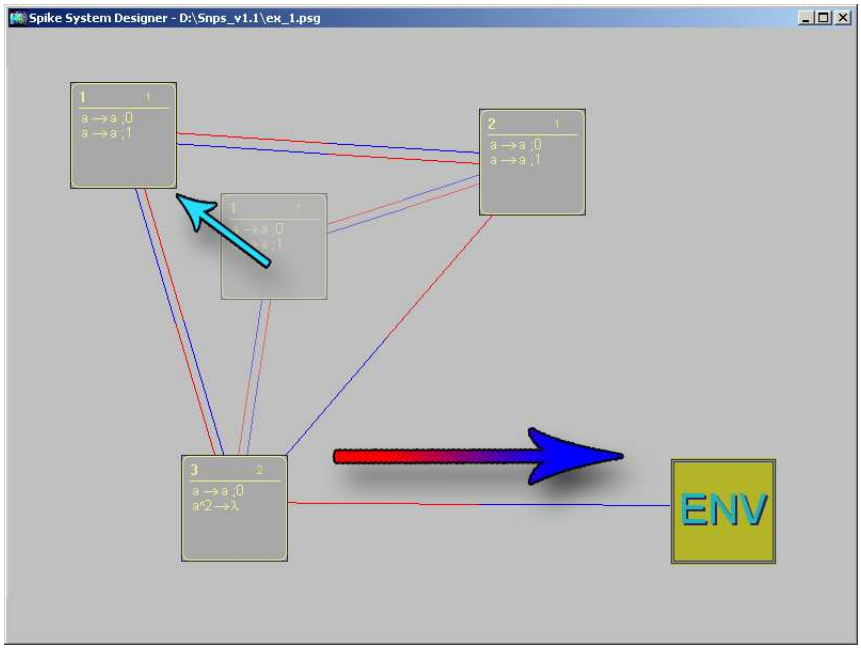


Fig. 2. Screen capture of the designer interface

to this membrane; in each neuron we specify the rules and the spikes present in the initial configuration. Figure 2 shows a screen capture of the designer module of our software by showing our graphical representation of the example.

The transition diagram of an SN P System is also a graph where the nodes are configurations and an arrow is drawn between two nodes/configurations. We draw an arrow if a direct transition is possible between them. Our software also provides labels to the arcs with information about the rules used in the transition.

3 The Simulator

The software tool[3] for simulating the evolution of an SN P system is composed by the integration of three modules:

- A Graphical User Interface (GUI) which allows an easy interface to users, decoupling experimentation and simulation from programming. This module is written in XBase++ [15], which is an object oriented language to program database oriented graphic applications.

[3] The simulator is available at [16].

```
rule(1-1,a \to a;0).          synapses([1-2,2-1,1-3,3-1,2-3,3-env]).
rule(1-2,a \to a;1).
rule(2-1,a \to a;0).          initial([1/0,1/0,2/0]).
rule(2-2,a \to a;1).
rule(3-1,a \to a;0).
rule(3-2,a^2 \to \lambda).
```

Fig. 3. Input file

- A second module, written in SWI-Prolog [14] which is the inference engine. It takes the initial configuration, the synapses and the rules and produces the transition diagram in text mode.
- A graphical designer tool, which allows the user to draw neurons, synapses and to associate spikes and rules with them by using a friendly interface.

Next we briefly describes some features of the software.

3.1 The input data

The basic way of providing the data to the simulator is by a plain text file with the information stored in an appropriate way. Figure 3 shows the input file of the example depicted in Figure 1.

The syntax is quite intuitive. The file consists of a set of literals with the predicate symbols `rule`, `synapses` and `initial`.

- Rules are of the form `rule(N-X,Latex)`, where `N` is the label of the neuron, `X` is the ordinal of the rule and `Latex` is just the rule written in Latex. The idea is that the researcher preparing a report on SN P systems can move the set of rules from the report to the simulator and backwards with the minimal changes. In the current version, we accept seven syntactically different types of rules:
  (1) `a^s/a^c \to a^p;d`
  (2) `a^s/a^c \to a;d`
  (3) `a^c \to a^p;d`
  (4) `a^c \to \lambda`
  (5) `a \to a;d`
  (6) `a^c \to a;d`
  (7) `a \to \lambda`
- `synapses` has as argument a list of pairs of neurons `Start-End`. Notice that the environment is considered as a special case of neuron without rules and labeled by `env`.
- `initial` stores the initial configuration. We always consider the neurons labeled with numbers $1, 2, 3, \ldots$ and we do not make explicit the label of the neuron in the configuration. In this way, a configuration is a list of pairs `A/B` where

the i-th pair `A/B` denotes the number of spikes `(A)` and the number of steps to became open `(B)` of the neuron with label i.

The simulator also provides a graphical interface which allows the user to design an SN P system, to store it and build its transition diagram. The SN P system can be also be exported form the graphical representation to a file with a text mode representation as described above.

3.2 The inference engine

After the input data have been provided (via the graphical interface or a plain text file), the generation of the transition diagram starts. The basic data structure consists of three lists which will be called `ExpandedNodes`, `NonExpandedNodes` and `Arcs`. We consider that a `Node` of the transition graph, i.e., a *configuration* of the SN P system has been *expanded* when all the configurations that can be reached from the `Node` in one step have been computed.

At the beginning, `NonExpandedNodes` contains the initial configuration and `ExpandedNodes` and `Arcs` are empty lists. The main procedure is quite natural. We take a `Node` from `NonExpandedNodes` and compute all the configurations that can be reached from the `Node` in one step. We will denote by `NewNodes` the list of these configurations. The `Node` is added to `ExpandedNodes` and all the elements of `NewNodes` that do not belong to `ExpandedNodes` are added to `NonExpandedNodes`. Analogously, the labelled arcs with the information of the rules that produce the reachable configurations are also stored in `Arcs`. The process ends when the list `NonExpandedNodes` is empty.

In the current version, the unique test for finishing the process is to check if `NonExpandedNodes` is empty. This means that the simulator is not able to deal with large transition diagrams, since the memory of the computer can be filled before the process ends.

The list `NewNodes` consists of all the configurations that can be reached from the `Node` in one step. In order to find all these configurations, for each neuron we check the set of *applicable* rules of the neuron according to the set of spikes and if the neuron is closed or open. To check if a general rule $E/a^c \rightarrow a^p; d \in R_i$ is applicable involves to know if an expression of type a^k belongs to $L(E)$. In the current version we only consider a weak version of applicability that can be extended in future versions of the simulator.

When the set of applicable rules have been found for each neuron, we consider the Cartesian product of these sets in order to get all the possibilities of reaching a new configuration. Each possibility gives us an arc of the transition diagram with a label composed with the following items:

- If the `X`-th rule of the neuron `N` is applied, then `N-X` is added to the label.
- If the neuron `N` is closed and therefore, no rule is applied, then `N-s` is added to the label.
- Finally, if the neuron `N` is open, but no rule can be applied then we add `N-0` to the label.

Fig. 4. Main window of SNPS-GUI

After the set of arcs with its corresponding label is fixed, the application of the rules starts. We have a set of rules (one rule at most for each neuron) associated with each arc that produces a new configuration. In order to apply these rules and build the new configuration we have to consider:

- The spikes sent out from all the neurons.
- For each neuron, all the incoming spikes. Some of them can have been sent some steps before with delay and arrive in the current step.

We need to keep in memory the spikes sent with delay in the previous steps, so the internal representation in the software of a neuron is more complicated than the list of pairs *Spikes/Delay* of the standard representation of the configuration.

The set of all the configurations reachable are determined by the set of arcs. When all these configurations have been generated, the list `NewNodes` is finished and the main procedure goes on.

3.3 The GUI for the inference engine

The transition diagram of an SN P system is usually a large graph and it can be useful to have a tool that shows the evolution in time of the diagram. The GUI module provides a set of images (PS and JPEG files) representing the steps on the

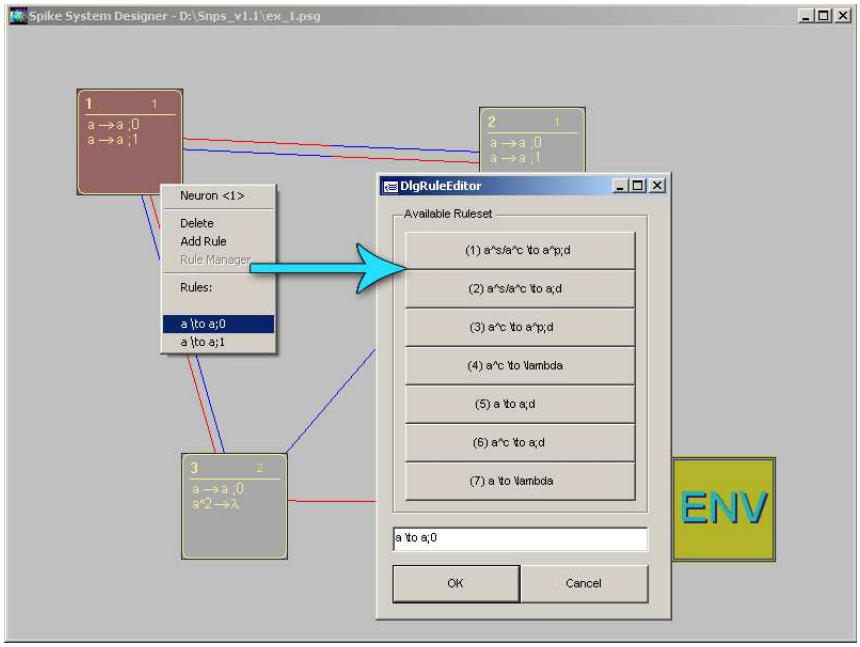


Fig. 5. Rule editor window

evolution of the system. At each step, the picture shows the set of reachable configurations till the moment together with the labeled arcs. This GUI also includes a system designer for the SN P systems.

Basics of SNPS-GUI

This software runs on Windows SO machines, so it has a typical windows 800x600 application look. To complete a simulation the user can follow the following steps[4]:

- Load an ASCII description of the SN P System which must be compatible with inference input description (see section 3.1.)
- Click the `RUN` button.
- Wait
- Navigate through preview images using the ARROW buttons.
- Catch output files, generated in the application directory, showing the evolution in a graphical way.

[4] Along this paper we provide a brief description of the use of the simulator. For a comprehensive list of features, see the file `Readme.txt` distributed with the software for complete information and updated features.

Controls and displays

- *Preview:* After running a simulation, the output PostScript files are converted into two JPEG series with different resolutions. The software shows the lowest resolution picture here to preview the evolution.
- *Navigation buttons:* Two arrows are provided in the preview window, so the user can see the step-by-step evolution of the graph.
- *Input content:* A simple text container where the user can check the input for the engine after having loaded it.
- *Command buttons:* With these buttons the user can browse for an input file in the O.S., run the simulation once the file has been loaded or call the *Graphical Designer* to paint neurons, synapses and to write rules.

The SN P System Graphical Designer

This internal module inside the GUI provides a graphic interface to generate SN P system description files. The user simply draw neurons, synapses and describe rules using a friendly interface. After that, the depicted graph can be exported into a file that can be loaded and modified into the GUI in future sessions.

As one can see in Figure 2, a neuron is just a colored square with some text inside. We have tried to keep notation as standard as possible according with the bibliography without adding unnecessary complexity to the code. As an effective and intuitive way of represent the information in the system we have inside of each neuron the following text:

- *Label (top-left corner of the neuron):* A labeling system based in the creation order is used. The current version does not allow the user to change the label of the neuron. The first created neuron is labeled as 1, second as 2 and so on. If during the creation process a neuron is deleted, the remaining ones are relabeled to avoid holes in the label list.
- *Initial configuration (top-right of the neuron):* Number of spikes of the neuron in the initial configuration.
- *Set of rules:* The `X`-th rule of the of the neuron `N` is denoted by `N-X`. Because of the space limitation, the application just show the first four rules inside each neuron this way. To see all the rules, the Neuron Menu must be activated.

The environment is depicted as a special neuron with a synapse from the output neuron. Obviously, no rules or synapses for this neuron can be defined. With this interface the user can create, destroy (delete) and connect neurons just by clicking neurons, environment and background surface.

Rule editor

A simple rule editor is also provided. It shows to the user the patterns of *compatible-engine* rules. By using it, the user can create, modify or delete rules related to the

selected neuron. Since there is no syntax corrector implemented in the rule editor at the current version, the user must check the rules before the system is exported. The rule editor gives patterns for the seven rules that the engine can recognize. Every a is intended to be a *spike* and the rest of constant (s, c, p and d) of the pattern (e.g. `a^s/a^c \to a^p;d`) *must be replaced by integers* in order to have an engine-compatible file.

4 An Example

In this section we present how the simulator works on a well known example. We have taken the SN P system of Figure 1 and show the use of our simulator in order to build its transition diagram. Figure 6, taken from [13], shows the transition diagram of that SN P system.

After loading the corresponding file, which can be generated by using a text editor or the designer tool, the simulation can start. At time zero, only the initial configuration is shown.

After that, the user can develop the transition diagram step by step. The new elements (nodes or arcs) in each step are depicted in red color. Figure 7 shows the different stages of the development of the transition diagram. When no new element can be added to the diagram, it is finished. Figure 8 shows the same transition diagram from Figure 6 with the representation of our simulator.

5 Final Remarks and Future Work

The success of the simulators in membrane computing is beyond any doubt. On the one hand, one of the main usefulness of these simulators lies in their use for a better understanding of membrane computing, so it is a pedagogical tool of first hand. On the other hand, the simulators have proved to be a useful assistant tool for the design and verification of complex P systems which solve problems, saving the researchers heavy hand-made calculations.

To the best of our knowledge, the simulators presented in the literature corresponds to cell-like models of P systems. In this paper we present a first software tool for dealing with SN P systems with the hope that it will become a useful tool in the same way like the cell-like ones, both for pedagogical purposes as well as an assistant for researchers.

Since membrane computing is a vivid area and new variants of P systems based on neurons and spikes can appear, we have designed a software tool with a code flexible enough to adapt the simulator to forthcoming ideas. Indeed, one of its main features is its modularity and the ability of embedding new P system models in future releases with a minimal modification of the code.

As pointed in [7], one of the common features of the first generation of simulators for cell-like P systems was the lack of efficiency in favor of the expressivity.

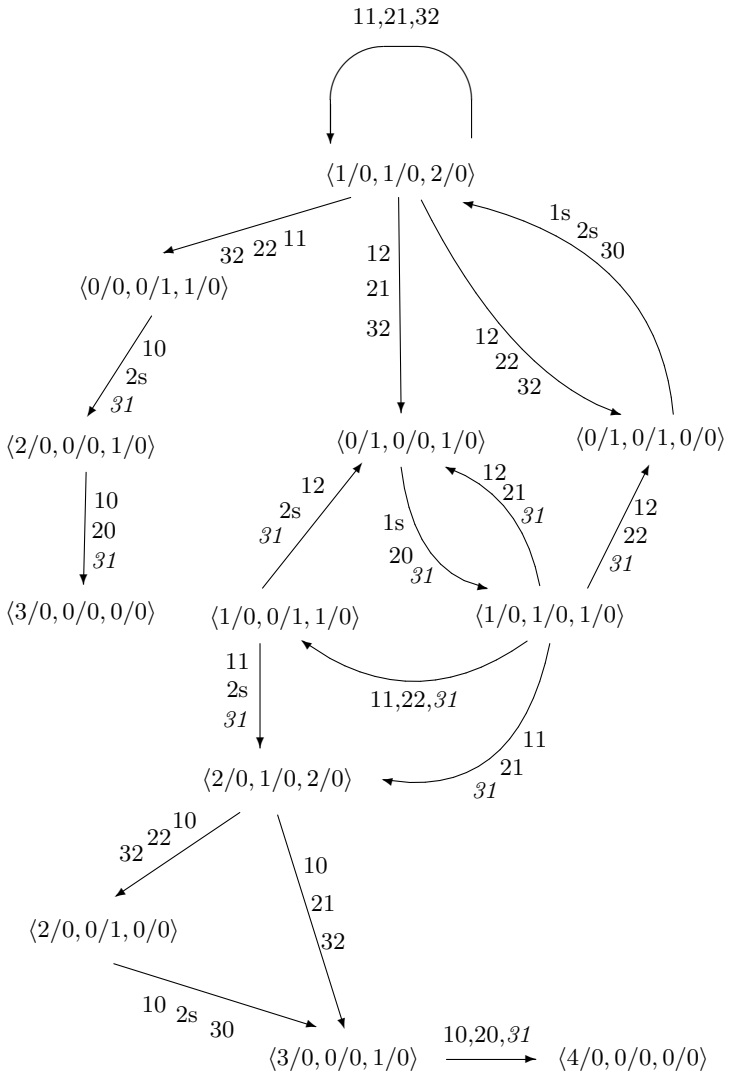


Fig. 6. The transition diagram from [13]

The current version of the simulator follows the same line: we keep a high degree of expressivity, looking for a friendly interface with the user, but some technical details can be improved. As pointed out above, one of them is the possibility of fixing a bound of the number of steps performed from the initial configuration in order to avoid that the software collapses. Anther point is to extend the definition of the applicability of rules, since in the current version only a restricted definition is used.

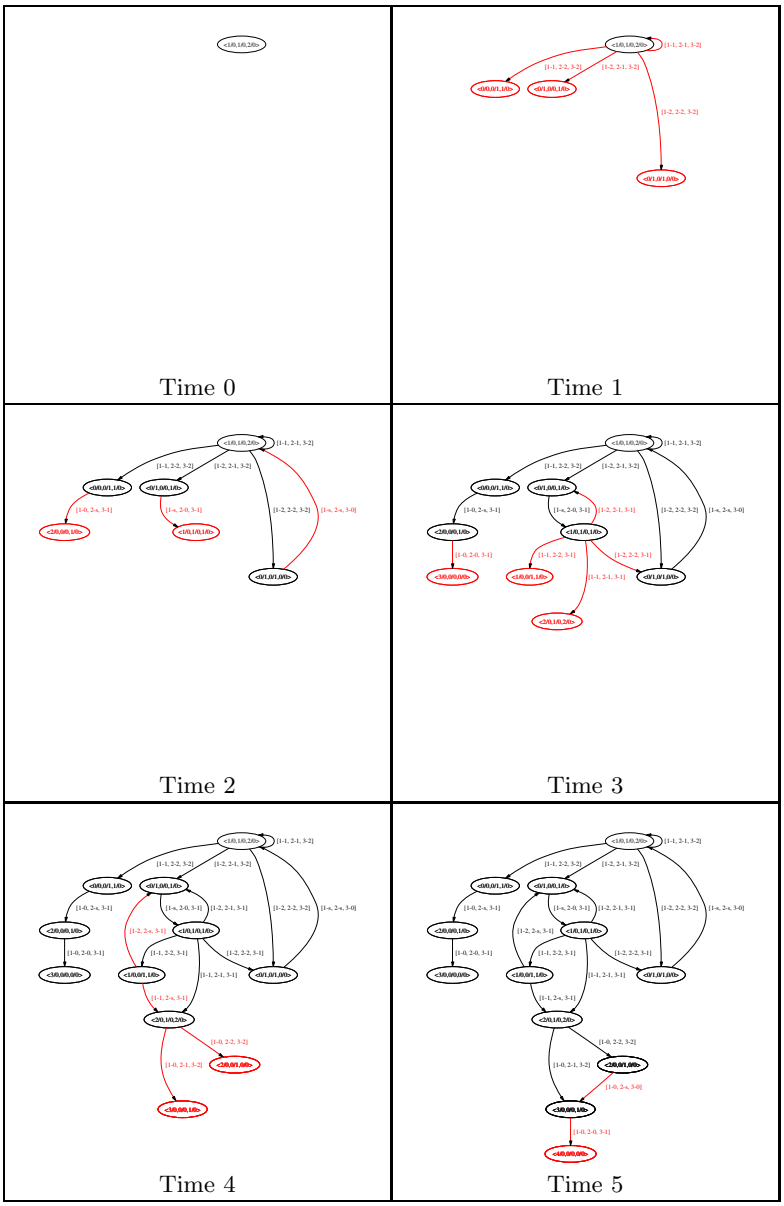


Fig. 7. Evolution of the example

We hope that this simulator will become a useful tool for the P systems community and our aim is to adapt it according to the new developments in the field. All suggestions are absolutely welcome.

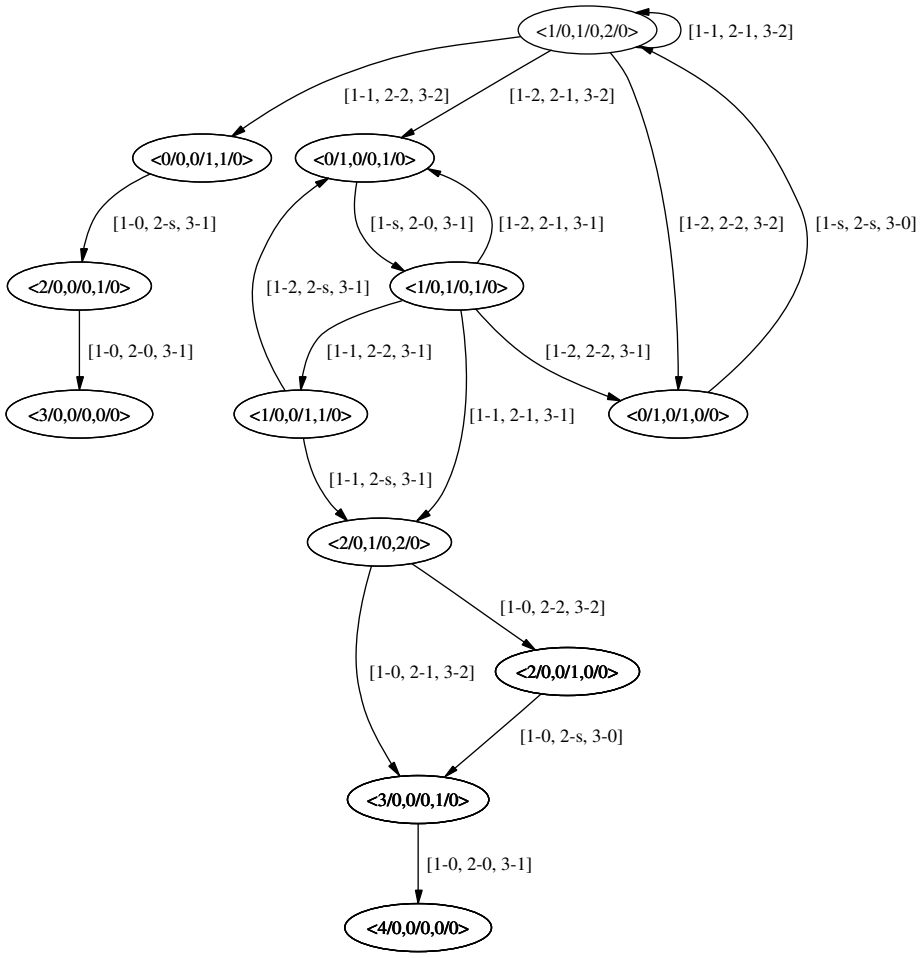


Fig. 8. The transition diagram from generated by the simulator

Acknowledgement

The second author acknowledges the support of the project TIN2005-09345-C04-01 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the project of excelence TIC-581 of the Junta de Andalucía.

References

1. L. Bianco, F. Fontana, G. Franco, and V. Manca: P Systems for Biological Dynamics. In [4], 83–128.
2. H. Cheng, R. Freund, M. Ionescu, Gh. Păun, and M.J. Pérez-Jiménez: On String Languages Generated by Spiking Neural P Systems. In *Fourth Brainstorming Week on Membrane Computing*, Vol. I (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.) Fénix Editora, Sevilla, 2006, 169–193.
3. G. Ciobanu and D. Paraschiv: P System Software Simulator. *Fundamenta Informaticae*, 49, 1-3 (2002), 61–66.
4. G. Ciobanu, Gh. Păun, and M.J. Pérez-Jiménez, eds.: *Applications of Membrane Computing*. Springer, 2006.
5. G. Ciobanu and G. Wenyuan: P Systems Running on a Cluster of Computers. In *Membrane Computing WMC 2003*. (C. Martín-Vide, Gh. Păun, G. Rozenberg, A. Salomaa, eds.) Lecture Notes in Computer Science, 2933, (2004), 123–139.
6. M. Ionescu, Gh. Păun, and T. Yokomori: Spiking Neural P Systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
7. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, and A. Riscos-Núñez: Available Membrane Computing Software. In [4], 411–439.
8. M. Maliţa: Membrane Computing in Prolog, *Pre-proceedings of the Workshop on Multiset Processing* (C.S. Calude, M.J. Dinneen, Gh. Păun, eds.) Curtea de Argeş, Romania, CDMTCS TR 140, Univ. of Auckland, 2000, 159–175.
9. C. Martín Vide, J. Pazos, Gh. Păun, and A. Rodríguez Patón: A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. *Lecture Notes in Computer Science* 2387, (2002), 290–299.
10. C. Martín Vide, J. Pazos, Gh. Păun, and A. Rodríguez Patón: Tissue P Systems. *Theoretical Computer Science*, 296, (2003), 295–326.
11. Gh. Păun: Computing with Membranes. *Journal of Computer and System Sciences*, 61, 1, (2000), 108–143.
12. Gh. Păun: *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
13. Gh. Păun: Twenty Six Research Topics About Spiking Neural P Systems. In this volume.
14. The SWI-Prolog web page `http://www.swi-prolog.org`
15. The Alaska Software web page `http://www.alaska-software.com`
16. P systems web page `http://psystems.disco.unimib.it/`
17. Group for Models of Natural Computing in Verona `http://www.di.univr.it`

On Two Families of Multiset Tree Automata*

José M. Sempere, Damián López

Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
{jsempere,dlopez}@dsic.upv.es

Summary. The relation between the membrane structures of P systems and an extension of tree automata which introduces multisets in the transition function has been proposed in previous works. Here we propose two features of tree automata which have been previously studied (namely, reversibility and local testability) in order to extend them to multiset tree automata. The characterization of these families will introduce a new characterization of membrane structures defined by the set of rules used for membrane creation and deletion.

1 Introduction

The relation between membrane structures and tree languages has been explored in previous works. So, Freund et al. [4] proved that P systems are able to generate recursively enumerable sets of trees through their membrane structures. Other works have focused on extending the definition of finite tree automata in order to take into account the membrane structures generated by P systems. So, in [13], the authors propose an extension of tree automata, namely multiset tree automata, in order to recognize membrane structures. In [7], the authors propose the use of this model to calculate editing distances between membrane structures. Later, the authors proposed a method to infer multiset tree automata from membrane observations [14].

In this work we introduce two new families of multiset tree automata, by using previous results taken from tree language theory. We propose a formal definition of reversible multiset tree automata and local testable multiset tree automata. These features have been widely studied in previous works [6, 8].

The structure of this work is simple: first we give basic definitions and notation for tree languages, P systems and multiset tree automata and we define the new families of multiset tree automata. Finally, we give some guidelines for future research.

* Work supported by the Spanish Generalitat Valenciana under contract GV06/068.

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 [12], [10] and [2].

Multisets

First, we will provide some definitions from multiset theory as exposed in [15].

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 the multiset $(A \ominus B) \oplus (B \ominus A)$ is empty.*

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 mappings*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \rightarrow \mathbb{N}^n$ where $D = \{d_1, d_2, \cdots, d_n\}$. 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.

P systems

We will introduce basic concepts from membrane systems taken from [10]. 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 $u \to v$) 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*).
  From now on, we will denote the set *tar* by $\{here, out, in_k : 1 \leq k \leq m\}$.
- 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*.

One of the multiple variations of P systems is related to the creation, division and modification of membrane structures. There have been several works in which these variants have been proposed (see, for example, [1, 9, 10, 11]).

In the following, we enumerate some kind of rules which are able to modify the membrane structure:

1. 2-division: $[_h a]_h \to [_{h'} b]_{h'} [_{h''} c]_{h''}$
2. Creation: $a \to [_h b]_h$
3. Dissolving: $[_h a]_h \to b$

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.

Tree automata and tree languages

Now, we will introduce some concepts from tree languages and automata as exposed in [3, 5]. First, let a *ranked alphabet* be the association of an alphabet V together with a finite relation r in $V \times \mathbb{N}$. We denote by V_n the subset $\{\sigma \in V : (\sigma, n) \in r\}$.

The set V^T of trees over V, is 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 catenation 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 unlabeled 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 $t : D \rightarrow V$. 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 $|x|$. 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 $depth(t) = max\{|x| : x \in dom(t)\}$. 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:

$$Sub(a) = \{a\} \text{ for all } a \in V_0$$
$$Sub(t) = \{t\} \cup \bigcup_{i=1,...,n} Sub(t_i) \text{ for } t = \sigma(t_1, ..., t_n) \ (n > 0)$$

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 $H^t =< root(t_1), \ldots, root(t_n) >$. Finally, $leaves(t)$ will denote the set of leaves of the tree t.

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 the set of final states and* $\delta = \bigcup_{i:V_i \neq \emptyset} \delta_i$ *is a set of transitions defined as follows:*

$$\delta_n : (V_n \times (Q \cup V_0)^n) \rightarrow Q \qquad n = 1, \ldots, m$$
$$\delta_0(a) = a \qquad \forall a \in V_0$$

Given the state $q \in Q$, we define the *ancestors* of the state q, denoted by $Ant(q)$, as the set of strings

$$Ant(q) = \{p_1 \cdots p_n : p_i \in Q \cup V_0 \wedge \delta_n(\sigma, p_1, ..., p_n) = q\}$$

From now on, we will refer to finite deterministic tree automata simply as *tree automata*. We suggest [3, 5] 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:

$$\begin{aligned}
&Q = Sub(T) \\
&F = T \\
&\delta_n(\sigma, u_1, \ldots, u_n) = \sigma(u_1, \ldots, u_n) && \sigma(u_1, \ldots, u_n) \in Q \\
&\delta_0(a) = a && a \in V_0
\end{aligned}$$

Let \$ be a new symbol in V_0, and $V_\$^T$ the set of trees $(V \cup \{\$\})^T$ where each tree contains \$ only once. We will name the node with label \$ as *link point* when necessary. Given $s \in V_\$^T$ and $t \in V^T$, the operation $s\#t$ is defined as:

$$s\#t(x) = \begin{cases} s(x) \text{ if } x \in dom(s),\ s(x) \neq \$ \\ t(z) \text{ if } x = yz,\ s(y) = \$,\ y \in dom(s) \end{cases}$$

therefore, given $t, s \in V^T$, let the tree quotient $(t^{-1}s)$ be defined as:

$$t^{-1}s = \begin{cases} r \in V_\$^T \ :\ s = r\#t & \text{if } t \in V^T - V_0. \\ t & \text{if } t \in V_0. \end{cases}$$

this quotient can be extended to consider set of trees $T \subseteq V^T$ as:

$$t^{-1}T = \{t^{-1}s \ :\ s \in T\}$$

For any $k \geq 0$, let the k-root of a tree t be defined as follows:

$$root_k(t) = \begin{cases} t, & \text{if } depth(t) < k \\ t' : t'(x) = t(x),\ x \in dom(t) \wedge |x| \leq k, & \text{otherwise} \end{cases}$$

Multiset tree automata and mirrored trees

We will extend over multisets some definitions of tree automata and tree languages. We will introduce the concept of multiset tree automata and then we will characterize the set of trees that it accepts.

Given any tree automata $A = (Q, V, \delta, F)$ 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 2.7 *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 transitions defined as follows:*

$$\delta = \bigcup_{\substack{1 \leq i \leq n \\ i\,:\,V_i \neq \emptyset}} \delta_i$$

$$\begin{aligned} \delta_i &: (V_i \times \mathcal{M}_i(Q \cup V_0)) \to \mathcal{P}(\mathcal{M}_1(Q)) && i = 1, \ldots, n \\ \delta_0(a) &= M_\Psi(a) \in \mathcal{M}_1(Q \cup V_0) && \forall a \in V_0 \end{aligned}$$

We can take notice that every tree automaton A defines a multiset tree automaton MA as follows

Definition 2.8 *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 $Ant_\Psi(q) = \{M : M_\Psi(q) \in \delta_n(\sigma, M)\}$.

Theorem 2.9 *(Sempere and López, [13]) 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 $M_\Psi(q) \in \delta'(t)$.*

Corolary 2.10 *(Sempere and López, [13]) 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 $t \in L(MA)$.*

We will introduce the concept of *mirroring* in tree structures as exposed in [13]. Informally speaking, two trees will be related by mirroring if some permutations at the structural level are hold. We propose a definition that relates all the trees with this mirroring property.

Definition 2.11 *Let 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* $\forall 1 \leq i \leq n\ t_i \bowtie s^i$

Theorem 2.12 *(Sempere and López, [13]) 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 $\delta'(t) = \delta'(s)$.*

Corolary 2.13 *(Sempere and López, [13]) 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 [13]. So, given two trees s and t, we can establish in time $\mathcal{O}((min\{|t|, |s|\})^2)$ if $t \bowtie s$.

3 *k*-testable in the strict sense (k-TSS) multiset tree languages and reversible multiset tree languages

In the following section, we will define two new classes of multiset tree languages. The definitions related to multiset tree automata come from the relation between mirrored trees and multiset tree automata which we have established in the previous section. So, whenever we refer to multiset tree languages we are taking under our consideration the set of (mirrored) trees accepted by multiset tree automata.

We refer [6] in order to know more about reversibility and local testability in tree languages.

First, we define k-TSS tree languages for any $k \geq 2$.

Definition 3.1 *Let $T \subseteq V^T$ and the integer value $k \geq 2$. T is a k-TSS multiset tree language if and only if, given whatever two trees $u_1, u_2 \in V^T$ such that $root_{k-1}(u_1) = root_{k-1}(u_2)$, $u_1^{-1}T \neq \emptyset$ and $u_2^{-1}T \neq \emptyset$ implies that $u_1^{-1}T = u_2^{-1}T$*

Any multiset tree automaton that holds the definition given before will be named a k-TSS multiset tree automaton. We can give the following characterization of such automata.

Corolary 3.2 *Let A be a k-TSS multiset tree automaton. There not exist two distinct states q_1, q_2 such that $root_k(q_1) \cap root_k(q_2) \neq \emptyset$*

Example 3.3 *Consider the multiset tree automaton with transitions:*

$$\begin{array}{r} \delta(\sigma, aa) = q_1 \\ \delta(\sigma, a) = q_2 \\ \delta(\sigma, aq_2) = q_2 \\ \delta(\sigma, q_1q_1) = q_1 \\ \hline \delta(\sigma, aq_2q_1) = q_3 \in F \end{array}$$

Note that the multiset tree language accepted by the automaton is k-TSS for any $k \geq 2$.

Note also that the following one does not hold the k-TSS condition for any $k \geq 2$:

$$\begin{array}{r} \delta(\sigma, aa) = q_1 \\ \delta(\sigma, bb) = q_2 \\ \delta(\sigma, q_2q_2) = q_2 \\ \delta(\sigma, q_1q_1) = q_1 \\ \hline \delta(\sigma, q_2q_1) = q_3 \in F \end{array}$$

because both the states q_1 and q_2 (and q_3) share a common k-root.

□

We also extend a previous result concerning k-reversible tree languages (for any $k \geq 0$) to give the following definition.

Definition 3.4 *Let $T \subseteq V^T$ and the integer value $k \geq 0$. T is a k-reversible multiset tree language if and only if, given whatever two trees $u_1, u_2 \in V^T$ such that $root_{k-1}(u_1) = root_{k-1}(u_2)$, whenever there exists a context $t \in V_{\$}^T$ such that both $u_1\#t, u_1\#t \in T$, then $u_1^{-1}T = u_2^{-1}T$*

Example 3.5 *Consider the multiset tree automaton with transitions:*

$$\begin{array}{r}
\delta(\sigma, aa) = q_1 \\
\delta(\sigma, a) = q_2 \\
\delta(\sigma, q_2q_2) = q_2 \\
\delta(\sigma, aaq_1) = q_1 \\
\hline
\delta(\sigma, q_1q_1) = q_3 \in F \\
\hline
\delta(\sigma, q_2q_1) = q_3 \in F
\end{array}$$

the multiset tree language accepted by this automaton is k-reversible and it is also an example of non k-TSS multiset tree language.

□

Finally, we can relate the two families of multiset tree languages that we have previously defined with the following result.

Theorem 3.6 *Let $T \subseteq V^T$ and an integer value $k \geq 2$, if T is k-TSS then T is $(k-1)$-reversible.*

Proof.

Let $t\#t_1$ and $t\#t_2$ belong to T, with $t \in V_{\$}^T$ and $root_k(t_1) = root_k(t_2)$, trivially $t_1^{-1}T \neq \emptyset$ and $t_2^{-1}T \neq \emptyset$. If T is a k-TSS tree language, then by previous definitions, $t_1^{-1}T = t_2^{-1}T$, and also T is $(k-1)$-reversible. □

4 Conclusions and future work

We have introduced two new families of multiset tree languages. Now, the open question is the characterization of membrane structures defined by them. We think that reversibility and local testability will introduce restrictions in the way of defining membrane creation and deletion. This will be explored in future works.

References

1. A. Alhazov, T.O. Ishdorj: Membrane operations in P systems with active membranes. In *Proc. Second Brainstorming Week on Membrane Computing*. TR 01/04 of RGNC, Sevilla University, 2004, 37–44.
2. C. Calude, Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Multiset Processing*. LNCS 2235, Springer, 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, release October, 1rst 2002.
4. R. Freund, M. Oswald, A. Păun: P systems generating trees. In *Pre-proceedings of Fifth Workshop on Membrane Computing*, WMC5 (G. Mauri, Gh. Păun, C. Zandron, eds.), MolCoNet project IST-2001-32008, 2004, 221–232.
5. F. Gécseg, M. Steinby: Tree languages. In *Handbook of Formal Languages*, volume 3, Springer, Berlin, 1997, 1–69.
6. D. López: *Inferencia de lenguajes de árboles.* PhD Thesis DSIC, Universidad Politécnica de Valencia, 2003.
7. D. López, J.M. Sempere: Editing distances between membrane structures. In *Proceedings of the 6th International Workshop on Membrane Computing*, Vienna, 2005 (R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 3850, Springer, 2006, 326–341.
8. D. López, J. M. Sempere, P. García: nference of reversible tree languages. *IEEE Transactions on Systems, Man and Cybernetics*, Part B: *Cybernetics*, 34, 4 (2004), 1658–1665.
9. A. Păun: On P systems with active membranes. In *Proc. of the First Conference on Unconventionals Models of Computation* (UMC2K), 2000, 187–201.
10. Gh. Păun: *Membrane Computing. An Introduction.* Springer, 2002.
11. Gh. Păun, Y. Suzuki, H. Tanaka, T. Yokomori: On the power of membrane division on P systems. In *Proc. Conf. on Words, Languages and Combinatorics*, Kyoto, 2000.
12. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages.* Springer, 1997.
13. J.M. Sempere, D. López :Recognizing membrane structures with tree automata. In *3rd Brainstorming Week on Membrane Computing, Sevilla, 2005.* RGNC Report 01/2005 Research Group on Natural Computing, Sevilla University, Fenix Editora, Sevilla, 2005, 305–316.
14. J.M. Sempere, D. López: Identifying P rules from membrane structures with an error-correcting approach. In *Proceedings of the 7th International Workshop on Membrane Computing*, Leiden, 2006 (H.J. Hoogeboom, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 4361, Springer, 2006, 507–520.
15. A. Syropoulos: Mathematics of multisets. In [2], 347–358.

Author Index