RGNC REPORT 2/2009

Seventh Brainstorming Week on Membrane Computing

Sevilla, February 2–February 6, 2009

Volume II

Miguel Angel Martínez-del-Amor,
Enrique Francisco Orejuela-Pinedo, Gheorghe Păun,
Ignacio Pérez-Hurtado, Agustín Riscos-Núñez
Editors

Research Group on
Natural Computing

REPORTS

UNIVERSIDAD DE SEVILLA

Seventh Brainstorming Week on Membrane Computing

Sevilla, February 2–February 6, 2009

Volume II

Miguel Angel Martínez-del-Amor,
Enrique Francisco Orejuela-Pinedo, Gheorghe Păun,
Ignacio Pérez-Hurtado, Agustín Riscos-Núñez
Editors

Seventh Brainstorming Week on Membrane Computing

Sevilla, February 2–February 6, 2009

Volume II

Miguel Angel Martínez-del-Amor,
Enrique Francisco Orejuela-Pinedo, Gheorghe Păun,
Ignacio Pérez-Hurtado, Agustín Riscos-Núñez
Editors

RGNC REPORT 2/2009

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2009

I.S.B.N.(Obra completa): 978-84-613-2836-9
I.S.B.N.(Volumen I): 978-84-613-2837-6
I.S.B.N.(Volumen II): 978-84-613-2839-0
Depósito Legal: SE-????–09
Edita: Fénix Editora
C/ Patricio Sáenz 13, 1–B
41004 Sevilla
fenixeditora1@telefonica.net
Telf. 954 41 29 91
Printed by Publidisa

Preface

These proceedings, consisting of two volumes, contain the papers emerged from the Seventh Brainstorming Week on Membrane Computing (BWMC), held in Sevilla, from February 2 to February 6, 2009, 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 five editions took place in Sevilla at the beginning of February 2004, 2005, 2006, 2007, and 2008, respectively.

In the style of previous meetings in this series, the seventh BWMC was conceived as a period of active interaction among the participants, with the emphasis on exchanging ideas and cooperation; this time, however, there were much more presentations than in the previous years, but still these presentations were "provocative", mainly proposing new ideas, open problems, research topics, results which need further improvements. The efficiency of this type of meetings was again proved to be very high and the present volumes prove this assertion.

As already usual, the number of participants was around 40, most of them computer scientists, but also a few biologists were present. It is important to note that several new names appeared in the membrane computing community, also bringing new view points and research topics to this research area.

The papers included in these volumes, 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 Computers, Control and Communication*. 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; after the fifth edition, a special issue of *International Journal of Unconventional Computing* was edited – volume 5, number 5, 2009; finally, a selection of papers elaborated during the sixth BWMC has appeared in a special issue of *Fundamenta Informaticae* – volume 87, number 1, 2008. Other papers elaborated during the seventh 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 domain website `http://ppage.psystems.eu`.

***

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

1. Alhazov Artiom, Hiroshima University, Japan,
   `aartiom@yahoo.com`
2. Ardelean Ioan, Institute of Biology of the Romanian Academy, Bucharest, Romania,
   `ioan.ardelean@ibiol.ro`
3. Bogdan Aman, A.I.Cuza University, Iasi, Romania,
   `baman@iit.tuiasi.ro`
4. Caravagna Giulio, University of Pisa, Italy,
   `caravagn@di.unipi.it`
5. Ceterchi Rodica, University of Bucharest, Romania,
   `rceterchi@gmail.com`
6. Colomer-Cugat M. Angels, University of Lleida, Spain,
   `Colomer@matematica.UdL.es`
7. Cordón-Franco Andrés, University of Sevilla, Spain,
   `acordon@us.es`
8. Csuhaj-Varjú Erzsébet, Hungarian Academy of Sciences, Budapest, Hungary,
   `csuhaj@sztaki.hu`
9. Díaz-Pernil Daniel, University of Sevilla, Spain,
   `sbdani@us.es`
10. Frisco Pierluigi, Heriot-Watt University, United Kingdom,
    `pier@macs.hw.ac.uk`
11. García-Quismondo Manuel, University of Sevilla, Spain,
    `mangarfer2@alum.us.es`
12. Graciani-Díaz Carmen, University of Sevilla, Spain,
    `cgdiaz@us.es`
13. Gutiérrez-Naranjo Miguel Ángel, University of Sevilla, Spain,
    `magutier@us.es`

14. Gutiérrez-Escudero Rosa, University of Sevilla, Spain,
rgutierrez@us.es
15. Henley Beverley, University of Sevilla, Spain,
bhenley@us.es
16. Ipate Florentin, University of Pitesti, Romania,
florentin.ipate@ifsoft.ro
17. Ishdorj Tseren-Onolt, Abo Akademi, Finland,
tishdorj@abo.fi
18. Krassovitskiy Alexander, Rovira i Virgili University, Tarragona, Spain,
alexander.krassovitskiy@estudiants.urv.cat
19. Leporati Alberto, University of Milano-Bicocca, Italy,
leporati@disco.unimib.it
20. Martínez-del-Amor Miguel Angel, University of Sevilla, Spain,
mdelamor@us.es
21. Mauri Giancarlo, University of Milano-Bicocca, Italy,
mauri@disco.unimib.it
22. Mingo Postiglioni Jack Mario, Carlos Tercero University, Madrid, Spain,
jmingo@inf.uc3m.es
23. Murphy Niall, NUI Maynooth, Ireland,
nmurphy@cs.nuim.ie
24. Obtułowicz Adam, Polish Academy of Sciences, Poland,
A.Obtulowicz@impan.gov.pl
25. Orejuela-Pinedo Enrique Francisco, University of Sevilla, Spain,
eorejuela@us.es
26. Pagliarini Roberto, University of Verona, Italy,
roberto.pagliarini@univr.it
27. Pan Linqiang, Huazhong University of Science and Technology, Wuhan, Hubei, China,
lqpan@mail.hust.edu.cn
28. Păun Gheorghe, Institute of Mathematics of the Romanian Academy, Bucharest, Romania, and University of Sevilla, Spain,
george.paun@imar.ro, gpaun@us.es
29. Pérez-Hurtado-de-Mendoza Ignacio, University of Sevilla, Spain,
perezh@us.es
30. Pérez-Jiménez Mario de Jesús, University of Sevilla, Spain,
marper@us.es
31. Porreca Antonio, University of Milano-Bicocca, Italy,
porreca@disco.unimib.it
32. Riscos-Núñez Agustín, University of Sevilla, Spain,
ariscosn@us.es
33. Rogozhin Yurii, Institute of Mathematics and Computer Science, Chisinau, Moldova,
rogozhin@math.md

34. Romero-Jiménez Alvaro, University of Sevilla, Spain,
Alvaro.Romero@cs.us.es
35. Sburlan Dragoş, Ovidius University, Constanţa, Romania,
dsburlan@univ-ovidius.ro
36. Sempere Luna José María, Polytechnical University of Valencia, Spain,
jsempere@dsic.upv.es
37. Vaszil György, Hungarian Academy of Sciences, Budapest, Hungary,
vaszil@sztaki.hu
38. Woods Damien, University of Sevilla, Spain,
dwoods@us.es
39. 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) Proyecto de Excelencia de la Junta de Andalucía, grant TIC 581, (ii) Proyecto de Excelencia con investigador de reconocida valía, de la Junta de Andalucía, grant P08 – TIC 04200, (iii) Proyecto del Ministerio de Educación y Ciencia, grant TIN2006 – 13425, (iv) IV Plan Propio de la Universidad de Sevilla, (v) Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía, well as by the Department of Computer Science and Artificial Intelligence from Sevilla University.

Gheorghe Păun
Mario de Jesús Pérez-Jiménez
(Sevilla, April 10, 2009)

Contents

Contents of Volume 1

Deterministic Solutions to QSAT and Q3SAT by Spiking Neural P Systems with Pre-Computed Resources

Tseren-Onolt Ishdorj[1], Alberto Leporati[2], Linqiang Pan[3⋆], Xiangxiang Zeng[3], Xingyi Zhang[3]

[1] Computational Biomodelling Laboratory
Department of Information Technologies
Åbo Akademi University, Turku 20520, Finland
tishdorj@abo.fi

[2] Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
alberto.leporati@unimib.it

[3] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, People's Republic of China
lqpan@mail.hust.edu.cn, zxxhust@gmail.com, xyzhanghust@gmail.com

Summary. In this paper we continue previous studies on the computational efficiency of spiking neural P systems, under the assumption that some pre-computed resources of exponential size are given in advance. Specifically, we give a deterministic solution for each of two well known **PSPACE**-complete problems: QSAT and Q3SAT. In the case of QSAT, the answer to any instance of the problem is computed in a time which is linear with respect to both the number n of Boolean variables and the number m of clauses that compose the instance. As for Q3SAT, the answer is computed in a time which is at most cubic in the number n of Boolean variables.

1 Introduction

Spiking neural P systems (in short, SN P systems) were introduced in [7] in the framework of Membrane Computing [15] as a new class of computing devices which are inspired by the neurophysiological behavior of neurons sending electrical impulses (spikes) along axons to other neurons. Since then, many computational properties of SN P systems have been studied; for example, it has been proved

⋆ Corresponding author. Tel.: +86-27-87556070. Fax: +86-27-87543130.

that they are Turing-complete when considered as number computing devices [7], when used as language generators [4, 2] and also when computing functions [13].

Investigations related to the possibility to solve computationally hard problems by using SN P systems were first proposed in [3]. The idea was to encode the instances of decision problems in a number of spikes, to be placed in an (arbitrarily large) pre-computed system at the beginning of the computation. It was shown that the resulting SN P systems are able to solve the **NP**-complete problem `SAT` (the satisfiability of propositional formulas expressed in conjunctive normal form) in a constant time. Slightly different solutions to `SAT` and `3-SAT` by using SN P systems with pre-computed resources were considered in [8]; here the encoding of an instance of the given problem is introduced into the pre-computed resources in a polynomial number of steps, while the truth values are assigned to the Boolean variables of the formula and the satisfiability of the clauses is checked. The answer associated to the instance of the problem is thus computed in a polynomial time. Finally, very simple semi-uniform and uniform solutions to the numerical **NP**-complete problem `Subset Sum` — by using SN P systems with exponential size pre-computed resources — have been presented in [9]. All the systems constructed above work in a deterministic way.

A different idea of constructing SN P systems for solving **NP**-complete problems was given in [11, 12], where the `Subset Sum` and `SAT` problems were considered. In these papers, the solutions are obtained in a semi-uniform or uniform way by using nondeterministic devices but without pre-computed resources. However, several ingredients are also added to SN P systems such as extended rules, the possibility to have a choice between spiking rules and forgetting rules, etc. An alternative to the constructions of [11, 12] was given in [10], where only standard SN P systems without delaying rules, and having a uniform construction, are used. However, it should be noted that the systems described in [10] either have an exponential size, or their computations last an exponential number of steps. Indeed, it has been proved in [12] that a deterministic SN P system of polynomial size cannot solve an **NP**-complete problem in a polynomial time unless **P=NP**. Hence, under the assumption that $\mathbf{P} \neq \mathbf{NP}$, efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance the efficiency of the system (pre-computed resources, ways to exponentially grow the workspace during the computation, nondeterminism, and so on).

The present paper deals with `QSAT` (the satisfiability of fully quantified propositional formulas expressed in conjunctive normal form) and with `Q3SAT` (where the clauses that compose the propositional formulas have exactly three literals), two well known **PSPACE**-complete problems. For `QSAT` we provide a family $\{\Pi_{QSAT}(2n, m)\}_{n,m\in\mathbb{N}}$ of SN P systems with pre-computed resources such that for all $n, m \in \mathbb{N}$ the system $\Pi_{QSAT}(2n, m)$ solves all the instances of `QSAT` which are built using $2n$ Boolean variables and m clauses. Each system $\Pi_{QSAT}(2n, m)$ is deterministic, and computes the solution in a time which is linear with respect to both n and m; however, the size of $\Pi_{QSAT}(2n, m)$ is exponential with respect to the size of the instances of the problem. As for `Q3SAT`, we provide a family

$\{\Pi_{Q3SAT}(2n)\}_{n\in\mathbb{N}}$ of SN P systems with pre-computed resources, such that for all $n \in \mathbb{N}$ the system $\Pi_{Q3SAT}(2n)$ solves all possible instances of Q3SAT which can be built using $2n$ Boolean variables. Also in this case the systems $\Pi_{Q3SAT}(2n)$ are deterministic and have an exponential size with respect to n. Given an instance of Q3SAT, the corresponding answer is computed in a time which is at most cubic in n.

An important observation is that we will not specify how our pre-computed systems could be built. However, we require that such systems have a *regular* structure, and that they do not contain neither "hidden information" that simplify the solution of specific instances, nor an encoding of all possible solutions (that is, an exponential amount of information that allows to cheat while solving the instances of the problem). These requirements were inspired by open problem Q27 in [15]. Let us note in passing that the regularity of the structure of the system is related to the concept of *uniformity*, that in some sense measures the difficulty of constructing the systems. Usually, when considering families $\{C(n)\}_{n\in\mathbb{N}}$ of Boolean circuits, or other computing devices whose number of inputs depends upon an integer parameter $n \geq 1$, it is required that for each $n \in \mathbb{N}$ a "reasonable" description (see [1] for a discussion on the meaning of the term "reasonable" in this context) of $C(n)$, the circuit of the family which has n inputs, can be produced in polynomial time and logarithmic space (with respect to n) by a deterministic Turing machine whose input is 1^n, the unary representation of n. In this paper we will not delve further into the details concerning uniformity; we just rely on reader's intuition, by stating that it should be possible to build the entire structure of the system in a polynomial time, using only a polynomial amount of information and a controlled replication mechanism, as it already happens in P systems with cell division. We will thus say that our solutions are *exp-uniform* (instead of *uniform*), since the systems $\Pi_{QSAT}(2n, m)$ and $\Pi_{Q3SAT}(2n)$ have an exponential size.

The paper is organized as follows. In section 2 we recall the formal definition of SN P systems, as well as some mathematical preliminaries that will be used in the following. In section 3 we provide an exp-uniform family $\{\Pi_{QSAT}(2n, m)\}_{n,m\in\mathbb{N}}$ of SN P systems with pre-computed resources such that for all $n, m \in \mathbb{N}$ the system $\Pi_{QSAT}(2n, m)$ solves all possible instances of QSAT containing $2n$ Boolean variables and m clauses. In section 4 we present an exp-uniform family $\{\Pi_{Q3SAT}(2n)\}_{n\in\mathbb{N}}$ of SN P systems with pre-computed resources such that for all $n \in \mathbb{N}$ the system $\Pi_{Q3SAT}(2n)$ solves all the instances of Q3SAT which can be built using $2n$ Boolean variables. Section 5 concludes the paper and suggests some possible directions for future work.

2 Preliminaries

We assume the reader to be familiar with formal language theory [16], computational complexity theory [5] as well as membrane computing [15]. We mention here only a few notions and notations which are used throughout the paper.

For an alphabet V, V^* denotes the set of all finite strings over V, with the empty string denoted by λ. The set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we simply write a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$.

A regular expression over an alphabet V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over V. Non-necessary parentheses can be omitted when writing a regular expression, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

For a string $str = y_1 y_2 \dots y_{2n}$, where $y_{2k-1} \in \{0, 1\}$, $y_{2k} \in \{0, 1, x_{2k}\}$, $1 \leq k \leq n$, we denote by $str|_i$ the ith symbol of the string str, $1 \leq i \leq 2n$. For given $1 \leq i \leq n$, if str such that the $2j$th symbol is x_{2j} for all $j \leq i$ and the $2j'$th symbol equals to 1 or 0 for all $j' \geq i$, then we denote by $str|_{2i} \leftarrow x$ a string which is obtained by replacing the $2i$th symbol of str with x_{2i}. In particular, for a binary string $bin \in \{0, 1\}^{2n}$, $bin|_i$ and $bin|_2 \leftarrow x_2$ are defined as the ith bit of bin and the string obtained by replacing the second bit of bin with x_2, respectively.

`QSAT` is a well known **PSPACE**-complete decision problem (see for example [5, pages 261–262], where some variants of the problem `Quantified Boolean Formulas` are defined). It asks whether or not a given fully quantified Boolean formula, expressed in the conjunctive normal form (CNF), evaluates to *true* or *false*. Formally, an instance of `QSAT` with n variables and m clauses is a formula $\gamma_{n,m} = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n (C_1 \wedge C_2 \wedge \ldots \wedge C_m)$ where each Q_i, $1 \leq i \leq n$, is either $\forall$ or $\exists$, and each clause C_j, $1 \leq j \leq m$, is a disjunction of the form $C_j = y_1 \vee y_2 \vee \ldots \vee y_{r_j}$, with each literal y_k, $1 \leq k \leq r_j$, being either a propositional variable x_s or its negation $\neg x_s$, $1 \leq s \leq n$. For example, the propositional formula $\beta = Q_1 x_1 Q_2 x_2 [(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)]$ is *true* when $Q_1 = \forall$ and $Q_2 = \exists$, whereas it is *false* when $Q_1 = \exists$ and $Q_2 = \forall$. The decision problem `Q3SAT` is defined exactly as `QSAT`, the only difference being that all the clauses now contain *exactly* three literals. It is known that even under this restriction the problem remains **PSPACE**-complete (see, for example, [5, page 262]).

In what follows we require that no repetitions of the same literal may occur in any clause. Without loss of generality we can also avoid the clauses in which both the literals x_s and $\neg x_s$, for any $1 \leq s \leq n$, occur. Further, we will focus our attention on the instances of `QSAT` and `Q3SAT` in which all the variables having an even index (that is, $x_2, x_4, \ldots$) are *universally* quantified, and all the variables with an odd index ($x_1, x_3, \ldots$) are *existentially* quantified. We will say that such instances are expressed in *normal form*. This may be done without loss of generality. In fact, for any instance $\gamma_{n,m} = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n (C_1 \wedge C_2 \wedge \ldots \wedge C_m)$ of `QSAT` having n variables and m clauses there exists an equivalent instance $\gamma'_{2n,m} = \exists x'_1 \forall x'_2 \ldots \exists x'_{2n-1} \forall x'_{2n} (C'_1 \wedge C'_2 \wedge \ldots \wedge C'_m)$ with $2n$ variables, where

each clause C'_j is obtained from C_j by replacing every variable x_i by x'_{2i-1} if $Q_i = \exists$, or by x'_{2i} if $Q_i = \forall$. Note that this transformation may require to introduce some "dummy" variables, that is, variables which are quantified in $\gamma'_{2n,m}$ to guarantee the alternance of even-numbered and odd-numbered variables, but that nonetheless do not appear in any clause. For example, for the propositional formula $\beta_1 = \forall x_1 \exists x_2[(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)]$ the normal form is $\beta'_1 = \exists x'_1 \forall x'_2 \exists x'_3 \forall x'_4[(x'_2 \vee x'_3) \wedge (\neg x'_2 \vee \neg x'_3)]$; for the propositional formula $\beta_2 = \exists x_1 \forall x_2[(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)]$ we have the normal form $\beta'_2 = \exists x'_1 \forall x'_2 \exists x'_3 \forall x'_4[(x'_1 \vee x'_4) \wedge (\neg x'_1 \vee \neg x'_4)]$. The same transformation may be applied on any instance of Q3SAT defined on n Boolean variables; in this case the result will be another instance of Q3SAT, defined on $2n$ variables. From now on we will denote by $QSAT(2n, m)$ the set of all possible instances of QSAT, expressed in the above normal form, which are built using $2n$ Boolean variables and m clauses. Similarly, we will denote by $Q3SAT(2n)$ the set of all possible instances of Q3SAT, expressed in normal form, which can be built using $2n$ Boolean variables.

2.1 Spiking neural P systems

As stated in the Introduction, SN P systems have been introduced in [7], in the framework of Membrane Computing. They can be considered as an evolution of P systems, corresponding to a shift from *cell-like* to *neural-like* architectures.

In SN P systems the cells (also called *neurons*) are placed in the nodes of a directed graph, called the *synapse graph*. The contents of each neuron consists of a number of copies of a single object type, called the *spike*. Every cell may also contain a number of *firing* and *forgetting* rules. Firing rules 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 applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use one of its rules, then one of such rules must be used. If two or more rules could be applied, then only one of them is nondeterministically chosen. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other. Note that, as usually happens in Membrane Computing, a global clock is assumed, marking the time for the whole system, and hence the functioning of the system is synchronized. When 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, then there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

Formally, a *spiking neural membrane system* (SN P system, for short) of degree $m \geq 1$, as defined in [7], 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 $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$, 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 form $E/a^c \rightarrow a^p; d$, where E is a regular expression over a, and $c \geq 1$, $p \geq 0$, $d \geq 0$, with the restriction $c \geq p$;
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.

A rule $E/a^c \rightarrow a^p; d$ with $p \geq 1$ is an *extended firing* (we also say spiking) rule; a rule $E/a^c \rightarrow a^p$ with $p = 0$ is written in the form $E/a^c \rightarrow \lambda$ and is called an *extended forgetting* rule. Rules of the types $E/a^c \rightarrow a; d$ and $a^c \rightarrow \lambda$ are said to be *standard*.

If a rule $E/a^c \rightarrow a^p; d$ has $E = a^c$, then we will write it in the simplified form $a^c \rightarrow a^p; d$; similarly, if a rule $E/a^c \rightarrow a^p; d$ has $d = 0$, then we can simply write it as $E/a^c \rightarrow a^p$; hence, if a rule $E/a^c \rightarrow a^p; d$ has $E = a^c$ and $d = 0$, then we can write $a^c \rightarrow a^p$.

The rules are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \rightarrow a^p; d$ is enabled and can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in neuron σ_i); 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, etc. 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 (this corresponds to the refractory period from neurobiology), 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). In the step $t + d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t + d + 1$, when the neuron can again apply rules). Once emitted from neuron σ_i, the p spikes reach immediately all neurons σ_j such that $(i, j) \in syn$ and which are open, that is, the p spikes are replicated and each target neuron receives p spikes; as stated above, spikes sent to a closed neuron are "lost", that is, they are removed from the system. In the case of the output neuron, p 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. If the rule is a forgetting one of the form $E/a^c \rightarrow \lambda$, then, when it is applied, $c \geq 1$ spikes are removed.

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 $E_1/a^{c_1} \rightarrow a^{p_1}; d_1$ and $E_2/a^{c_2} \rightarrow a^{p_2}; d_2$ 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; in such a case, only one of them is chosen in a nondeterministic way. However it is assumed that if a firing rule is applicable then no forgetting rule is applicable, and vice versa. Thus, the rules are used in the sequential manner in each

neuron (at most one in each step), but neurons work in parallel with each other. It is important to note that the applicability of a rule is established depending on the total number of spikes contained in the neuron.

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 the *state* of the neuron, that is, the number of steps to count down until it becomes open again (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 σ_i contains $r_i \geq 0$ spikes and it will be open after $t_i \geq 0$ steps, for $i = 1, 2, \ldots, m$; with this notation, the initial configuration of the system is $C_0 = \langle n_1/0, \ldots, n_m/0 \rangle$. Using the rules as described above, one 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.

Since in SN P systems the alphabet contains only one symbol (denoted by a), the input information is sometimes encoded as a sequence of "virtual" symbols, λ or a^i, $i \geq 1$, where λ represents no spike and a^i represents a multiset of i spikes. The input sequence is then introduced in the input neuron one virtual symbol at one time unite, starting from the leftmost symbol of the sequence. For instance, the sequence $a^2\lambda a^3$ is composed of three virtual symbols: a^2, λ and a^3. When providing this sequence as input to an SN P system, the virtual symbol a^2 (that is, two spikes) is introduced at the first computation step, followed by λ (0 spikes) at the next step, and finally by a^3 (three spikes) at the third step.

Another useful extension to the model defined above, already considered in [10, 12, 11, 8], is to use several input neurons, so that the introduction of the encoding of an instance of the problem to be solved can be done in a faster way, introducing parts of the code in parallel in various input neurons. Formally, we can define an SN P system of degree (m, ℓ), with $m \geq 1$ and $0 \leq \ell \leq m$, just like a standard SN P system of degree m, the only difference being that now there are ℓ input neurons denoted by $in_1, \ldots, in_\ell$. A *valid input* for an SN P system of degree (m, ℓ) is a set of ℓ binary sequences (where each element of the sequence denotes the presence or the absence of a spike), that collectively encode an instance of a problem.

Spiking neural P systems can be used to solve decision problems, both in a *semi–uniform* and in a *uniform* way. When solving a problem $\mathcal{Q}$ in the *semi–uniform* setting, for each specified instance $\mathcal{I}$ of $\mathcal{Q}$ we build in a polynomial time (with respect to the size of $\mathcal{I}$) an SN P system $\Pi_{\mathcal{Q},\mathcal{I}}$, whose structure and initial configuration depend upon $\mathcal{I}$, that halts (or emits a specified number of spikes in a given interval of time) if and only if $\mathcal{I}$ is a positive instance of $\mathcal{Q}$. On the other hand, a *uniform* solution of $\mathcal{Q}$ consists of a family $\{\Pi_{\mathcal{Q}}(n)\}_{n\in\mathbb{N}}$ of SN P systems such that, when having an instance $\mathcal{I} \in \mathcal{Q}$ of size n, we introduce a polynomial (in n) number of spikes in a designated (set of) input neuron(s) of $\Pi_{\mathcal{Q}}(n)$ and the computation halts (or, alternatively, a specified number of spikes is emitted in a

given interval of time) if and only if $\mathcal{I}$ is a positive instance. The preference for uniform solutions over semi–uniform ones is given by the fact that they are more strictly related to the structure of the problem, rather than to specific instances. Indeed, in the semi–uniform setting we do not even need any input neuron, as the instance of the problem is embedded into the structure (number of spikes, graph of neurons and synapses, rules) from the very beginning. If the instances of a problem $\mathcal{Q}$ depend upon two parameters (as is the case of `QSAT`, where n is the number of variables and m the number of clauses in a given formula), then we will denote the family of SN P systems that solves $\mathcal{Q}$ by $\{\Pi_{\mathcal{Q}}(n,m)\}_{n,m\in\mathbb{N}}$. Alternatively, if one does not want to make the family of SN P systems depend upon two parameters, it is possible to define it as $\{\Pi_{\mathcal{Q}}(\langle n,m\rangle)\}_{n,m\in\mathbb{N}}$, where $\langle n,m\rangle$ indicates the positive integer number obtained by applying an appropriate bijection (for example, Cantor's pairing) from $\mathbb{N}^2$ to $\mathbb{N}$.

In the above definitions it is assumed that the uniform (resp., semi-uniform) construction of $\Pi_{\mathcal{Q}}(n)$ (resp., $\Pi_{\mathcal{Q},\mathcal{I}}$) is performed by using a deterministic Turing machine, working in a polynomial time. As stated in the Introduction, the SN P systems we will describe will solve all the instances of `QSAT` and `Q3SAT` of a given size, just like in the uniform setting. However, such systems will have an exponential size. Since a deterministic Turing machine cannot produce (the description of) an exponential size object in a polynomial time, we will say that our solutions are *exp-uniform*.

3 An exp-Uniform Solution to `QSAT`

In this section we build an exp-uniform family $\{\Pi_{QSAT}(2n,m)\}_{n,m\in\mathbb{N}}$ of SN P systems such that for all $n,m\in\mathbb{N}$ the system $\Pi_{QSAT}(2n,m)$ solves all the instances of $QSAT(2n,m)$ in a polynomial number of steps with respect to n and m, in a deterministic way.

The instances of $QSAT(2n,m)$ to be given as input to the system $\Pi_{QSAT}(2n,m)$ are encoded as sequences of virtual symbols, as follows. For any given instance $\gamma_{2n,m}=\exists x_1\forall x_2\ldots\exists x_{2n-1}\forall x_{2n}(C_1\wedge C_2\wedge\ldots\wedge C_m)$ of $QSAT(2n,m)$, let $code(\gamma_{2n,m})=\alpha_{11}\alpha_{12}\cdots\alpha_{12n}\alpha_{21}\alpha_{22}\cdots\alpha_{22n}\cdots\alpha_{m1}\alpha_{m2}\cdots\alpha_{m2n}$, where each α_{ij}, for $1\leq i\leq m$ and $1\leq j\leq 2n$, is a *spike variable* whose value is an amount of spikes (a virtual symbol), assigned as follows:

$$\alpha_{ij}=\begin{cases}a, & \text{if } x_j \text{ occurs in } C_i;\\ a^2, & \text{if } \neg x_j \text{ occurs in } C_i;\\ \lambda, & \text{otherwise.}\end{cases}$$

In this way the sequence $\alpha_{i1}\alpha_{i2}\cdots\alpha_{i2n}$ of spike variables represents the clause C_i, and the representation of $\gamma_{2n,m}$ is just the concatenation of the representations of the single clauses. As an example, the representation of $\gamma_{2,2}=\exists x_1\forall x_2[(x_1\vee\neg x_2)\wedge(\neg x_2)]$ is $aa^2\lambda a^2$. The set of all the encoding sequences of

all possible instances of $QSAT(2n, m)$ is denoted by $code(QSAT(2n, m))$. For instance, $QSAT(2, 1)$ contains the following nine formulas (the existential and the universal quantifiers are here omitted for the sake of readability): $\gamma^1_{2,1}$ = no variable appears in the clause, $\gamma^2_{2,1} = x_2$, $\gamma^3_{2,1} = \neg x_2$, $\gamma^4_{2,1} = x_1$, $\gamma^5_{2,1} = x_1 \vee x_2$, $\gamma^6_{2,1} = x_1 \vee \neg x_2$, $\gamma^7_{2,1} = \neg x_1$, $\gamma^8_{2,1} = \neg x_1 \vee x_2$, $\gamma^9_{2,1} = \neg x_1 \vee \neg x_2$. Therefore, $code(QSAT(2, 1)) = \{\lambda\lambda, \lambda a, \lambda a^2, a\lambda, aa, aa^2, a^2\lambda, a^2a, a^2a^2\}$.

The structure of the pre-computed SN P system that solves all possible instances of $QSAT(2n, m)$ is illustrated in a schematic way in Figures 1 and 2. The system is a structure of the form $\Pi^{(2n,m)}_{QSAT} = (\Pi_{QSAT}(2n, m), code(QSAT(2n, m)))$ with:

- $\Pi_{QSAT}(2n, m) = (O, \mu, in, out)$, where:
  1. $O = \{a\}$ is the singleton alphabet;
  2. $\mu = (H, \bigcup_{i\in H}\{m_i\}, \bigcup_{j\in H} R_j, syn)$ is the structure of the SN P system, where:
  - $H = H_0 \cup H_1 \cup H_2 \cup H_3$ is a finite set of neuron labels, with
    $H_0 = \{in, out, d\} \cup \{d_i \mid 0 \leq i \leq 2n\}$,
    $H_1 = \{Cx_i, Cx_i1, Cx_i0 \mid 1 \leq i \leq 2n\}$,
    $H_2 = \{bin, Cbin \mid bin \in \{0, 1\}^{2n}\}$,
    $H_3 = \{y_1y_2 \dots y_{2n-1}y_{2n} \mid y_i \in \{0, 1\}$ when i is odd and $y_i \in \{0, 1, x_i\}$ when i is even, and there exists at least one $k \in \{1, 2, \dots, n\}$ such that $y_{2k} = x_{2k}\}$ (we recall that even values of i correspond to universally quantified variables).
    All the neurons are injectively labeled with elements from H;
  - $m_{d_0} = 2$, $m_d = 1$ and $m_i = 0$ ($i \in H$, $i \neq d_0, d$) are the numbers of spikes that occur in the initial configuration of the system;
  - R_k, $k \in H$, is a finite set of rules associated with neuron σ_k, where:
    $R_{in} = \{a \to a, a^2 \to a^2\}$, $R_d = \{a \to a; 2mn + n + 6\}$,
    $R_{d_i} = \{a^2 \to a^2\}$, for $i \in \{0, 1, \dots, 2n-1\}$, and $R_{d_{2n}} = \{a^2 \to a^2, a^3 \to \lambda\}$,
    $R_{Cx_i} = \{a \to \lambda, a^2 \to \lambda, a^3 \to a^3; 2n - i, a^4 \to a^4; 2n - i\}$, for $i \in \{1, 2, \dots, 2n\}$,
    $R_{Cx_i1} = \{a^3 \to a^2, a^4 \to \lambda\}$ and $R_{Cx_i0} = \{a^3 \to \lambda, a^4 \to a^2\}$, for $i \in \{1, 2, \dots, 2n\}$,
    $R_{Cbin} = \{(a^2)^+/a \to a\} \cup \{a^{2k-1} \to \lambda \mid k = 1, 2, \dots, 2n\}$, for $bin \in \{0, 1\}^{2n}$,
    $R_{bin} = \{a^m \to a\}$, for $bin \in \{0, 1\}^{2n}$,
    $R_{str} = \{a^2 \to a\}$, where $str \in H_3$ and there exists at least one $i \in \{1, 2, \dots, n\}$ such that $str|_{2i} \neq x_{2i}$,
    $R_{str'} = \{a^2 \to a^2\}$, where $str' \in H_3$ and $str'|_{2k} = x_{2k}$, for all $1 \leq k \leq n$,
    $R_{out} = \{(a^2)^+/a \to a\}$;
  - syn is the set of all the synapses between the neurons. The following synapses are used in the input module (see Figure 1): (in, Cx_i), (d_{i-1}, d_i), (d_i, Cx_i), (Cx_i, Cx_i1) and (Cx_i, Cx_i0), for all $1 \leq i \leq 2n$, as well as (d_{2n}, d_1) and (d, d_{2n}).
    The synapses connecting the other neurons are illustrated in Figure 2:

$(Cx_ij, Cbin)$, where $bin \in \{0,1\}^{2n}$ and $bin|_i = j, 1 \leq i \leq 2n$, $j \in \{0,1\}$,
$(Cbin, bin)$, where $bin \in \{0,1\}^{2n}$,
(bin, str), where $bin \in \{0,1\}^{2n}$, $str \in H_3$, and $str = (bin|_i \leftarrow x_2)$,
(str_{j_1}, str_{j_2}), where $str_{j_1}, str_{j_2} \in H_3$ and $str_{j_2} = (str_{j_1}|_{2i} \leftarrow x_{2i})$, $2 \leq i \leq n$,
(str, out), where $str \in H_3$ and $str|_{2k} = x_{2k}$, for all $1 \leq k \leq n$;

3. in, out indicate the *input* and *output* neurons, respectively;

- $code(QSAT(2n,m))$ is the set of all the encoding sequences for all the possible instances of $QSAT(2n,m)$, as defined above.

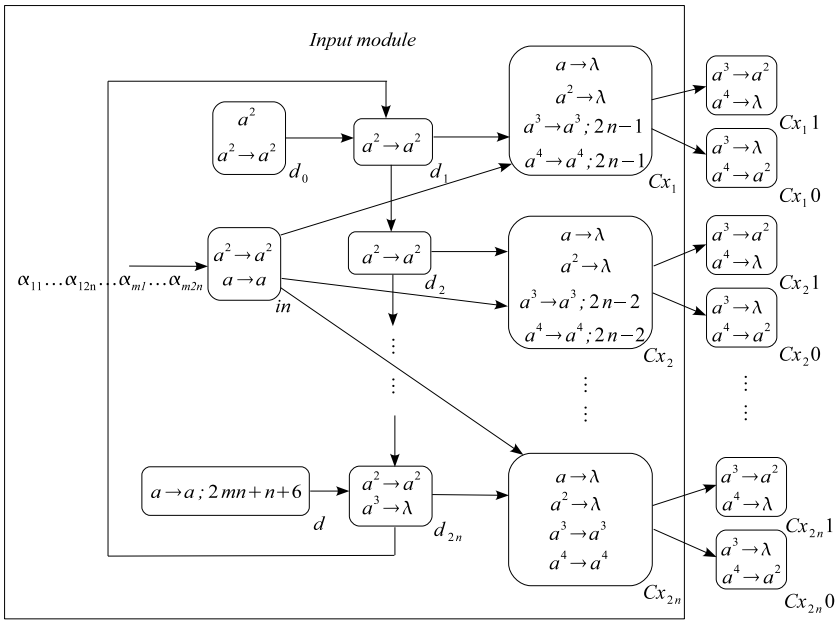


Fig. 1. The input module of $\Pi_{QSAT}(2n,m)$

The system is composed of four modules: *input, satisfiability checking, quantifier checking*, and *output*. To simplify the description of the system and its working, the neurons in the system are arranged in $n+7$ layers in Figures 1 and 2. The input module has three layers (the first layer contains three neurons σ_{d_0}, σ_d and σ_{in}; the second layer contains $2n$ neurons σ_{d_i}, $1 \leq i \leq 2n$; the third layer contains $2n$ neurons σ_{Cx_i}, $1 \leq i \leq 2n$). The satisfiability checking module has also three layers (the fourth layer contains $4n$ neurons σ_{Cx_ij}, $1 \leq i \leq 2n$, $j = 0,1$; the fifth layer contains 2^{2n} neurons σ_{Cbin}, $bin \in \{0,1\}^{2n}$; the sixth layer contains 2^{2n} neurons σ_{bin}, $bin \in \{0,1\}^{2n}$). The quantifier checking module is composed of n layers, from the 7th to the $(n+6)$th layer, where a total of $2^{2n} - 2^n$ neurons are used.

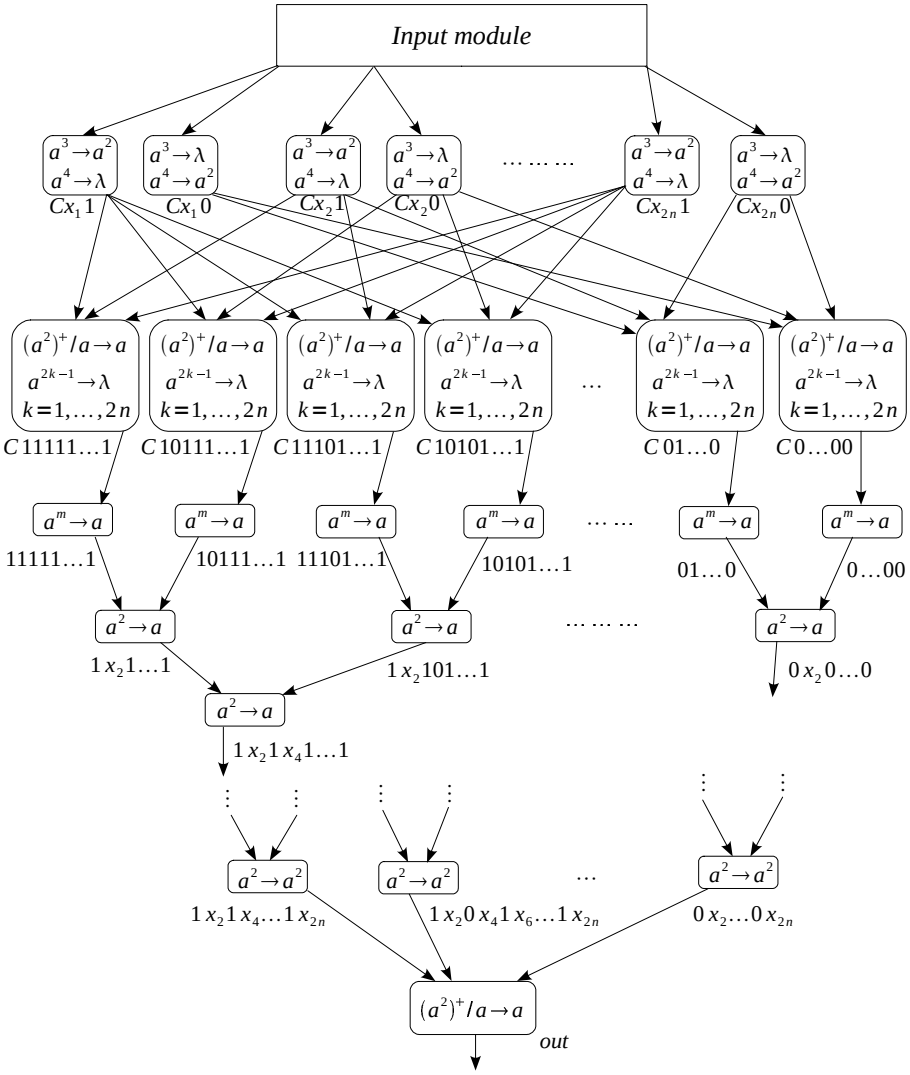


Fig. 2. Structure of the SN P system $\Pi_{QSAT}(2n, m)$

The output module only contains one neuron σ_{out}, which appears in the last layer. In what follows we provide a more detailed description of each module, as well as its working when solving a given instance $\gamma_{2n,m} \in QSAT(2n, m)$.

- *Input*: The input module consists of $4n + 3$ neurons, contained in the layers 1 – 3 as illustrated in Figure 1; σ_{in} is the unique input neuron. The values of the spike variables of the encoding sequence $code(\gamma_{2n,m})$ are introduced into σ_{in}

one by one, starting from the beginning of the computation. At the first step of the computation, the value of the first spike variable α_{11}, which is the virtual symbol that represents the occurrence of the first variable in the first clause, enters into neuron σ_{in}; in the meanwhile, neuron σ_{d_1} receives two auxiliary spikes from neuron σ_{d_0}. At this step, the firing rule in neuron σ_d is applied; as a result, neuron σ_d will send one spike to neuron $\sigma_{d_{2n}}$ after $2mn+n+6$ steps (this is done in order to halt the computation after the answer to the instance given as input has been determined). In the next step, the value of the spike variable α_{11} is replicated and sent to neurons σ_{Cx_i}, for all $i \in \{1, 2, \ldots, 2n\}$; the two auxiliary spikes contained in σ_{d_1} are also sent to neurons σ_{Cx_1} and σ_{d_2}. Hence, neuron σ_{Cx_1} will contain 2, 3 or 4 spikes: if x_1 occurs in C_1, then neuron σ_{Cx_1} collects 3 spikes; if $\neg x_1$ occurs in C_1, then it collects 4 spikes; if neither x_1 nor $\neg x_1$ occur in C_1, then it collects two spikes. Moreover, if neuron σ_{Cx_1} has received 3 or 4 spikes, then it will be closed for $2n-1$ steps, according to the delay associated with the rules in it; on the other hand, if 2 spikes are received, then they are deleted and the neuron remains open. At the third step, the value of the second spike variable α_{12} from neuron σ_{in} is distributed to neurons σ_{Cx_i}, $2 \leq i \leq 2n$, where the spikes corresponding to α_{11} are deleted. At the same time, the two auxiliary spikes are duplicated and one copy of them enters into neurons σ_{Cx_2} and σ_{d_3}, respectively. The neuron σ_{Cx_2} will be closed for $2n-2$ steps only if it contains 3 or 4 spikes, which means that this neuron will not receive any spike during this period. In neurons σ_{Cx_i}, $3 \leq i \leq 2n$, the spikes represented by α_{12} are forgotten in the next step.
In this way, the values of the spike variables are introduced and delayed in the corresponding neurons until the value of the spike variable α_{12n} of the first clause and the two auxiliary spikes enter together into neuron $\sigma_{Cx_{2n}}$ at step $2n+1$. At that moment, the representation of the first clause of $\gamma_{2n,m}$ has been entirely introduced in the system, and the second clause starts to enter into the input module. The entire sequence $code(\gamma_{2n,m})$ is introduced in the system in $2mn+1$ steps.

- *Satisfiability checking*: Once all the values of spike variables α_{1i} $(1 \leq i \leq 2n)$, representing the first clause, have appeared in their corresponding neurons σ_{Cx_i} in layer 3, together with a copy of the two auxiliary spikes, all the spikes contained in σ_{Cx_i} are duplicated and sent simultaneously to the pair of neurons σ_{Cx_i1} and σ_{Cx_i0}, for $i \in \{1, 2, \ldots, 2n\}$, at the $(2n+2)$nd computation step. In this way, each neuron σ_{Cx_i1} and σ_{Cx_i0} receives 3 or 4 spikes when x_i or $\neg x_i$ occurs in C_1, respectively, whereas it receives no spikes when neither x_i or $\neg x_i$ occurs in C_1. In general, if neuron σ_{Cx_i1} receives 3 spikes, then the literal x_i occurs in the current clause (say C_j), and thus the clause is satisfied by all those assignments in which x_i is true. Neuron σ_{Cx_i0} will also receive 3 spikes, but it will delete them during the next computation step. On the other hand, if neuron σ_{Cx_i1} receives 4 spikes, then the literal $\neg x_i$ occurs in C_j, and the clause is satisfied by those assignments in which x_i is false. Since neuron σ_{Cx_i1} is designed to process the case in which x_i occurs in C_j, it will

delete its 4 spikes. However, also neuron σ_{Cx_i0} will have received 4 spikes, and this time it will send two spikes to those neurons which are bijectively associated with the assignments for which x_i is false. Note that all possible 2^{2n} truth assignments to $x_1, x_2, \ldots, x_{2n}$ are represented by the neurons' labels $Cbin$ in layer 5, where bin is generated from $\{0,1\}^{2n}$; precisely, we read bin, where $bin|_i = j$, $j \in \{0,1\}$, as a truth assignment whose value for x_i is j. In the next step, those neurons σ_{Cbin} that received at least two spikes send one spike to the corresponding neurons σ_{bin} in layer 6 (the rest of the spikes will be forgotten), with the meaning that the clause is satisfied by the assignment bin. This occurs in the $(2n+4)$th computation step. Thus, the check for the satisfiability of the first clause has been performed; in a similar way, the check for the remaining clauses can proceed. All the clauses can thus be checked to see whether there exist assignments that satisfy all of them.
If there exist some assignments that satisfy all the clauses of $\gamma_{2n,m}$, then the neurons labeled with the values of $bin \in \{0,1\}^{2n}$ that correspond to these assignments succeed to accumulate m spikes. Thus, the rule $a^m \to a$ can be applied in these neurons. The satisfiability checking module completes its process in $2mn+5$ steps.

- *Quantifier checking*: The universal and existential quantifiers of the fully quantified formula $\gamma_{2n,m}$ are checked starting from step $2mn+6$.
Since all the instances of $QSAT(2n,m)$ are in the normal form, it is not difficult to see that we need only to check the universal quantifiers associated to even-numbered variables $(x_2, x_4, \ldots)$. These universal quantifiers are checked one by one, and thus the quantifier checking module needs n steps to complete its process. The module starts by checking the universal quantifier associated with x_2, which is performed as follows. For any two binary sequences bin_1 and bin_2 with $bin_1|_i = bin_2|_i$ for all $i \neq 2$ and $bin_1|_2 = 1$, $bin_2|_2 = 0$, if both neurons σ_{bin_1} and σ_{bin_2} contain m spikes, then neuron σ_{str} will receive two spikes from them at step $2mn+5$, where $str = (bin_1|_2 \leftarrow x_2)$. This implies that, no matter whether $x_2 = 1$ or $x_2 = 0$, if we assign each variable x_j with the value $str|_j$, $j \neq 2$, $1 \leq j \leq 2n$, then all the clauses of $\gamma_{2n,m}$ are satisfied. As shown in Figure 2, in this way the system can check, in the 7th layer, the satisfiability of the universal quantifier associated to variable x_2. The system is then ready to check the universal quantifier associated with variable x_4, which is performed in a similar way as follows. For any two sequences str_1 and str_2 with $str_1|_i = str_2|_i \in \{0,1\}$, for all $i \neq 2,4$, and $str_1|_2 = str_2|_2 = x_2$, $str_1|_4 = 1$, $str_2|_4 = 0$, if both neurons σ_{str_1} and σ_{str_2} contain two spikes, then σ_{str_3} will receive two spikes, where str_3 is obtained by replacing the fourth symbol of str_1 with x_4 (i.e., $str_3 = (str_1|_4 \leftarrow x_4)$). In this way, we check the (simultaneous) satisfiability of the universal quantifiers associated to the two variables x_2 and x_4. Similarly, the system can check the satisfiability of the universal quantifier associated with variable x_6 by using the neurons in the ninth layer. Therefore, after n steps (in the $(n+6)$th layer) the system has checked the satisfiability of all the universal quantifiers associated with

the variables $x_2, x_4, \ldots, x_{2n}$. If a neuron σ_{str} accumulates two spikes, where $str|_{2k} = x_{2k}$ for all $1 \leq k \leq n$, then we conclude that this assignment not only makes all the clauses satisfied, but also satifies all the quantifiers. Therefore, the neurons which accumulate two spikes will send two spikes to the output neuron, thus indicating that the instance of the problem given as input is positive.

- *Output*: From the construction of the system, it is not difficult to see that the output neuron sends exactly one spike to the environment at the $(2mn+n+6)$th computation step if and only if $\gamma_{2n,m}$ is *true*. At this moment, neuron σ_d will also send a spike to the auxiliary neuron $\sigma_{d_{2n}}$ (the rule is applied in the first computation step). This spike stays in neuron $\sigma_{d_{2n}}$ until two further spikes arrive from neuron $\sigma_{d_{2n-1}}$; when this happens, all three spikes are forgotten by using the rule $a^3 \to \lambda$ in neuron $\sigma_{d_{2n}}$. Hence, the system eventually halts after a few steps since the output neuron fires.

Note that the number m of clauses appearing in a $QSAT(2n, m)$ problem may be very large (e.g., exponential) with respect to n: every variable can occur negated or non-negated in a clause, or not occur at all, and hence the number of all possible clauses is 3^{2n}. This means that the running time of the system may be exponential with respect to n, and also the rules $a^m \to a$ in some neurons of the system are required to work on a possibly very large number of spikes. As we will see in the next section, these "problems" (if one considers them as problems) do not occur when considering `Q3SAT`, since each clause in the formula contains exactly three literals, and thus the number of possible clauses is at most cubic in n.

3.1 An example

Let us present a simple example which shows how the system solves the instances of $QSAT(2n, m)$, for specified values of n and m, in an exp-uniform way. Let us consider the following fully quantified propositional formula, which has two variables and two clauses (i.e., $n = 1, m = 2$):

$$\gamma_{2,2} = \exists x_1 \forall x_2 (x_1 \vee \neg x_2) \wedge x_1$$

Such a formula is encoded as the sequence $code(\gamma_{2,2}) = aa^2a\lambda$ of virtual symbols.

The structure of the SN P system which is used to solve all the instances of $QSAT(2, 2)$ is pre-computed as illustrated in Figure 3. It is composed of 22 neurons; its computations are performed as follows.

Input: Initially, neuron σ_{d_0} contains two spikes and neuron σ_d contains one spike, whereas the other neurons in the system contain no spikes. The computation starts by inserting the leftmost symbol of the encoding sequence $code(\gamma_{2,2}) = aa^2a\lambda$ into the input neuron σ_{in}. When this symbol (a) enters into the system, neuron σ_{d_0} emits its two spikes to neuron σ_{d_1}. At this moment, the rule occurring in neuron σ_d also fires; as a result, it will send one spike to neuron σ_{d_2} after 11 steps. At the next step, two spikes from σ_{d_1} are sent to neurons σ_{d_2} and σ_{Cx_1},

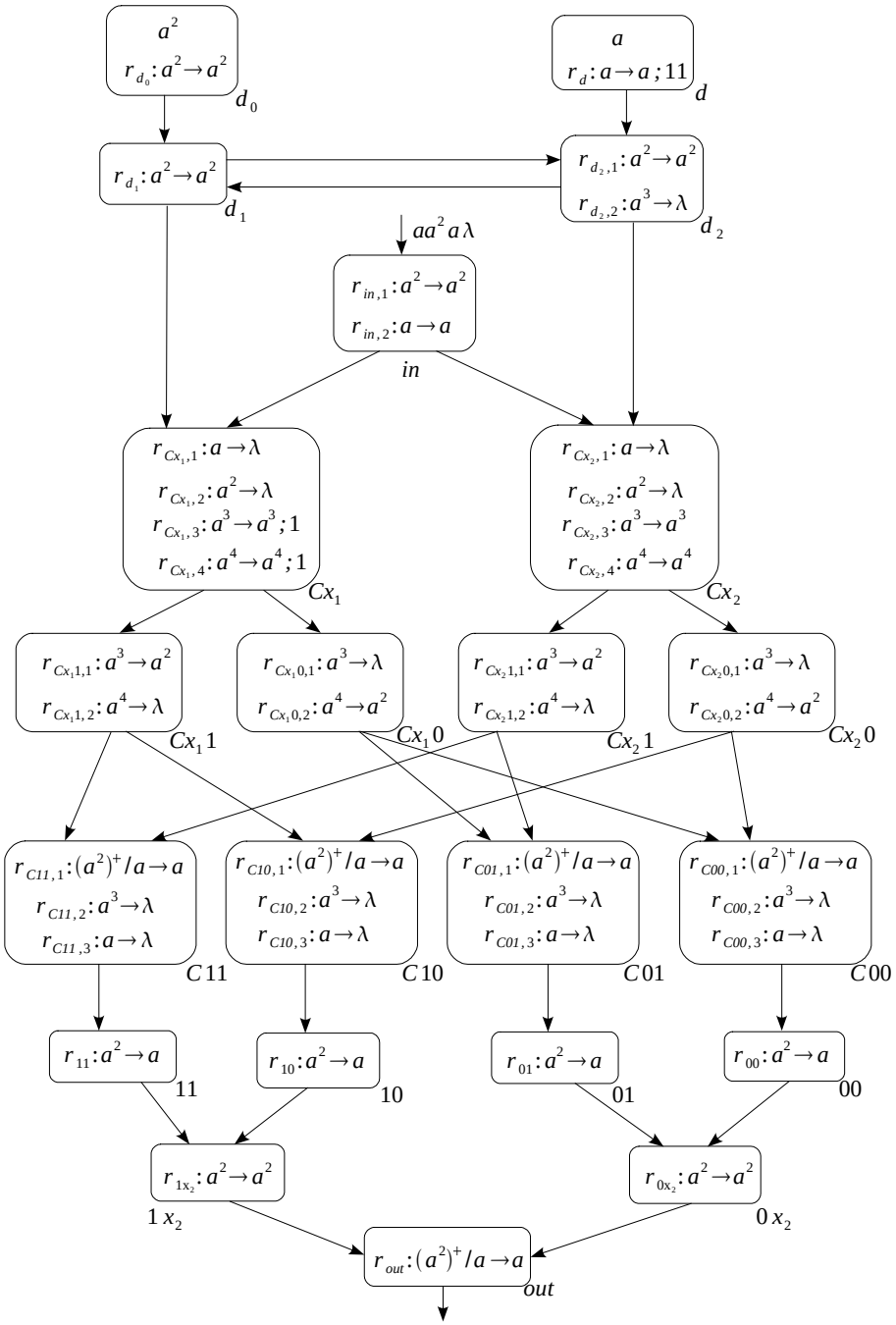


Fig. 3. The pre-computed structure of the SN P system $\Pi_{QSAT}(2,2)$

while the symbol a is sent by σ_{in} to neurons σ_{Cx_1} and σ_{Cx_2}. At the same time, the value a^2 of the second spike variable α_{12} of $code(\gamma_{2,2})$ is introduced into σ_{in}.

Neuron σ_{Cx_1} has accumulated three spikes and thus the rule $a^3 \to a^2; 1$ can be applied in it, while the spike in neuron σ_{Cx_2} is forgotten by using the rule $a \to \lambda$ at the third computation step. Simultaneously, the value a^2 of the second spike variable α_{12} from σ_{in} and two spikes from σ_{d_2} enter together into σ_{Cx_2}; neuron σ_{Cx_1} does not receive any spike, as it has been closed for this step. Thus, at the third computation step the representation of the first clause aa^2 has appeared in the input module. At this step, the value a of the first spike variable α_{21} of the second clause also enters the input neuron, while neuron σ_{d_1} receives two spikes again, which triggers the introduction of the representation of the second clause $(a\lambda)$ in the input module.

Satisfiability checking: Now, neuron σ_{Cx_1} is open and fires, sending three spikes to neurons σ_{Cx_11} and σ_{Cx_10}. The three spikes in neuron σ_{Cx_11} denote that literal x_1 occurs in the current clause (C_1), and thus the clause is satisfied by all those assignments in which $x_1 = 1$. And, in fact, σ_{Cx_11} sends two spikes to neurons σ_{C11} and σ_{C10}, to indicate that the first clause is satisfied by the assignments *bin* whose first value is 1. The three spikes in neuron σ_{Cx_10} denote that the current clause (C_1) does not contain the literal $\neg x_1$. Hence, no spike is emitted from neuron σ_{Cx_10}; its three spikes are forgotten instead. Similarly, the presence of four spikes in neuron σ_{Cx_21} (resp., in σ_{Cx_20}) denotes the fact that literal x_2 (resp., $\neg x_2$) does not occur (resp., occurs) in clause C_1. Hence, the spikes in neuron σ_{Cx_21} are forgotten, whereas neuron σ_{Cx_20} sends two spikes to neurons σ_{C10} and σ_{C00} to denote that clause C_1 is satisfied by those assignments in which $x_2 = 0$.

At step 5, the configuration of the system is as follows. Three spikes occur in neuron σ_{Cx_1}, since literal x_1 occurs in the second clause; no spikes occur in σ_{Cx_2}, as the clause does not contain variable x_2; the two auxiliary spikes appear alternately in neurons σ_{d_1} and σ_{d_2} in the input module. At the same time, neurons σ_{C11} and σ_{C00} contain two spikes each, whereas neuron σ_{C10} contains four spikes.

In the next step, neuron σ_{Cx_1} sends three spikes to its two target neurons σ_{Cx_11} and σ_{Cx_10}, while each of the neurons σ_{C11}, σ_{C00} and σ_{C10} sends one spike towards their related neurons in the next layer, thus confirming that the first clause is satisfied by the corresponding assignments. The rest of spikes in these neurons will be forgotten in the following step. At step 7, neuron σ_{Cx_11} sends two spikes to neurons σ_{C11} and σ_{C10} by using the rule $a^3 \to a^2$, whereas the three spikes in neuron σ_{Cx_10} are forgotten. Note that the spike in neurons σ_{11}, σ_{10} and σ_{00}, which is received from their related neurons σ_{C11}, σ_{C00} and σ_{C10}, remains unused until one more spike is received. At step 8, neuron σ_{C11} sends one spike to its related neuron σ_{11} and neuron σ_{C10} sends one spike to its related neuron σ_{10}, while the rest of spikes are forgotten. In this way, neurons σ_{11} and σ_{10} succeed to accumulate a sufficient number (two) of spikes to fire. On the other hand, neuron σ_{00} fails to accumulate the desired number of spikes (it obtains only one spike), thus the rule in it cannot be activated.

Quantifier checking: We now pass to the module which checks the universal and existential quantifiers associated to the variables. At step 9 neuron σ_{1x_2} receives two spikes, one from σ_{11} and another one from σ_{10}, which means that the formula $\gamma_{2,2}$ is satisfied when $x_1 = 1$, no matter whether $x_2 = 0$ or $x_2 = 1$. On the other hand, neuron σ_{0x_2} does not contain any spike. At step 10 the rule $a^2 \to a^2$ is applied in neuron σ_{1x_2}, making neuron σ_{out} receive two spikes.

Output: The rule occurring in neuron σ_{out} is activated and one spike is sent to the environment, indicating that the instance of the problem given as input is positive (that is, $\gamma_{2,2}$ is *true*). At this step, as neuron σ_{d_2} will receive a "trap" spike from neuron σ_d, the two auxiliary spikes circulating in the input module are deleted as soon as they arrive in it, because of the rule $a^3 \to \lambda$. Thus, the system halts after 13 computation steps since it has been started.

In order to illustrate how the system from Figure 3 evolves in detail, its transition diagram (generated with the software tool developed in [6]) is also given in Figure 4. In this figure, $\langle r_1/t_1, \ldots, r_{21}/t_{21}, r_{22}/t_{22}\rangle$ is the configuration in which neurons σ_d, σ_{d_0}, σ_{d_1}, σ_{d_2}, σ_{in}, σ_{Cx_1}, σ_{Cx_2}, $\sigma_{Cx_1 1}$, $\sigma_{Cx_1 0}$, $\sigma_{Cx_2 1}$, $\sigma_{Cx_2 0}$, σ_{C11}, σ_{C10}, σ_{C01}, σ_{C00}, σ_{11}, σ_{10}, σ_{01}, σ_{00}, σ_{1x_2}, σ_{0x_2} and σ_{out} contain $r_1, r_2, \ldots, r_{22}$ spikes, respectively, and will be open after $t_1, t_2, \ldots, t_{22}$ steps, respectively. Between two configurations we draw an arrow if and only if a direct transition is possible between them. For simplicity, the rules are indicated only when they are used, while unused rules are omitted. When neuron σ_k spikes after being closed for $s \geq 0$ steps, we write $r_{k,s}$. We omit to indicate s when it is zero. Finally, we have highlighted the firing of σ_{out} by writing r_{out} in bold.

4 Solving Q3SAT

As stated in section 2, the instances of Q3SAT are defined like those of QSAT, with the additional constraint that each clause contains exactly three literals. In what follows, by $Q3SAT(2n)$ we will denote the set of all instances of Q3SAT which can be built using $2n$ Boolean variables $x_1, x_2, \ldots, x_{2n}$, with the following three restrictions: (1) no repetitions of the same literal may occur in any clause, (2) no clause contains both the literals x_s and $\neg x_s$, for any $s \in \{1, 2, \ldots, 2n\}$, and (3) the instance is expressed in the normal form described in section 2 (all even-numbered and odd-numbered variables are universally and existentially quantified, respectively).

As stated in the previous section, the number m of possible clauses that may appear in a formula $\gamma_{n,m} \in QSAT(n, m)$ is exponential with respect to n. On the contrary, the number of possible 3-clauses which can be built using $2n$ Boolean variables is $4n \cdot (4n-2) \cdot (4n-4) = \Theta(n^3)$, a polynomial quantity with respect to n. This quantity, that we denote by $Cl(2n)$, is obtained by looking at a 3-clause as a triple, and observing that each component of the triple may contain one of the $4n$ possible literals, with the constraints that we do not allow the repetition of literals in the clauses, or the use of the same variable two or three times in a clause.

Fig. 4. The transition diagram of the system illustrated in Figure 3

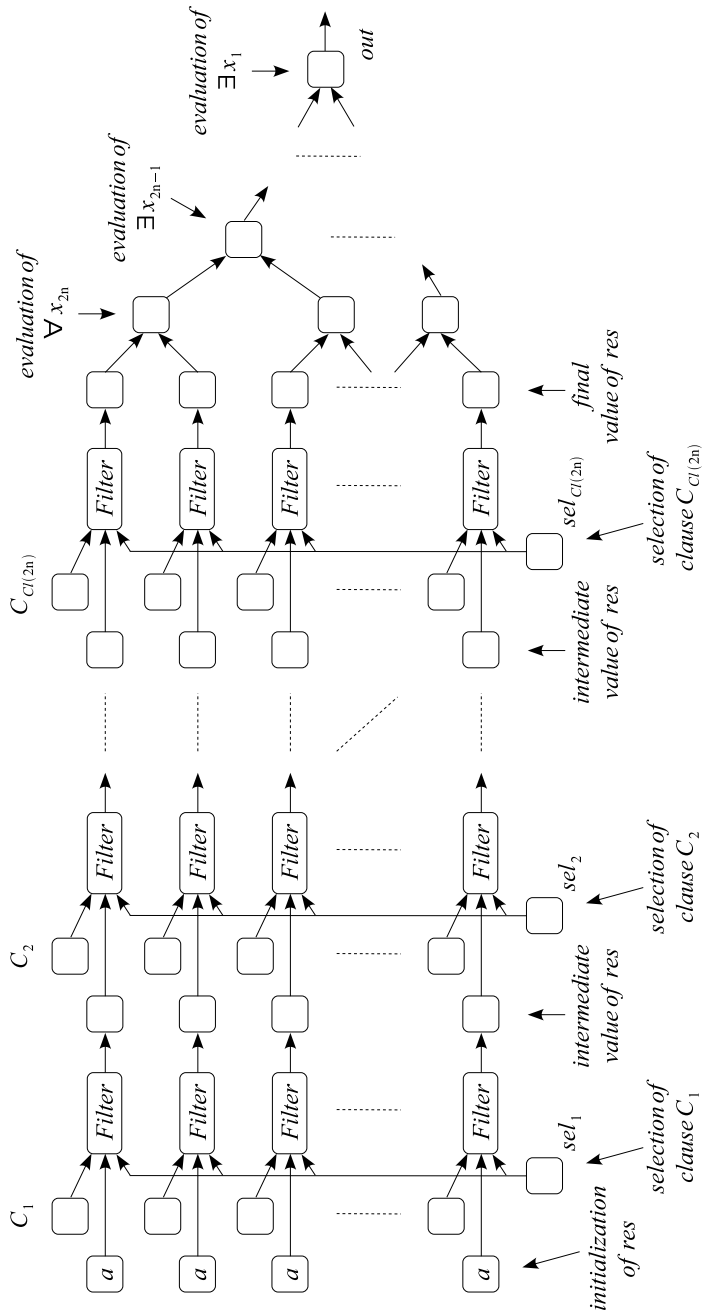


Fig. 5. Sketch of a deterministic SN P system that solves all possible instances of $Q3SAT(2n)$

Figure 5 outlines an SN P system which can be used to solve any instance $\gamma_{2n,m}$ of $Q3SAT(2n)$. The input to this system is once again the instance of `Q3SAT` we want to solve, but this time such an instance is given by specifying — among all the possible clauses that can be built using n Boolean variables — which clauses occur in the instance. The selection is performed by putting (in parallel, in the initial configuration of the system) one spike in each of the input neurons $sel_1, sel_2, \ldots, sel_{Cl(2n)}$ that correspond to the selected clauses.

The system is a simple modification of the one used in [8] to uniformly solve all the instances of the **NP**-complete problem `3-SAT` of a given size. To see how it works, let us consider the family $\{M^{(2n)}\}_{n\in\mathbb{N}}$ of Boolean matrices, where $M^{(2n)}$ has 2^{2n} rows — one for each possible assignment to the variables $x_1, x_2, \ldots, x_{2n}$ — and one column for each possible 3-clause that can be built using the same variables. As stated above, the number of columns is $Cl(2n) \in \Theta(n^3)$, a polynomial quantity in n. In order to make the construction of the matrix $M^{(2n)}$ as regular as possible, we could choose to list all the 3-clauses in a predefined order; however, our result is independent of any such particular ordering, and hence we will not bother further with this detail. For every $j \in \{1, 2, 3, \ldots, 2^{2n}\}$ and $i \in \{1, 2, \ldots, Cl(2n)\}$, the

x_1	x_2	x_3	x_4	$\cdots$	$x_1 \vee x_2 \vee \neg x_4$	$\cdots$	$\neg x_1 \vee \neg x_2 \vee x_3$	$\cdots$
0	0	0	0	$\cdots$	1	$\cdots$	1	$\cdots$
0	0	0	1	$\cdots$	0	$\cdots$	1	$\cdots$
0	0	1	0	$\cdots$	1	$\cdots$	1	$\cdots$
0	0	1	1	$\cdots$	0	$\cdots$	1	$\cdots$
0	1	0	0	$\cdots$	1	$\cdots$	1	$\cdots$
0	1	0	1	$\cdots$	1	$\cdots$	1	$\cdots$
0	1	1	0	$\cdots$	1	$\cdots$	1	$\cdots$
0	1	1	1	$\cdots$	1	$\cdots$	1	$\cdots$
1	0	0	0	$\cdots$	1	$\cdots$	1	$\cdots$
1	0	0	1	$\cdots$	1	$\cdots$	1	$\cdots$
1	0	1	0	$\cdots$	1	$\cdots$	1	$\cdots$
1	0	1	1	$\cdots$	1	$\cdots$	1	$\cdots$
1	1	0	0	$\cdots$	1	$\cdots$	0	$\cdots$
1	1	0	1	$\cdots$	1	$\cdots$	0	$\cdots$
1	1	1	0	$\cdots$	1	$\cdots$	1	$\cdots$
1	1	1	1	$\cdots$	1	$\cdots$	1	$\cdots$
Assignments				Clauses				

Fig. 6. An excerpt of matrix $M^{(4)}$. On the left we can see the assignments which are associated to the corresponding rows of the matrix. Only the columns corresponding to the clauses $x_1 \vee x_2 \vee \neg x_4$ and $\neg x_1 \vee \neg x_2 \vee x_3$ are detailed

element $M^{(2n)}_{ji}$ is equal to 1 if and only if the assignment associated with row j satisfies the clause associated with column i. Figure 6 shows an excerpt of matrix $M^{(4)}$, where each row has been labelled with the corresponding clause; only the

```
CHECK SATISFIABILITY(M^(2n))
res ← [1 1 ··· 1]        // 2^2n elements
for all columns C in M^(2n)
    do if C corresponds to a selected clause
          then res ← res ∧ C        // bit-wise AND
return res
```

Fig. 7. Pseudocode of the algorithm used to select the assignments that satisfy all the clauses of $\gamma_{2n,m} \in Q3SAT(2n)$

columns that correspond to clauses $x_1 \vee x_2 \vee \neg x_4$ and $\neg x_1 \vee \neg x_2 \vee x_3$ are shown in details.

Let us now consider the algorithm given in pseudocode in Figure 7. The variable res is a vector of length 2^{2n}, whose components — which are initialized to 1 — are bijectively associated with all the possible assignments of $x_1, x_2, \ldots, x_{2n}$. The components of res are treated as flags: when a component is equal to 1, it indicates that the corresponding assignment satisfies all the clauses which have been examined so far. Initially we assume that all the flags are 1, since we do not have examined any clause yet. The algorithm then considers all the columns of $M^{(2n)}$, one by one. If the column under consideration does not correspond to a selected clause, then it is simply ignored. If, on the other hand, it corresponds to a clause which has been selected as part of the instance, then the components of res are updated, putting to 0 those flags that correspond to the assignments which do not satisfy the clause. At the end of this operation, which can be performed in parallel on all the components, only those assignments that satisfy all the clauses previously examined, as well as the clause currently under consideration, survive the filtering process. After the last column of $M^{(2n)}$ has been processed, we have that the components of vector res indicate those assignments that satisfy all the clauses of the instance $\gamma_{2n,m}$ of Q3SAT given as input. Stated otherwise, res is the output column of the truth table of the unquantified propositional formula contained in $\gamma_{2n,m}$.

This algorithm can be easily transformed into an exponential size Boolean circuit, that mimics the operations performed on the matrix $M^{(2n)}$, described by the pseudocode given in Figure 7. Such a circuit can then be easily simulated using the SN P system that we have outlined in Figure 5 (precisely, the left side of the system, until the column of neurons that contain the final value of vector res). This part of the system is composed of three layers for each possible 3-clause that can be built using $2n$ Boolean variables. Two of these layers are used to store the intermediate values of vector res and the values contained in the columns of $M^{(2n)}$, respectively. The third layer, represented by the boxes marked with *Filter* in Figure 5, transforms the current value of res to the value obtained by applying

the corresponding iteration of the algorithm given in Figure 7. This layer is in turn composed by three layers of neurons, as we will see in a moment.

The last part of the system is used to check the satisfiability of the universal and existential quantifiers associated with the variables $x_1, x_2, \ldots, x_{2n}$. The neurons in this part of the system compose a binary tree of depth $2n$; the first layer of neurons corresponds to the bottom of the tree, and checks the satisfiability of the quantifier $\forall x_{2n}$; the second layer checks the satisfiability of $\exists x_{2n-1}$ and so on, until the last layer, whose only neuron is σ_{out} (the output neuron), that checks the satisfiability of $\exists x_1$. To see how the check is performed, let us consider the fully quantified

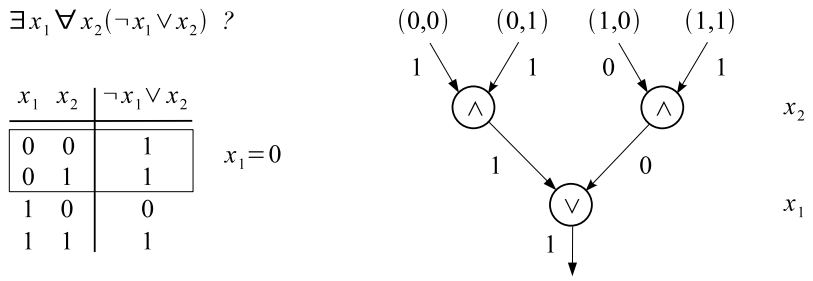


Fig. 8. Example of a quantified Boolean formula formed by one clause, built using two Boolean variables. On the left, its truth table is reported with an indication of the truth assignments that make the formula *true*. On the right, the tree which is used to check the satisfaction of the quantifiers $\forall$ and $\exists$ is depicted

formula $\exists x_1 \forall x_2 (\neg x_1 \vee x_2)$. This formula is composed of a single 2-clause (hence it is not a valid instance of `Q3SAT`), built using two Boolean variables. In Figure 8 we can see the truth table of the clause, and an AND/OR Boolean circuit that can be used to check whether the quantifiers associated with x_1 and x_2 are satisfied. This circuit is a binary tree whose nodes are either AND or OR gates. Each layer of nodes is associated with a Boolean variable: precisely, the output layer is associated to x_1, the next layer to x_2, and so on until the input layer, which is associated to x_n. If $Q_i = \forall$ then the nodes in the layer associated with x_i are AND gates; on the contrary, if $Q_i = \exists$ then the nodes in such a layer are OR gates. The input lines of the circuit are bijectively associated to the set of all possible truth assignments. It is not difficult to see that when these input lines are fed with the values contained in the output column of clause's truth table, the output of the circuit is 1 if and only if the fully quantified formula is *true*. Since in the first part of the system we have computed the output column of the truth table of the unquantified propositional formula contained in $\gamma_{2n,m}$, we just have to feed these values to an SN P system that simulates the above AND/OR Boolean circuit to see whether $\gamma_{2n,m}$ is *true* or not. Implementing such a Boolean circuit by means of an SN P system is trivial: to see how this can be done, just compare the last two layers of the circuits illustrated in Figures 10 and 11.

The overall system then works as follows. During the computation, spikes move from the leftmost to the rightmost layer. One spike is expelled to the environment by neuron σ_{out} if and only if $\gamma_{2n,m}$ is *true*. In the initial configuration, every neuron in the first layer (which is bijectively associated with one of the 2^{2n} possible assignments to the Boolean variables $x_1, x_2, \ldots, x_{2n}$) contains one spike, whereas neurons $sel_1, sel_2, \ldots, sel_{Cl(2n)}$ contain one or zero spikes, depending upon whether or not the corresponding clause is part of the instance $\gamma_{2n,m}$ given as input. Stated otherwise, the user must provide one spike — in the initial configuration of the system — to every input neuron sel_i that corresponds to a clause that has to be selected. In order to deliver these spikes at the correct moment to all the filters that correspond to the ith iteration of the algorithm, every neuron sel_i contains the rule $a \to a; 4(i-1)$, whose delay is proportional to i. In order to synchronize the execution of the system, also the neurons that correspond to the ith column of $M^{(2n)}$ deliver their spikes simultaneously with those distributed by neurons sel_i, using the same rules. An alternative possibility is to provide the input to the system in a sequential way, for example as a bit string of length $Cl(2n)$, where a 1 (resp., 0) in a given position indicates that the corresponding clause has to be selected (resp., ignored). In this case we should use a sort of delaying subsystem, that delivers — every four time steps — the received spike to all the neurons that correspond to the column of $M^{(2n)}$ currently under consideration. Since the execution time of our algorithm is proportional to the number $Cl(2n)$ of all possible clauses containing $2n$ Boolean variables, this modification keeps the computation time of the entire system cubic with respect to n.

In the first computation step, all the inputs going into the first layer of filters are ready to be processed. As the name suggests, these filters put to 0 those flags which correspond to the assignments that do not satisfy the first clause (corresponding to the first column of $M^{(2n)}$). This occurs only if the clause has been selected as part of the instance $\gamma_{2n,m} \in Q3SAT(2n)$ given as input, otherwise all the flags are kept unchanged, ready to be processed by the next layer of filters. In either case, when the resulting flags have been computed they enter into the second layer of filters together with the values of the second column of $M^{(2n)}$, and the input sel_2 that indicates whether this column is selected or not as being part of the instance. The computation proceeds in this way until all the columns of $M^{(2n)}$ have been considered, and the resulting flags (corresponding to the final value of vector res in the pseudocode of Figure 7) have been computed.

Before looking at how the system checks the satisfiability of the universal and existential quantifiers $\exists x_1, \forall x_2, \ldots, \forall x_{2n}$, let us describe in detail how the filtering process works. This process is performed in parallel on all the flags: if the clause C_i has been selected then an AND is performed between the value $M_{ji}^{(2n)}$ (that indicates whether the jth assignment satisfies C_i) and the current value of the flag res_j (the jth component of res); as a result, res_j is 1 if and only if the jth assignment satisfies all the selected clauses which have been considered up to now. On the other hand, if the clause C_i has not been selected then the old value of res_j is kept unaltered. This filtering process can be summarized by the pseudocode

```
FILTER(sel_i, res_j, C_i)
if sel_i = 0 then return res_j
             else return res_j ∧ C_i
```

Fig. 9. Pseudocode of the Boolean function computed by the blocks marked with FILTER in Figure 5

given in Figure 9, which is equivalent to the following Boolean function:

$$(\neg sel_i \wedge res_j) \vee (sel_i \wedge res_j \wedge C_i)$$

Such a function can be computed by the Boolean circuit depicted in Figure 10, that in turn can be simulated by the SN P system illustrated in Figure 11. Note the system represented in this latter figure is a generic module which is used many times in the whole system outlined in Figure 5, hence we have not indicated the delays which are needed in neurons sel_i and C_i. Also neuron 1, which is used to negate the value emitted by neuron sel_i, must be activated together with sel_i, that is, after $4(i-1)$ steps after the beginning of the computation. The spike it contains can be reused in the namesake neuron that occurs in the next layer of filters.

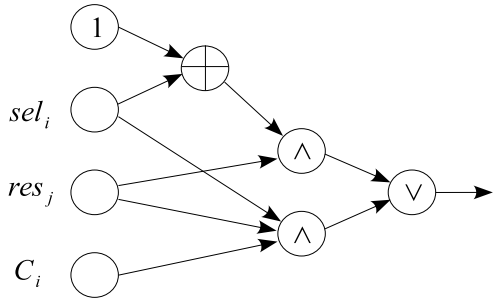


Fig. 10. The Boolean circuit that computes the function FILTER whose pseudocode is given in Figure 9

The last part of the system illustrated in Figure 5 is devoted to check the satisfiability of the universal and existential quantifiers $\exists x_1, \forall x_2, \ldots, \forall x_{2n}$ associated to the Boolean variables $x_1, x_2, \ldots, x_{2n}$. As we have seen, the final values of vector res represent the output column of the truth table of the unquantified propositional formula contained in $\gamma_{2n,m}$. Hence, to check whether all the universal and existential quantifiers are satisfied, it suffices to feed these values as input to an SN P system that simulates a depth $2n$ AND/OR Boolean circuit that operates as described in Figure 8. Each gate is simply realized as a neuron that contains

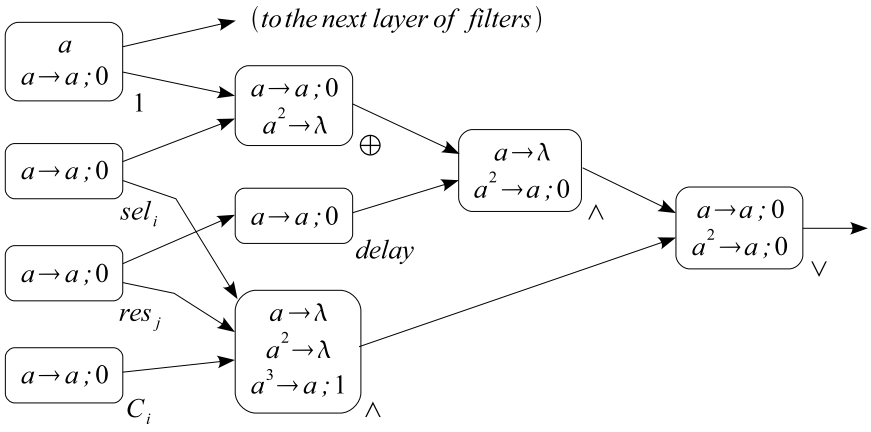


Fig. 11. An SN P system that computes the function FILTER given in Figure 9, simulating the Boolean circuit of Figure 10

two rules, as depicted in the last two layers of Figure 11. At each computation step, one quantifier is checked; when the check terminates, one spike is emitted to the environment by the output neuron σ_{out} if and only if the fully quantified formula $\gamma_{2n,m} \in Q3SAT(2n)$ given as input to the entire system is *true*. The total computation time of the system is proportional to the number $Cl(2n)$ of columns of $M^{(2n)}$, that is, $\Theta(n^3)$.

As we can see, the structure of the system that uniformly solves all the instances of $Q3SAT(2n)$ is very regular, and does not contain "hidden information". For the sake of regularity we have also omitted some possible optimizations, that we briefly mention here. The first column of neurons in Figure 5 corresponds to the initial value of vector res in the pseudocode given in Figure 7. Since this value is fixed, we can pre-compute part of the result of the first step of computation, and remove the entire column of neurons from the system. In a similar way we can also remove the subsequent columns that correspond to the intermediate values of res, and send these values directly to the next filtering layer. A further optimization concerns the values $M_{ji}^{(2n)}$, which are contained in the neurons labelled with C_i. Since these values are given as input to AND gates, when they are equal to 1 they can be removed since they do not affect the result; on the other hand, when they are equal to 0 also the result is 0, and thus we can remove the entire AND gate.

5 Conclusions and Remarks

In this paper we have shown that QSAT, a well known **PSPACE**-complete problem, can be deterministically solved in linear time with respect to the number n of variables and the number m of clauses by an exp-uniform family of SN P systems

with pre-computed resources. We have also considered `Q3SAT`, a restricted (but still **PSPACE**-complete) version of `QSAT` in which all the clauses of the instances have exactly three literals; we have shown that in this case the problem can be solved in a time which is at most cubic in n, independent of m. Each pre-computed SN P system of the family can be used to solve all the instances of `QSAT` (or `Q3SAT`), expressed in a normalized form, of a given size.

Note that using pre-computed resources in spiking neural P systems is a powerful technique, that simplifies the solution of computationally hard problems. The pre-computed SN P systems presented in this paper have an exponential size with respect to n but, on the other hand, have a regular structure. It still remains open whether such pre-computed resources can be constructed in a regular way by using appropriate computation devices that, for example, use a sort of controlled duplication mechanism to produce an exponential size structure in a polynomial number of steps. A related interesting problem is to consider whether alternative features can be introduced in SN P systems to uniformly solve **PSPACE**-complete problems. Nondeterminism is the first feature that comes to mind, but this is usually considered too powerful in the Theory of Computation.

Acknowledgements

Valuable comments and suggestions from professor Mario J. Pérez-Jiménez are greatly appreciated. This work was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60803113), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). The first author was partially supported by Academy of Finland, project 122426. The second author was partially supported by MIUR project "Mathematical aspects and emerging applications of automata and formal languages" (2007).

References

1. J.L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity*, Voll. I and II, Springer-Verlag, Berlin, 1988–1990.
2. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75 (2007), 141–162.
3. H. Chen, M. Ionescu, T.-O. Ishdorj, On the efficiency of spiking neural P systems. *Fourth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh, Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), Sevilla, January 30– February 3, Sevilla University, Fénix Editora, 2006, 195–206.
4. 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*, 7, 2 (2008), 147–166.

5. M.R. Garey, D.S. Johnson: *Computers and Intractability. A Guide to the Theory of* **NP**-*Completeness*. W.H. Freeman and Company, San Francisco, 1979.
6. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, D. Ramírez-Martínez: A software tool for verification of spiking neural P systems. *Natural Computing*, 7, 4 (2008), 485–497.
7. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2–3 (2006), 279–308.
8. T.-O. Ishdorj, A. Leporati: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. *Natural Computing*, 7, 4 (2008), 519–534.
9. A. Leporati, M.A. Gutiérrez-Naranjo: Solving Subset Sum by spiking neural P systems with pre-computed resources. *Fundamenta Informaticae*, 87, 1 (2008), 61–77.
10. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez: Uniform solutions to SAT and Subset Sum by spiking neural P systems. *Natural Computing*, online version (DOI: 10.1007/s11047-008-9091-y).
11. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical **NP**-complete problem with spiking neural P systems. *Membrane Computing, International Workshop, WMC8* (G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Thessaloniki, Greece, 2007, LNCS 4860, Springer-Verlag, Berlin, 2007, 336–352.
12. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: On the computational power of spiking neural P systems. *International Journal of Unconventional Computing*, to appear.
13. A. Păun, Gh. Păun: Small universal spiking neural P systems. *BioSystems*, 90, 1 (2007), 48–60.
14. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 1, 61 (2000), 108–143. See also Turku Centre for Computer Science – TUCS Report No. 208, 1998.
15. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
16. A. Salomaa: *Formal Languages*. Academic Press, New York, 1973.
17. The P systems Web page: http://ppage.psystems.eu

On the Power of Insertion P Systems of Small Size

Alexander Krassovitskiy

Research Group on Mathematical Linguistics,
Rovira i Virgili University
Av. Catalunya, 35, Tarragona 43002, Spain
alexander.krassovitskiy@estudiants.urv.cat

Summary. In this article we investigate insertion systems of small size in the framework of P systems. We consider P systems with insertion rules having one symbol context and we show that they have the computational power of matrix grammars. If contexts of length two are permitted, then any recursively enumerable language can be generated. In both cases an inverse morphism and a weak coding were applied to the output of the corresponding P systems.

1 Introduction

The study of insertion-deletion operations on strings has a long history. We just mention [18], [5], [7], [13], [15]. Motivated from molecular computing they have been studied in [1], [6], [17], [19], [12]. With some linguistic motivation they may be found in [11] and [3].

In general form, an insertion operation means adding a substring to a given string in a specified (left and right) context, while a deletion operation means removing a substring of a given string from a specified (left and right) context. A finite set of insertion/deletion rules, together with a set of axioms provide a language generating device: starting from the set of initial strings and iterating insertion-deletion operations as defined by the given rules we get a language. The number of axioms, the length of the inserted or deleted strings, as well as the length of the contexts where these operations take place are natural descriptional complexity measures of the insertion-deletion systems.

Some combinations of parameters lead to systems which are not computationally complete [14], [8] or even decidable [20]. It was shown that the operations of insertion and deletion considered in P systems framework can easily increase the computational power with respect to ordinary insertion-deletion systems [9], [10].

Traditionally, language generating devices having only insertion rules were studied. Early computational models based only on insertion appear already in [11], and are discussed in [17] and [16] (with membrane tree structure). It was

proved that pure insertion systems having one letter context are always context-free. Yet, there are insertion systems with two letter context which generate non-semilinear languages (see Theorem 6.5 in [17]). On the other hand, it appears that by using only insertion operation the obtained language classes with context greater than one are incomparable with many known language classes. For example there is a simple linear language $\{a^n ba^n \mid n \geq 1\}$ which cannot be generated by any insertion system (see Theorem 6.6 in [17]).

In order to overcome this obstacle one can use some codings to "interpret" generated strings. In [17], in a natural way, there were used two additional string operations: a morphism h and a weak coding φ. The result is considered as a product of application $h^{-1} \circ \varphi$ on the generated strings. Clearly, the obtained languages have greater expressivity and the corresponding language class is more powerful. It appears that with the help of morphisms and codings one can obtain every RE language if insertion rules have sufficiently large context. It is proved in [17] that for every recursively enumerable language L there exists a morphism h, a weak coding φ and a language L' generated by an insertion system with rules using the length of the contexts at most 7, such that, $L = h(\varphi^{-1}(L'))$. The result was improved in [2], showing that rules having at most 5 letter context are sufficient to encode every recursively enumerable language. Recently, in [4] it was shown that the same result can be obtained with the length of contexts equal to 3.

In this article we consider the encoding as a part of insertion P systems. The obtained model is quite powerful and has the power of matrix languages if contexts of length one are used. We also show that if no encoding is used, then the corresponding family is strictly included in MAT and equals CF if no membranes are used. If an insertion of two symbols in two letters contexts is used, then all recursively enumerable languages can be generated (of course, using the inverse morphism and the weak coding).

2 Prerequisites

All formal language notions and notations we use here are elementary and standard. The reader can consult any of the many monographs in this area (see details, e.g., in [18]).

We denote by $|w|$ the length of a word w and by $card(A)$ the cardinality of the set A, and ε denotes the empty string.

An *insertion-deletion system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V, and I, D are finite sets of triples of the form (u, α, v), where u, α, and v are strings over V. The elements of T are *terminal* letters (in contrast, those of $V \backslash T$ are called nonterminals), those of A are *axioms*, the triples in I are *insertion rules*, and those from D are *deletion rules*. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted in between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be

removed from the context (u, v). As stated otherwise, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \to u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \to uv$. We denote by $\Longrightarrow_{ins}$ the relation defined by an insertion rule (formally, $x \Longrightarrow_{ins} y$ iff $x = x_1uvx_2, y = x_1u\alpha vx_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$) and by $\Longrightarrow_{del}$ the relation defined by a deletion rule (formally, $x \Longrightarrow_{del} y$ iff $x = x_1u\alpha vx_2, y = x_1uvx_2$, for some $(u, \alpha, v) \in D$ and $x_1, x_2 \in V^*$). We refer by $\Longrightarrow$ to any of the relations $\Longrightarrow_{ins}, \Longrightarrow_{del}$, and denote by $\Longrightarrow^*$ the reflexive and transitive closure of $\Longrightarrow$ (as usual, $\Longrightarrow^+$ is its transitive closure).

The language generated by ID is defined by

$$L(ID) = \{w \in T^* \mid x \Longrightarrow^* w, x \in A\}.$$

A (pure) *insertion systems* of weight (n, m, m') is a construct $ID = (V, A, I)$, where V is a finite alphabet, $I \subseteq V^*$ is a finite set of axioms, I is a finite set of insertion rules of the form (u, α, v), for $u, \alpha, v \in V^*$, and

$$n = \max\{|\alpha| \mid (u, \alpha, v) \in I\},$$
$$m = \max\{|u| \mid (u, \alpha, v) \in I\},$$
$$m' = \max\{|v| \mid (u, \alpha, v) \in I\}.$$

We denote by $INS_n^{m,m'}$ corresponding families of languages generated by insertion systems.

An *insertion P system* is the following construct:

$$\Pi = (V, \mu, M_1, \ldots, M_k, R_1, \ldots, R_k),$$

where

- V is a finite alphabet,
- μ is the membrane (tree) structure of the system which has n membranes (nodes). This structure will be represented by a word containing correctly nested marked parentheses.
- M_i, for each $1 \leq i \leq k$, is a finite language associated to the membrane i.
- R_i, for each $1 \leq i \leq k$, is a set of insertion rules with target indicators associated to membrane i and which have the following form: $(u, x, v; tar)$, where (u, x, v) is an insertion rule, and tar, called the *target indicator*, is from the set $\{here, in_j, out\}, 1 \leq j \leq k$.

Any k-tuple $(N_1, \ldots, N_k)$ of languages over V is called a configuration of Π. For two configurations $(N_1, \ldots, N_k)$ and $(N'_1, \ldots, N'_k)$ of Π we write $(N_1, \ldots, N_k) \Longrightarrow (N'_1, \ldots, N'_k)$ if we can pass from $(N_1, \ldots, N_k)$ to $(N'_1, \ldots, N'_k)$ by applying nondeterministically the insertion rules, to all possible strings from the corresponding regions, and following the target indications associated with the rules. We assume that every string represented in a membrane has arbitrary many copies. Hence, by applying a rule to a string we get both arbitrary many copies of resulted string as well as old copies of the same string. More specifically, if $w \in N_i$

and $r = (u, x, v; tar) \in R_i$, such that $w \Longrightarrow^r w'$ then w' will go to the region indicated by tar. If $tar = here$, then the string remains in N_i, if $tar = out$, then the string is moved to the region immediately outside the membrane i (maybe, in this way the string leaves the system), if $tar = in_j$, then the string is moved to the region j.

A sequence of transitions between configurations of a given insertion P system Π, starting from the initial configuration $(M_1, \ldots, M_n)$, is called a computation with respect to Π. The result of a computation consists of all strings over V which are sent out of the system at any time during the computation. We denote by $L(\Pi)$ the language of all strings of this type. We say that $L(\Pi)$ is generated by Π.

The *insertion-deletion tissue P systems* are defined in an analogous manner. As the tissue P systems use arbitrary graph structure we write the target indicator in the form $tar = go_j, j = 1, \ldots, k$. The result of a computation consists of all strings over V which are sent to one selected output cell.

The *weight* of insertion rules (n, m, m') and the membrane *degree* k describe the complexity of an insertion P system. We denote by $LSP_k(ins_n^{m,m'})$ (see, for example [16]) the family of languages $L(\Pi)$ generated by insertion P systems of degree at most $k \geq 1$ having the weight at most (n, m, m'). If some of the parameters n, m, m', or k is not specified we write " * " instead.

We say that a language L' is from $hINS_n^{m,m'}$ (from $hLSP_k(ins_n^{m,m'})$, correspondingly) if there exist a morphism h, weak coding φ and $L(\Pi) \in INS_n^{m,m'}$ ($L(\Pi) \in LSP_k(ins_n^{m,m'})$ such that $\varphi(h^{-1}(L(\Pi))) = L'$.

We write an instance of the system $hLSP$ in the form

$$(V, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n, h, \varphi),$$

where

- h is a morphism $h : V \to V^+$,
- φ is a weak coding $\varphi : T \to T \cup \{\varepsilon\}$,
- other components are defined as for insertion P system.

We insert "t" before P to denote classes corresponding to the tissue cases (e.g., $hLStP$). Insertion (t) holds for both tissue and tree membrane structure.

We say that a letter a is *marked* in a sentential form waw' if it is followed by #, i.e., $|w'| > 0$, and # is the prefix of w'. In the following proofs we use a marking technique introduced in [16]. The technique works as follow: in order to simulate rewriting production $A \to B$ we add adjacently right from A the word $\#B$ specifying that letter A is already rewritten. And, as soon as the derivation is completed, every pair $A\#$ in the sentential form is subject to the inverse morphism.

3 Main Results

Let us consider insertion systems (without membranes) with one letter context rules $hINS_*^{1,1}$. Applying the marking technique we get a characterization of context-free languages.

Theorem 1 $hINS_*^{1,1} = CF$

Proof. First we show that $CF \subseteq hINS_3^{1,1}$.

Let $G = (V, T, S, P)$ be a context-free grammar in Chomsky normal form. Consider the following system from $hINS_3^{1,1}$

$$\Pi = (T \cup V \cup \{\#\}, R, \{S\}, h, \varphi),$$

where $R = \{(A, \#BC, \alpha) \mid \alpha \in T \cup V, A \to BC \in P\}$, the morphism h

$$h(a) = a\#, \text{ if } a \in V, \text{ and } h(a) = a, \text{ if } a \in T,$$

and the weak coding φ

$$\varphi(a) \to \varepsilon, \text{ if } a \in V, \varphi(a) \to a, \text{ if } a \in T.$$

We claim that $L(\Pi) = L(G)$. Indeed, each rule $(A, \#BC, \alpha) \in R$ can be applied in the sentential form $wA\alpha w'$ if A is unmarked (not rewritten). Thus, the production $A \to BC \in P$ can be applied in the corresponding derivation G. Hence, by applying the counterpart rules we get equivalent derivations.

At the end of derivation, by applying the inverse morphism h^{-1} we warranty that every nonterminal is marked. Finally, we delete every nonterminal by the weak coding φ. Hence $L(\Pi) = L(G)$, and we get $CF \subseteq hINS_3^{1,1}$.

The equivalence of the two classes follows from Theorem 6.4 in [17] stating $INS_*^{1,1} \subseteq CF$ and the fact that context-free languages are closed under inverse morphisms and weak codings.

Now we consider insertion systems with contexts controlled by membranes (P systems). It is known from Theorem 5.5.1 in [16] that $LSP_2(ins_2^{1,1})$ contains non context-free languages. We show that this class is bounded by matrix grammars:

Lemma 2 $LStP_*(ins_*^{1,1}) \subset MAT$.

Proof. The proof uses the similar technique presented in [17], Theorem 6.4 for context-free grammars.

Let $\Pi = (V, \mu, M_1, \ldots, M_n, R_1, \ldots, R_n)$ be a system from $LStP_n(ins_*^{1,1})$ for some $n \geq 1$.

Consider a matrix grammar $G = (D \cup Q \cup \{S\}, V, S, P)$, where $Q = \{Q_i \mid i = 1, \ldots, n\}$, $D = \{D_{a,b} \mid a, b \in V \cup \{\varepsilon\}\}$, and P is constructed as follows:

1. For every rule $(a, b_1 \dots b_k, c, go_j) \in R_i, a, b_1, \dots, b_k, c \in V \cup \{\varepsilon\}, k > 0$ we add to P $(Q_i \to Q_j, D_{\overline{a},\overline{c}} \to D_{\overline{a},b_1} D_{b_1,b_2} \dots D_{b_{k-1},b_k} D_{b_k,\overline{c}})$, where

$$\overline{a} = \begin{cases} a, & \text{if } a \in V, \\ t, \forall t \in V \cup \{\varepsilon\}, & \text{if } a = \varepsilon \end{cases} \quad \overline{c} = \begin{cases} c, & \text{if } c \in V, \\ t, \forall t \in V \cup \{\varepsilon\}, & \text{if } c = \varepsilon \end{cases}$$

2. For every rule $(a, \varepsilon, c, in_j) \in R_i, a, c \in V \cup \{\varepsilon\}, k > 0$ we add to P $(Q_i \to Q_j, D_{\overline{a},\overline{c}} \to D_{\overline{a},\overline{c}})$, where $\overline{a}$ and $\overline{c}$ defined as in the previous case.
3. Next, for every $w = b_1 \cdots_k \in M_i, i = 1, \dots, n, k > 0$ we add to P the matrix $(S \to Q_i D_{\varepsilon,b_1} D_{b_1,b_2} \dots D_{b_{k-1},b_k} D_{b_k,\varepsilon})$.
4. In special case if $\varepsilon \in M_i$ we add $(S \to Q_i D_{\varepsilon,\varepsilon})$ to P.
5. Also, for every $D_{a,b} \in D, a, b \in V \cup \{\varepsilon\}$ we add $(D_{a,b} \to a)$ to P.
6. Finally, we add $(p_1 \to \varepsilon)$ to P (we assume that the first cell is the output cell).

The simulation of Π by the matrix grammar is straightforward. We store the label of current cell by nonterminals Q. Every nonterminal $D_{a,c} \in D, a, c \in V \cup \{\varepsilon\}$, represents a pair of adjacent letters, so we can use them as a context. A rule $(a, b_1 \dots b_k, c, in_j) \in R_i, a, c \in V, b_1 \dots b_k \in V^k$, can be simulated by the grammar iff the sentential form contains both Q_i and $D_{a,c}$. It results the label of current cell is rewritten to Q_j and $D_{a,c}$ is rewritten to the string $D_{a,b_1} D_{b_1,b_2} \dots D_{b_{k-1},b_k} D_{b_k,a}$. Clearly, the string preserves one symbol context. In order to treat those rules which have no context we introduce productions that preserve arbitrary context ($\overline{a} \in V \cup \{\varepsilon\}$ and $\overline{c} \in V \cup \{\varepsilon\}$).

The simulation of the grammar starts with a nondeterministic choice of the axiom. Then, during the derivation any rule corresponding to the context (a, b) have to be applied (in a one to one correspondence with grammar productions). Finally, the string over V is produced by the grammar iff Q_1 has been deleted from the simulated sentential form. The deletion of Q_1 specifies that Π reached the output cell. So, we obtain $L(\Pi) = L(G)$. Hence, $LStP_*(ins_*^{1,1}) \subseteq MAT$.

The strictness of the inclusion follows from the fact there are languages from MAT which cannot be generated by any insertion P system from $LStP_*(ins_n^{1,1})$, for any $n \geq 1$. Indeed, consider $L_a = \{ca^k ca^k c \mid k \geq 1\}$. One may see that the matrix grammar $(\{S_l, S_r, S\}, \{a, b\}, S, P')$ generates L_a, where

$$\begin{aligned} P' = \{&(S \to cS_l cS_r c), \\ &(S_l \to \varepsilon, S_r \to \varepsilon), \\ &(S_l \to aS_l, S_r \to aS_r)\}. \end{aligned}$$

On the other hand, $L_a \notin LStP_*(ins_n^{1,1})$, for any $n \geq 1$. For the contrary, assume there is such a system. We note that the system cannot delete or rewrite any letter so every insertion is terminal. And, as the language of axioms is finite, we need an insertion rule of letter a. Consider alternatives for a final insertion step in a derivation which has at most one step and derives a word $ca^k ca^k c$, for some $k > n$: (1) the last applied rule inserts the central letter c, or (2) it does not insert the

central letter c. (1) The central c can be inserted between any two letters a. So we get a contradiction because the prefix ca^pc may not be equal to the suffix ca^qc.
(2) The last applied rule can (2.1) either insert the letter c(at the end or start of the string) or, (2.2) no c is inserted by the final rule.
(2.1) We get a contradiction because c can be alternatively inserted in between two a as we assumed $k > n$.
(2.2) The last rule cannot distinguish whether to insert a before the central c or after the central c. So, again, we get a contradiction because the prefix ca^pc may not be equal to the suffix ca^qc.

So we proved $L_a \notin LStP_*(ins_n^{1,1})$, for any $n \geq 1$ and hence $LStP_*(ins_*^{1,1}) \subset MAT$.

Corollary 3 $LSP_*(ins_*^{1,1}) \subset MAT$.

Proof. A tree is a special case of a graph.

Lemma 4 $MAT \subseteq hLSP_*(ins_2^{1,1})$.

Proof. We prove the theorem by a direct simulation of a matrix grammar $G = (N, T, S, P)$. We assume that G is in binary normal form, i.e., every matrix has the form $i : (A \to BC, A' \to B'C') \in P$, where $A, A' \in N, B, B', C, C' \in N \cup T \cup \{\varepsilon\}$ and $i = 1, \ldots, n$.

Consider a system $\Pi \in hLSP_{n+3}(ins_2^{1,1})$,

$$\Pi = (V, [_1\ [_2\ [_3\ \left(\prod_{i=1,\ldots,n} [_{i+3}\ ]_{i+3}\right] _3\ ]_2\ ]_1, \{S\$\}, \emptyset, \ldots, \emptyset, R_1, \ldots, R_{n+3}, h, \varphi),$$

where $V = N \cup T \cup \{C_i, C_i' \mid i = 1, \ldots, n\} \cup \{\#, \$\}$.

For every matrix $i : (A \to BC, A' \to B'C')$ we add

$r.1.1 : (A, \#C_i, \alpha, in_2)$,		to R_1;
$r.2.1 : (C_i, BC, \alpha, in_3)$,	$r.2.2 : (C_i', \#, \alpha, out)$	to R_2;
$r.3.1 : (C_i, \#, \alpha, in_{i+3})$,	$r.3.2 : (C_i', B'C', \alpha, out)$	to R_3;
$r.i + 3.1 : (A', \#C_i', \alpha, out)$,		to R_{i+3}

for every $\alpha \in V \backslash \{\#\}$. In addition we add $(\varepsilon, \$, \varepsilon, out)$ to R_1.

We also define the morphism h and the weak coding φ by:

$$h(a) = \begin{cases} a, & \text{if } a \in T, \\ a\# & \text{if } a \in V \end{cases}$$

$$\varphi(a) = \begin{cases} a, & \text{if } a \in T, \\ \varepsilon & \text{if } a \in V \cup \{\$\}. \end{cases}$$

We claim that $L(\Pi) = L(G)$. To do so it is enough to prove that $w \in L(G)$ iff $w' \in L(\Pi)$ and $w' = \varphi(h^{-1}(w))$.

First we show that for every $w \in L(G)$ there exists $w' \in L(\Pi)$ and $w' = \varphi(h^{-1}(w))$. Consider the simulation of the i-th matrix $(A \to BC, A' \to B'C') \in P$. The simulation is controlled by letters C_i and C'_i. First, we insert $\#C_i$ in the context of a unmarked A and send the obtained string to the second membrane. Then we use C_i as a context to insert adjacently right the word BC. After that, we mark the control letter C_i and send the sentential form to the $i+3$ membrane. Here we choose nondeterministically one letter A', mark it, write adjacently right new control letter C'_i, and, after that, send the obtained string to the third membrane. We mention that it is not possible to apply the rule $r.i+3.1 : (A', \#C'_i, \alpha; out)_e$ in the $i+3$ membrane and to reach the skin membrane if the sentential form does not contain the unmarked A'. So, this branch of computation cannot influence the result and may be omitted in the consideration. Next, in the third membrane, $B'C'$ is inserted in the context of unmarked C'_i and the sentential form is sent to the second membrane. Finally, we mark C'_i and send the resulting string back to the skin membrane.

We assume that at the beginning of this simulation the sentential form in the skin membrane does not contain unmarked C_i, C'_i. Hence, the insertions in the second and third membranes are deterministic. The derivation preserves the assumption, as after the sentential form is sent back to the skin membrane the introduced C_i, and C'_i are marked. At the end of computation we send the obtained sentential form out of the system by the rule $(\varepsilon, \$, \varepsilon, out)$.

Let w be a string in the skin region which contains some unmarked A and A'. If the letter A precedes B, then we can write $w = w_1 A \alpha_1 w_2 A' \alpha_1 w_3$. The simulation of the matrix is the following

$$w_1 A \alpha_1 w_2 A' \alpha_1 w_3 \overset{r.1.1, r.2.1, r.3.1}{\Longrightarrow} w_1 A \#C_i \#BC \alpha_1 w_2 A' \alpha_1 w_3 \overset{r.3+i.1, r.3.2, r.2.2}{\Longrightarrow} w_1 A \#C_i \#BC \alpha_1 w_2 A' \#C'_i \#B'C' \alpha_1 w_3,$$

where $w_1, w_2, w_3 \in V^*, \alpha_1, \alpha_2 \in V \backslash \{\#\}$. We can write the derivation similarly if B precedes A.

Hence, as a result of the simulation of i-th matrix we get both A and A' marked and BC, $B'C'$ inserted in the right positions. The derivation in Π may terminate by the rule $(\varepsilon, \$, \varepsilon, out)$ only in the first membrane. This guarantees that the simulation of each matrix has been finished. According to the definition of Π the string w' belongs to the language if $w' = \varphi(h^{-1}(w))$, where w is the generated string. This is the case only if the resulting output of Π does not contain unmarked nonterminals. Hence we proved $L(G) \subseteq \varphi(h^{-1}(L(\Pi)))$.

The inverse inclusion is obvious since every rule in Π has its counterpart in G. The case when the derivation in Π is blocked corresponds to the case a matrix cannot be finished.

Hence, we get $MAT \subseteq hLSP_*(ins_2^{1,1})$.

Remark 5 *One can mention that a similar result can be obtained with a smaller number of membranes at the cost of increasing the maximal length of inserted*

words. I.e., for any grammar G' from MAT there is a P insertion system Π' corresponding to $hLSP_{n+1}(ins_3^{1,1})$ such that $L(G) = L(\Pi')$, and n is the number of matrices in G'. To prove this we can use the same argument as in the previous theorem and replace rules $(r..*)$ by*

$$\begin{aligned} &(A, \#BC, \alpha, in_{i+1}), \alpha \in V\backslash\{\#\} && \text{to } R_1 \\ &(A', \#B'C', \alpha, in_1), \alpha \in V\backslash\{\#\} && \text{to } R_{i+1}. \end{aligned}$$

Corollary 6 $MAT \subseteq hLStP_*(ins_2^{1,1})$.

Proof. Obvious, since a tree is a special case of a graph.

Taking into account Lemma 4, Lemma 2, and the fact that the class of matrix grammars is closed under inverse morphisms and weak codings we get the following characterization of MAT:

Theorem 7 $hLS(t)P_*(ins_*^{1,1}) = MAT$.

Next we consider computationally complete insertion P systems. In order to use concise representations of productions in 0-type grammars we need an auxiliary lemma.

Lemma 8 *For every 0-type grammar $G' = (N', T, S', P')$ there exists a grammar $G = (N, T, S, P)$ such that $L(G) = L(G')$ and every production in P has the form*

$$\begin{aligned} AB \to AC \text{ or} \qquad & AB \to CB \text{ or} && (1) \\ A \to AC \text{ or} \qquad & A \to CA \text{ or} && (2) \\ A \to \delta, & && (3) \end{aligned}$$

where A, B and C are from N and $\delta \in T \cup N \cup \{\varepsilon\}$.

Proof. To prove the lemma it is enough to show that for any grammar in Penttonnen normal form there is an equivalent grammar having productions of the form (1)–(3). To do so it is enough to simulate the context-free productions $A \to BC$ by productions of the form (1)–(3).

Let $G = (N, T, S, P)$ be a grammar in Penttonnen normal form whose production rules in P are of the form:

$$\begin{aligned} AB \to AC \text{ or} \\ A \to BC \text{ or} \\ A \to \alpha \end{aligned}$$

where A, B, C and D are from N and $\alpha \in T \cup N \cup \{\varepsilon\}$.

Let $P_{CF} \subseteq P$ denotes the set of all context-free productions $A \to BC \in P$ such that $B \neq C$. Suppose that rules in P_{CF} are ordered and $n = card(P_{CF})$.

Consider a grammar $G' = (N', T, S, P')$, where $N' = N \cup \{X_i, Y_i, Z_i \mid i = 1, \ldots, n\}$, $P' = (P\backslash P_{CF}) \cup P'_{CF}$, and P'_{CF} is constructed as follows: for every $i : A \to BC \in P_{CF}$ add to P'_{CF} the following productions

$$r.i.1 : A \to X_i, \qquad r.i.2 : X_i \to X_iY_i,$$
$$r.i.3 : X_iY_i \to Z_iY_i, \qquad r.i.4 : Z_iY_i \to Z_iC$$
$$r.i.5 : Z_iC \to BC$$

Clearly, the obtained grammar has the form specified by (1)–(3). Now we prove that $L(G) = L(G')$. The inclusion $L(G) \subseteq L(G')$ is obvious as for every derivation in G we use its counterpart derivation in G' replacing i-th context-free production from P_{CF} by the sequence of productions $r.i.1, r.i.2, r.i.3, r.i.4, r.i.5$:

$$wAw' \overset{r.i.1}{\Longrightarrow} wX_iw' \overset{r.i.2}{\Longrightarrow} wX_iY_iw' \overset{r.i.3}{\Longrightarrow} wZ_iY_iw' \overset{r.i.4}{\Longrightarrow} wZ_iCw' \overset{r.i.5}{\Longrightarrow} wBCw'.$$

In order to prove that $L(G') \subseteq L(G)$ we show that for every terminal derivation in G' we can construct a derivation in G so that they both produce the same word. We use the counterpart productions from $P \backslash P_{CF}$ to mimic analogous production. For the productions $P_{CF'}$ we show that any deviation from the above defined sequence does not produce any new terminal derivation. First, we mention that the sequence of productions corresponding to $i : A \to BC$ starts by rewriting A on the new nonterminal, so, other productions not in $r.i.*$ cannot interfere the sequence. Yet, the production rule $r.i.2$ may generate extra Y_i (for simplicity, we assume one extra Y_i generated).

$$wAw' \overset{r.i.1}{\Longrightarrow} wX_iw' \overset{(r.i.2)^2}{\Longrightarrow} wX_i(Y_i)^2w' \overset{r.i.3,r.i.4}{\Longrightarrow} \overset{r.i.5}{\Longrightarrow} wBCY_iw'.$$

As we need to consider only terminal derivations we may assume that Y_i will be necessary rewritten. The only rule to rewrite Y_i is $r.i.4$. In order to perform it the letter Z_i must precede by the letter Y_i. It implies that the letter A must appear adjacently left from the letter Y_i. Then the sequence of productions $r.i.1, r.i.3, r.i.4, r.i.5$ results to the same sentential form as if $r.i.2$ is applied once per every rewriting of A.

$$wBCY_iw' \Longrightarrow^* w_1AY_iw_1' \overset{r.i.1}{\Longrightarrow} wX_iY_iw_1' \overset{r.i.3,r.i.4,r.i.5}{\Longrightarrow} w_1BCw_1'.$$

Therefore, it can be produced by rules from P_{CF}. We also mention that in $r.i.1 - r.i.5$ we start rewriting letters from $N \backslash N'$ by corresponding letters from N only after the letter X_i is rewritten. This imply that after $r.i.4$, we cannot insert additional Y_i adjacently left from C_i. So, Z_i can be rewritten unambiguously.

Finally, consider the case when the production $r.i.4$ is followed by some production from $P \backslash P_{CF}$ which rewrites C.

$$\ldots \overset{r.i.4}{\Longrightarrow} wZ_iC_iw' \Longrightarrow^* w_2Z_iC_iw_2' \overset{r.i.5}{\Longrightarrow} w_2BC_iw_2'.$$

As Z_i can be rewritten only if C_i appears to the right we may consider the equivalent derivation with the production $r.i.5$ applied directly after $r.i.4$. So the derivation is equivalent to the derivation with $i : A \to BC \in P_{CF}$. Hence we proved $L(G') \subseteq L(G)$, and hence $L(G') = L(G)$.

Every grammar in the normal form has the following property: every production can rewrite/add at most one (nonterminal) letter.

Now we increase the maximal size of the context of insertion rules to two letters. It is known from [17] that the class $INS_2^{2,2}$ contains non-semilinear languages. Considering these systems with membrane regulation we get

Theorem 9 $hLSP_3(ins_2^{2,2}) = RE$.

Proof. We prove the theorem by simulating a 0-type grammar in the normal form from Lemma 8. Let $G = (N, T, S, P)$ be such a grammar. Suppose that rules in P are ordered and $n = card(P)$.

Now consider the following insertion P system,

$$\Pi = (V, [_1\ [_2\ [_3\ ]_3\ ]_2\ ]_1, \{S\$\}, \emptyset, \emptyset, R_1, R_2, R_3, h, \varphi), \text{ where}$$

$V = T \cup N \cup F \cup \overline{F} \cup \{\#, \overline{\#}, \$\}$, $F = \{F_A, |A \in N\}$, $\overline{F} = \{\overline{F}_A, |A \in N\}$.

We include into R_1 the following rules:

$$\begin{aligned}
(AB, \#C, \alpha, here), &\text{ if } AB \to AC \in P;\\
(A, \#C, B\alpha, here), &\text{ if } AB \to CB \in P;\\
(A, C, \alpha, here), &\text{ if } A \to AC \in P;\\
(\varepsilon, C, A\alpha, here), &\text{ if } A \to CA \in P;\\
(A, \#\delta, \alpha, here), &\text{ if } A \to \delta \in P;\\
(\$, \varepsilon, \varepsilon, out),&
\end{aligned}$$

where $\alpha \in V \backslash \{\#\}$. It may happen that the pair of letters AB subjected to be rewritten by a production $AB \to AC$ or $AB \to CB \in R$ is separated by letters that have been marked. We use two additional membranes to transfer a letter over marked ones. In order to transfer $A \in N$ we add

$$r.1.1 : (A, \#F_A, \alpha, in_2), \alpha \in V \backslash \{\#\}$$

to the skin membrane. Then we add to the second membrane

$$\begin{aligned}
&r.2.1 : (F_A, \#A, \alpha', out), && r.2.2 : (\overline{F_A}, \overline{\#}A, \alpha', out),\\
&r.2.3 : (F_A X, \#\overline{F_A}, \#, in_3), && r.2.4 : (F_A\overline{F_B}, \overline{\#}F_A, \overline{\#}, in_3)\\
&r.2.5 : (\overline{F_A}X, \#\overline{F_A}, \#, in_3), && r.2.6 : (\overline{F_A}\ \overline{F_B}, \overline{\#}F_A, \overline{\#}, in_3)\\
&r.2.7 : (F_A\overline{\#}, F_A, \alpha, in_3), && r.2.8 : (\overline{F_A}\#, F_A, \alpha, in_3),\\
&r.2.9 : (F_A\overline{\#}, F_A, \overline{\#}, in_3), && r.2.10 : (\overline{F_A}\#, F_A, \overline{\#}, in_3),\\
&r.2.11 : (F_A\overline{\#}, \overline{F_A}, \#, in_3), && r.2.12 : (\overline{F_A}\#, \overline{F_A}, \#, in_3),
\end{aligned}$$

for every

$$X \in F \cup N, \overline{F_B} \in \overline{F}, \alpha \in V \backslash \{\#, \overline{\#}\},$$
$$\alpha' \in \{ab \mid a \in N \cup T, b \in N \cup T \cup \{\$\}\} \cup \{\$\}.$$

Finally, we add to the third membrane the rules

$$r.3.1 : (F_A, \#, \alpha, out), \alpha \in V \backslash \{\#\}, \qquad r.3.2 : (\overline{F_A}, \overline{\#}, \alpha, out), \alpha \in V \backslash \{\overline{\#}\}$$

The morphism h is defined by

$$h(a) = \begin{cases} a, & \text{if } a \in V \backslash N, \\ a\# & \text{if } a \in N. \end{cases}$$

The weak coding φ is defined by

$$\varphi(a) = \begin{cases} a, & \text{if } a \in T, \\ \varepsilon & \text{if } a \in V \backslash T. \end{cases}$$

We simulate the productions of P in the skin membrane by marking nonterminals from N and inserting corresponding letters of the productions. This is possible to do with insertion rules of weight $(2, 2, 2)$ since the grammar has such a form so every production rewrite/add at most one letter.

The simulation of the transfer is done in the second and the third membranes. The idea of the simulation is (1) to *mark* nonterminal we want to transfer, (2) *jump* over the marked letters with help of one special letter, at the end (3) *mark* the special letter and insert the original nonterminal. Since we use two letter contexts, in one step we can jump only over a single letter. Also we need to jump over the marking letter # as well as over marked nonterminals, and the letters inserted previously. We use letters $F_A \in F$ and $\overline{F_A} \in \overline{F}$ to keep information about the letter A we want to transfer. In order to jump over # we introduce one additional marking symbol $\overline{\#}$. We mark letters from $\overline{F}$ by $\overline{\#}$, and all other letters in $V \backslash \{\overline{\#}, \#\}$ by #. E.g., in a words $\overline{F_A}\#$, letter $\overline{F_A}$ is unmarked.

(1) The rule $r.1.1 : (A, \#F_A, \alpha, in_2)$, specifies that every unmarked letter from N may be used for the transfer.

(2) The rules $r.2.3 - r.2.12$ in the second membrane specify that F_A or $\overline{F_A}$ is copied to the right in such a way that the inserted letter would not be marked. In order to do so, the appropriate rule chooses to insert either the overlined copy $\overline{F_A}$ or the simple copy F_A. The rules $r.2.3 - r.2.6$ describe the possible jumps over one letter not in $\{\#, \overline{\#}\}$, and $r.2.7 - r.2.12$ describe the possible jumps over the marking letters #, $\overline{\#}$. These rules send the sentential form to the third membrane. The rules in the third membrane mark one symbol $F_A \in F$ or $\overline{F_A} \in \overline{F}$ and send the sentential form back to the second membrane.

(3) The rules $r.2.1$ and $r.2.2$ may terminate the transferring procedure and send the sentential form to the first membrane if letter \$ or two letters from $\{ab \mid a \in N \cup T, b \in N \cup T \cup \{\$\}\}$ appear in the the right context.

For example, consider the transfer of A in the string $AX\#C\$$ (here, we underline inserting substrings)

$$AX\#C\$ \overset{r.1.1}{\Longrightarrow} A\underline{\#F_A}X\#C\$ \overset{r.2.3}{\Longrightarrow} \$A\#F_AX\underline{\#\overline{F_A}}\#C\$ \overset{r.3.1}{\Longrightarrow}$$
$$A\#F_A\underline{\#}X\#\overline{F_A}\#C\$ \overset{r.2.6}{\Longrightarrow} A\#F_A\#X\#\overline{F_A}\underline{\#F_A}C\$ \overset{r.3.2}{\Longrightarrow}$$
$$A\#F_A\#X\#\overline{F_A}\ \underline{\overline{\#}}\#F_AC\$ \overset{r.2.1}{\Longrightarrow} A\#F_A\#X\#\overline{F_A}\ \overline{\#}\#F_A\underline{\#A}C\$$$

The sentential form preserves the following *invariant*:

- The first membrane does not contain unmarked letters from $F \cup \overline{F}$.
- There is exactly one unmarked letter $F \cup \overline{F}$ in the second membrane.
- There are always two unmarked letters from $F \cup \overline{F}$ in the third membrane.

We mention that the invariant is preserved by every derivation. Indeed, we start derivation from the axiom $S\$$ that satisfies the invariant, then one unmarked symbol is inserted by $r.1.1$. Rules $r.2.3 - r.2.12$ always add one more unmarked letter. And rules $r.2.1, r.2.2, r.3.1, r.3.2$ always mark one letter from $F \cup \overline{F}$.

In order to verify that Π simulates the same language as G we note that every reachable sentential form in G will be reachable also in Π by simulating the same production.

Also we note that the derivation in Π may terminate by the rule $(\$, \varepsilon, \varepsilon, out)$ only in the first membrane. Hence it guarantees that every transfer will be completed. It follows from the invariant that the simulation of the transfer is deterministic in the second membrane. There is a nondeterministic choice in the third membrane, where corresponding rules may mark one of the two unmarked letters. In the case the rule marks the rightmost letter, the derivation has to "jump" again over the inserted letter. The transfer satisfies the property that every terminating sequence replaces a nonterminal via arbitrary large string of marked letters, if it starts (by $r.1.1$) adjacently left from it. And in case the rule $r.1.1$ starts the transfer of a letter next to unmarked letter then it produces two marked symbols which do not affect on the result of the simulation.

The output string w is in the language only if $w' = \varphi(h^{-1}(w))$ is defined. This is the case only if the resulting output of Π does not contain unmarked nonterminals. On the other hand, every final derivation in Π has its counterpart in G. By applying the inverse morphism h^{-1} we filter out every sentential form with unmarked nonterminals from N. Hence, the corresponding derivation in G is terminated. Finally, the weak coding φ filters away supplementary letters. Hence we have $L(G) = L(\Pi)$.

Remark 10 *One may mention that there is a trade-off between the number of membranes and the maximal length of productions. By introducing additional nonterminals and fitting the grammar into the normal form we decrease the amount of used membranes. It is also the case in the other way: by growing the number of membranes we can simulate larger production rules.*

4 Conclusion

This article investigates the expressive power of insertion P systems with encodings. The length of insertion rules and number of membranes are used as a measure of descriptional complexity of the system. In the article we use the fact that morphisms and weak codings are incorporated to insertion P systems. The obtained family $hLS(t)P_*(ins_*^{1,1})$ serves to characterize matrix languages. When no membranes are used, the class $hINS_*^{1,1}$ equals the family of context-free languages. We proved the universality for the family $hLSP_*(ins_*^{2,2})$. More precisely, for every recursively enumerable language we can construct an insertion P system of weight $(2, 2, 2)$ and degree 3, so that applying an inverse morphism and a weak coding we generate the same language. Also, we want to mention that computational completeness of considered families indicates some trade-offs between descriptional complexity measures. Moreover, the descriptional complexity used in the paper may be extended by internal system parameters as, e.g., the size of alphabet, the number of rules per membrane, etc.

Also, it seems quite promising to investigate decidable computational properties of the language family $LS(t)P_*(ins_*^{*,*})$. We conjecture that it is incomparable with many known language families.

We recall the open problem posed in [4], namely, whether $hLSP_1(ins_*^{2,2})$ equals RE One may see that in order to solve the problem by the technique used in the article, it is enough to find a concise way to transfer a letter over a marked context. In our case this can be reduced to the question whether it is possible to compute the membrane regulation in the skin membrane.

Acknowledgments

The author acknowledges the support PIF program of University Rovira i Virgili, and projectno. MTM2007-63422 from the Ministry of Science and Education of Spain. The author sincerely thanks the help of Yurii Rogozhin and Seghey Verlan without whose supervision the work would not been done. The author also warmly thanks Gheorghe Păun and Erzsèbet Csuhaj-Varjú who draw attention, during BWMC-09, to the possibility of using different normal forms of grammars to simulate P insertion-deletion systems.

References

1. M. Daley, L. Kari, G. Gloor, R. Siromoney: Circular contextual insertions/deletions with applications to biomolecular computation. *Proc. SPIRE'99*, 1999, 47–54.
2. M. Mutyam, K. Krithivasan, A. Siddhartha Reddy: On characterizing recursively enumerable languages by insertion grammars. *Fundam. Inform.*, 64, 1-4 (2005), 317–324
3. B.S. Galiukschov: Semicontextual grammars. *Matematika Logica i Matematika Linguistika*, Tallin University, 1981 38–50 (in Russian).

4. L. Kari, Petr Sosík: On the weight of universal insertion grammars. *Theoretical Computer Sci.*, 396, 1-3 (2008), 264–270.
5. L. Kari: From micro-soft to bio-soft. Computing with DNA. *Proc. on Biocomputing and Emergent Computations*, 1997, 146–164.
6. L. Kari, Gh. Păun, G. Thierrin, S. Yu: At the crossroads of DNA computing and formal languages: characterizing RE using insertion-deletion systems. *Proc. of 3rd DIMACS*, 1997, 318–333.
7. L. Kari, G. Thierrin: Contextual insertion/deletion and computability. *Information and Computation*, 131, 1 (1996), 47–61.
8. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: Further results on insertion-deletion systems with one-sided contexts. *LNCS* 5196, 2008, 333–344.
9. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: One-sided insertion and deletion. Traditional and P systems case. *Proc. CBM08*, 2008, Austria, 53–64.
10. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: Computational power of P systems with small size insertion and deletion rules. *Proc. CSP08*, Ireland, 2008, 137–148.
11. S. Marcus: Contextual grammars. *Rev. Roum. Math. Pures Appl.*, 14 (1969), 1525–1534.
12. M. Margenstern, Gh. Păun, Yu. Rogozhin, S. Verlan: Context-free insertion-deletion systems. *Theoretical Computer Sci.*, 330 (2005), 339–348.
13. C. Martin-Vide, Gh. Păun, A. Salomaa: Characterizations of recursively enumerable languages by means of insertion grammars. *Theoretical Computer Sci.*, 205, 1–2 (1998), 195–205.
14. A. Matveevici, Yu. Rogozhin, S. Verlan: Insertion-deletion systems with one-sided contexts. *LNCS* 4664, 2007, 205–217.
15. Gh. Păun: *Marcus Contextual Grammars*. Kluwer, Dordrecht, 1997.
16. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
17. Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer, Berlin, 1998.
18. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer, Berlin, 1997.
19. A. Takahara, T. Yokomori: On the computational power of insertion-deletion systems. *LNCS*, 2568, 2003, 269–280.
20. S. Verlan: On minimal context-free insertion-deletion systems. *Journal of Automata, Languages and Combinatorics*, 12, 1-2 (2007, 317-328.

Simulation of Recognizer P Systems by Using Manycore GPUs

Miguel A. Martínez–del–Amor[1], Ignacio Pérez–Hurtado[1],
Mario J. Pérez–Jiménez[1], Jose M. Cecilia[2],
Ginés D. Guerrero[2], José M. García[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
{mdelamor,perezh,marper}@us.es
[2] Grupo de Arquitectura y Computación Paralela
Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia
Campus de Espinardo, 30100 Murcia, Spain
{chema,gines,jmgarcia}@ditec.um.es

Summary. Software development for cellular computing is growing up yielding new applications. In this paper, we describe a simulator for the class of recognizer P systems with active membranes, which exploits the massively parallel nature of the P systems computations by using a massively parallel computer architecture, such as Compute Unified Device Architecture (CUDA) from Nvidia, to obtain better performance in the simulations. We illustrate it by giving a solution to the N-Queens problem as an example.

1 Introduction

Membrane computing (or cellular computing) is an emerging branch within natural computing that was introduced by Gh. Păun [20]. The main idea is to consider biochemical processes taking place inside living cells from a computational point of view, in a way that gives us a new nondeterministic model of computation by using cellular machines.

Since the model was presented, many software applications have been produced [9]. The common purpose of all these software applications is to simulate P systems devices (cellular machines), and hence the designers have faced similar difficulties. However, these systems were usually focused on, and adapted for, particular cases, making it difficult to work on generalizations.

P systems simulators are tools that help the researchers to extract results from a model. These simulators have to be as much efficient as possible when handling

large problem sizes. The massively-parallel nature of the P systems computations points out to looking for a massively-parallel technology where the simulator can run efficiently.

The newest generation of graphics processor units (GPUs) are massively parallel processors which can support several thousand of concurrent threads. Many general purpose applications have been designed on these platforms due to its huge performance [12], [15], [23]. Current NVIDIA GPUs, for example, contain up to 240 scalar processing elements per chip [14], and they are programmed using C and CUDA [26], [17].

In this paper we present a massively parallel simulator for the class of recognizing P systems with active membranes using CUDA. The simulator executes the P system which is defined by using the P-Lingua [4] programming language. The simulator is divided in two main stages: the *selection stage* and the *execution stage*. At this development stage, the *selection stage* is executed, in a parallel fashion, on the GPU and the *execution stage* is executed on the CPU.

The rest of the paper is structured as follows. In Section 2 several definitions and concepts are given for a correct understanding of the paper. Section 3 introduces the Compute Unified Device Architecture (CUDA) and some concepts of programming on GPUs are specified. In Section 4 we explain the design of the simulator. In Section 5 we implement a solution to the N-Queens problem using the simulator and P-Lingua. Finally, in Section 6 we show some results and compare them with the sequential version of the simulator. The paper ends with some conclusions and ideas for future work in Section 7.

2 Preliminaries

Polynomial time solutions to **NP**-complete problems in membrane computing are achieved by trading time for space. This is inspired by the capability of cells to produce an exponential number of new membranes in polynomial time. There are many ways a living cell can produce new membranes: *mitosis* (cell division), *autopoiesis* (membrane creation), *gemmation*, etc. Following these inspirations a number of different models of P systems has arisen, and many of them proved to be computationally universal [4].

For the sake of simplicity, we shall focus in this paper on a model, *P systems with active membranes*. It is a construct of the form $\Pi = (O, H, \mu, \omega_1, \ldots, \omega_m, R)$, where $m \geq 1$ is the initial degree of the system; O is the alphabet of *objects*, H is a finite set of *labels* for membranes; μ is a membrane structure, consisting of m membranes injectively labeled with elements of H, $\omega_1, \ldots, \omega_m$ are strings over O, describing the *multisets of objects* placed in the m regions of μ; and R is a finite set of *rules*, where each rule is of one of the following forms:

(a) $[a \rightarrow v]^{\alpha}_{h}$, where $h \in H$, $\alpha \in \{+, -, 0\}$ (electrical charges), $a \in O$ and v is a string over O describing a multiset of objects (*object evolution rules*).

(b) $a\,[\,]_h^\alpha \rightarrow [b]_h^\beta$, where $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in O$ (*send-in communication rules*). An object is introduced in the membrane, possibly modified, and the initial charge α is changed to β.
(c) $[a]_h^\alpha \rightarrow [\,]_h^\beta b$, where $h \in H$, $\alpha, \beta \in \{+, -, 0\}$, $a, b \in O$ (*send-out communication rules*). An object is sent out of the membrane, possibly modified, and the initial charge α is changed to β.
(d) $[a]_h^\alpha \rightarrow b$, where $h \in H$, $\alpha \in \{+, -, 0\}$, $a, b \in O$ (*dissolution rules*). A membrane with a specific charge is dissolved in reaction with a (possibly modified) object.
(e) $[a]_h^\alpha \rightarrow [b]_h^\beta\,[c]_h^\gamma$, where $h \in H$,$\alpha, \beta, \gamma \in \{+, -, 0\}$, $a, b, c \in O$ (*division rules*). A membrane is divided into two membranes. The objects inside the membrane are replicated, except for a, that may be modified in each membrane.

Rules are applied according to the following principles:

- All the elements which are not involved in any of the operations to be applied remain unchanged.
- Rules associated with label h are used for all membranes with this label, no matter whether the membrane is an initial one or whether it was generated by division during the computation.
- Rules from (a) to (e) are used as usual in the framework of membrane computing, i.e., in a maximal parallel way. In one step, each object in a membrane can only be used by at most one rule (non-deterministically chosen), but any object which can evolve by a rule must do it (with the restrictions indicated below).
- Rules (b) to (e) cannot be applied simultaneously in a membrane in one computation step.
- An object a in a membrane labeled with h and with charge α can trigger a division, yielding two membranes with label h, one of them having charge β and the other one having charge γ. Note that all the contents present before the division, except for object a, can be the subject of rules in parallel with the division. In this case we consider that in a single step two processes take place: "first" the contents are affected by the rules applied to them, and "after that" the results are replicated into the two new membranes.
- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin is never dissolved.

Recognizing P systems were introduced in [21], and constitute the natural framework to study the solvability of decision problems, since deciding whether an instance has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizing P systems are associated in a natural way with P systems with *input*. The data representing an instance of the decision problem has to be provided to the P system to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation, *yes* or *no*, is sent to the environment [4].

In this paper, we present a simulation tool to simulate recognizer P systems with active membranes. The act of simulating something generally entails representing certain key characteristics or behavior of some physical, or abstract, system. On the other hand, an emulation tool duplicates the functions of one system by using a different system, so that the second system behaves like (and appears to be) the first system.

With the current technology, we can not emulate the functionality of a cellular machine by using a conventional computer to solve **NP**-complete problems in polynomial time, but we can simulate these cellular machines, not necessarily in polynomial time, in order to aid researchers. However, depending on the underlying technology where the simulator is executed, the simulations can take too much time.

The technology used for this work is called CUDA (Compute Unified Device Architecture). CUDA is a co-designed hardware and software solution to make easier developing general-purpose applications on the Graphics Processor Unit (GPU) [28]. The GPUs, that are one of the main components of traditional computers, originally were specialized for math-intensive, highly parallel computation which is the nature of graphics applications. These characteristics of the GPU were very attractive to accelerate scientific applications whose have massively parallel applications. However, the problem was the way to program applications on the GPU. This way involved to deal with GPUs designed for video games, so they have had to tune their applications using programming idioms tied to computer graphics, programming environment tightly constrained, etc [15], [12]. The CUDA extensions developed by Nvidia provides an easier environment to program general-purpose applications onto the GPU because it is based on ANSI C supported by several keywords and constructs. ANSI C is the standard published by the American National Standards Institute (ANSI) for the C programming language, which is one of the most used.

The P system devices are massively parallel which fits into massively parallel nature of the GPUs with thousands of threads running in parallel. These threads are units of execution which execute the same code concurrently on different piece of data. This idea of thread is very important and used in parallel computing.

3 Underlying Architecture

This work uses a graphics processor unit (GPU) from Nvidia as hardware target for its study: Tesla C1060. This section introduces the Tesla C1060 computing architecture, and it shows architecture parameters that can affect the performance. In addition, it analyzes the threading model of Tesla architectures depending on its computing capability, and also the most important issues in the CUDA programming environment.

3.1 Tesla 10 Base Microarchitecture

The Tesla C1060 [14] is based on scalable processor array which has 240 streaming-processor (SP) cores organized as 30 streaming multiprocessor (SMs). The applications start at the host side (the CPU) which communicates with the device side (the GPU) through a bus, which is a PCI Express bus standard (see Figure 1).

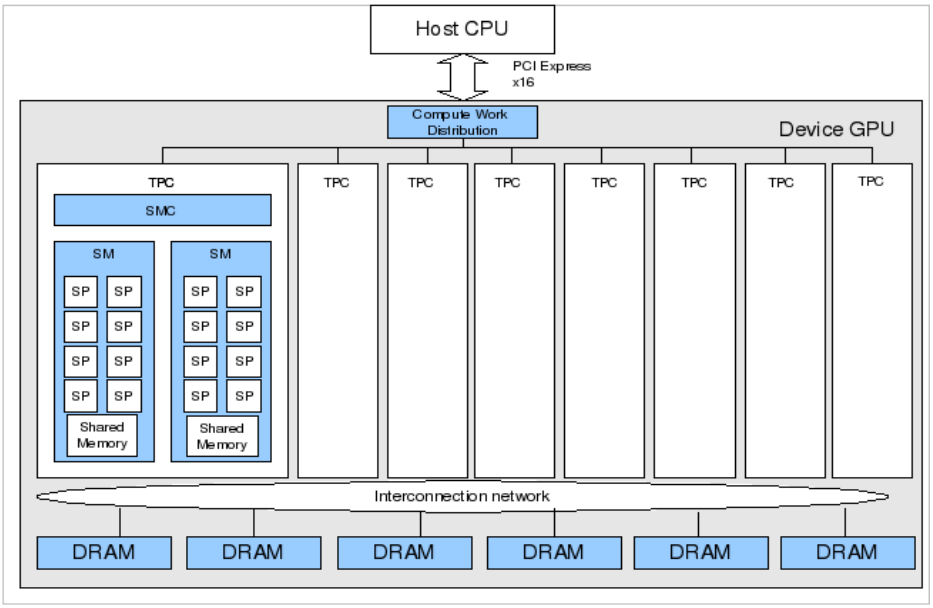


Fig. 1. Tesla Unified Architecture. TPC: Texture/processor cluster. SM: Streaming Multiprocessor, distributed among TPCs. SP: Streaming Processor.

The SM is the processing unit and it is unified graphics and computing multiprocessor. The parallel computing programs are programmed using ANSI C programming language along with CUDA extensions [28].

Every SM contains the following units: eight SPs arithmetic cores, one double precision unit, an instruction cache, a read only constant cache, 16-Kbyte read/write shared memory, a set of 16384 registers, and access to the off-chip memory (device/local memory).

The local and global (device) memory spaces are not cached, which means that every memory access to global memory (or local memory) generates an explicit memory access. A multiprocessor takes four clock cycles to issue one memory instruction for a "Warp" (see next subsection). Accessing local or global memory incurs an additional 400 to 600 clock cycles of memory latency [26], that is more expensive than accessing share memory and registers that incurs 4 cycles.

The Tesla C1060 achieves 102 GB/sec of bandwidth to the off-chip memory (running at 800 MHz). This bandwidth is not enough for the big set of cores and the possibilities to saturate it are high. To obtain the maximum bandwidth available it is needed to coalesce accesses to the device memory. The coalesced accesses are obtained whenever the accesses are contiguous 16-word lines, otherwise a fraction of this bandwidth it is obtained. Coalesced accesses will be a critical point in the optimization process.

In addition, the threads can use other memories like constant memory or texture memory. Reading from constant cache is as fast as reading from a registers, as long as all threads in the same warp read the same address. Texture Memory is optimized for 2D spatial locality (see Table 1).

Table 1. Memory System on the Tesla C1060

Memory	Location	Size	Latency	Access
Registers	On-Chip	16384 32-bits Registers per SM	$\simeq 0$ cycles	R/W
Shared Memory	On-Chip	16 KB per SM	$\simeq registers$	R/W
Constant	On-Chip	64 KB	$\simeq registers$	R
Texture	On-Chip	Up to Global	> 100 cycles	R
Local	Off-Chip	4 GB	400-600 cycles	R/W
Global	Off-Chip	4 GB	400-600 cycles	R/W

3.2 Threading Model

A SM is a hardware device specifically designed with multithreaded capabilities. Each SM manages and executes up to 1024 threads in hardware with zero scheduling overhead. Each thread has its own thread execution state and can execute an independent code path. The SMs execute threads in a Single-Instruction Multiple-Thread (SIMT) fashion [14]. Basically, in the SIMT model all the threads execute the same instruction on different piece of data. The SMs create, manage, schedule and execute threads in groups of 32 threads. This set of 32 threads is called *Warp*. Each SM can handle up to 32 Warps (1024 threads in total, see Table 2). Individual threads of the same Warp must be of the same type and start together at the same program address, but they are free to branch and execute independently.

The execution flow begins with a set of Warps ready to be selected. The instruction unit selects one of them, which is ready for issue and execute instructions. The SM maps all the threads in an active Warp to the SP cores, and each thread executes independently with its own instructions and register state. Some threads of the active Warp can be inactive due to branching or predication, and this is also another critical point in the optimisation process. The maximum performance is achieved when all the threads in an active Warp takes the same path (the same execution flow). If the threads of a Warp diverge, the Warp serially executes each branch path taken, disabling threads that are not on that path, and when all the paths complete, the threads reconverge to the original execution path.

Table 2. Major Hardware and Software Limitations programming on CUDA

Configuration Parameters	Limitation
Threads/SM	1024
Thread Blocks/SM	8
32-bit Registers/SM	16384
Shared Memory/SM	16KB
Threads/Block	512
Threads/Warp	32
Warps/SM	32

3.3 Parallel Computing with CUDA

The GPU is, nowadays, a single-chip massively parallel system which is inexpensive and readily available. However, programming a highly-parallel system has historically been a domain of few experts [25]. The emergence of Compute Unified Device Architecture (CUDA) has helped develop highly-parallel applications easier than before. CUDA programming toolkit is an extension of ANSI C including several keywords and constructs.

The GPU is seen as a coprocessor that executes data-parallel **kernel** functions. The user creates a program encompassing CPU code (Host code) and GPU code (Kernel code). These are separated and compiled by nvcc (Nvidia's compiler for CUDA code). The host code is responsible for transfer data to and from the GPU memory (device memory) via API calls, to initiates the kernel code executed on the GPU.

The threads executes the kernel code, and they are organized into a three-level hierarchy. At the highest level, each kernel creates a single grid that consists of many thread blocks. Besides, each thread block can contain up to 512 threads which can share data through Shared Memory and can perform barrier synchronization by invoking the `--syncthreads` primitive [25]. On the other hand, blocks can not perform synchronization. The synchronization across blocks can only be obtained by terminating the kernel. Finally, the threads within the block are organized into warps of 32 threads.

Each block within the grid have their own identifier[18]. This identifier can be one, two or three dimensions depending on how the programmer has declared the grid. In the same way, each thread within the block have their own identifier which can be one, two or three dimensions as well. Combining thread and block identifiers, the threads can access to different data address and also select the work that they have to do.

4 A Design of the Simulator for the Class of Recognizing P Systems

4.1 Algorithm Design

Whenever we design algorithms in the CUDA programming model, our main effort is dividing the required work into processing pieces, which have to be processed by *TB* thread blocks of *T* threads each. Using a thread block size of *T=256*, we have empirically determined to obtain the overall best performance on the Tesla C1060. Each thread block access to one different set of input data, and assigns a single or small constant number of input elements to each thread.

Each thread block can be considered independent to the other, and it is at this level at which internal communication (among threads) is cheap using explicit barriers to synchronize, and external communication (among blocks) becomes expensive, since global synchronization only can be achieved by the barrier implicit between successive kernel calls. The need of global synchronization in our designs requires successive kernel calls even to the same kernel.

4.2 P System Simulator with Active Membranes

The simulator simulates a recognizer P system with active membranes, i.e $\Pi = (O, H, \mu, \omega_1, \ldots, \omega_m, R)$ according to the notation described in section 2.

The simulator is executed into two main stages: *selection stage* and *execution stage*. The *selection stage* consists of the search for the rules to be executed in each membrane. Once the rules have been selected, the *execution stage* consists of the execution of these rules. The *selection stage* takes the major part of the simulation time in the sequential code, since this part of the algorithm implies to check all the rules of the system in every membrane. So we have parallelized the *selection stage* on the GPU, and the *execution stage* is still executed on the CPU at this point of the implementation.

The input data for the *selection stage* consists of the description of the membranes with their multisets (strings over O, labels associated with the membrane in H, etc...) and the set of rules R to be selected. The output data of this stage will be the set of selected rules per membrane. Only the *execution stage* changes the information of the configuration.

Besides, we have identified each membrane as a thread block where each thread represents an element of the alphabet O. Each thread block runs in parallel looking for the set of rules that has to execute, and each individual thread is responsible for identifying if there are some rules associated with the element that it represents, and if so, send it back to the *execution stage*. Finally, the CPU takes the control and executes the rules previously selected.

As result of the *execution stage*, the membranes can vary including news elements, dissolving membranes, dividing membranes, etc. Therefore, we have to

modify the input data for the *selection stage* with the newest structure of membranes, and then call the selection again. It is an iterative process until a system response is reached.

Our simulator presents two restrictions: it can handle only two levels of membrane hierarchy for simplicity (the skin and the rest of elementary membranes), what is enough for solving lots of **NP**-complete problems; moreover, the number of objects in the alphabet must be divisible by a number smaller than 512 (the maximum thread block size), in order to distribute the objects among the threads equally.

5 A Case Study: Implementing a Solution to the N-Queens Problem

In this section, we present a solution to the **N-Queens** problem, given by Miguel A. Gutiérrez–Naranjo et al [8], using our simulator. The **N-Queens** problem is expressed as a formula in conjunctive normal form, in such way that one truth assignment of the formula is considered as **N-Queens** solution. A family of recognizer P system for the SAT problem [22] can state whether exists a solution to the formula or not sending *yes* or *no* to the environment.

However, the *yes* ot *no* answer from the recognizer P system is not enough. Besides, the system needs to give us the way to encode the state of the N-Queens problem.

The P system designed for solving the **N-Queens** problem is a modification of the P system for the SAT problem. It is an uniform family of deterministic recognizer P system which solves SAT as a decision problem (i.e., the P system sends *yes* or *no* to the environment in the last computation step), but it also stores the truth assignments that makes true the formula encoded in the elementary membranes of the halting configuration.

5.1 Implementation

P-Lingua 1.0[4] is a programming language useful for defining P system models with active membranes. We use P-Lingua to encode a solution to the **N-Queens** problem, and also to generate a file that our simulator can use as input. Figure 2 shows the P-Lingua process to generate the input for our simulator.

P-Linga 2.0[5] translates a model written in P-Lingua language into a binary file. A binary file is a file whose information is encoded in Bytes and bits (not understandable by humans like plain text), which is suitable for trying to compress the data. This binary file contains all the information of the P system (Alphabet, Labels, Rules, ...) which is executed by our simulator.

In our tests, we use the recognizer P system for solving the 3-Queens problem. This problem creates 512 membranes and up to 1300 different objects. For the 4-Queens problem, the system would create 65536 membranes and up to 8000

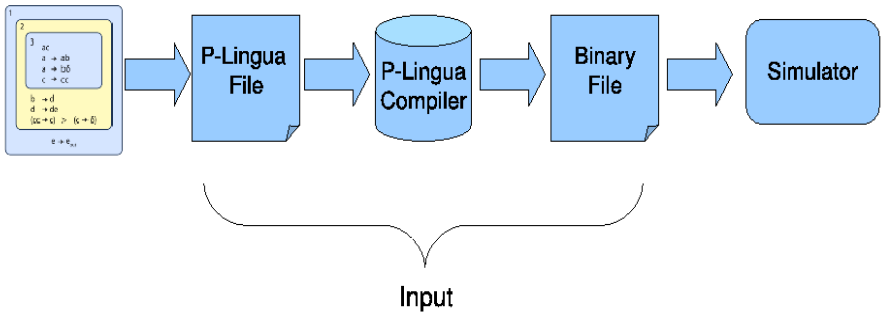


Fig. 2. Generation of the simulator's input

different objects. Currently, our simulator can not handle this example due to memory space limitation (requires up to 8GB in device memory). This problem can be solved with overlays of data and subsequent calls to the GPU. We are working on this solution, and also in other solutions to make possible execute problems with bigger memory size constraints on our simulator. On the other hand, note that 2-Queens is a system with only 4 membranes, what is not enough for exploiting the parallelism in P systems.

6 Performance Analysis

We now examine the experimental performance of our simulator. Our performance test are based on the solution to 3-Queen problem previously explained 5.1. Although this problem does not cover all the **NP**-complete problems that we want to simulate in our simulator, it states an example of how a **NP**-complete problem can be solved on the P system with active membranes simulator. We report the *selection stage* time which is executed on the GPU, and compare it with the *selection stage* for the sequential code. We do not include the cost of transferring input data from host CPU memory across the PCI-Express bus to the GPU's on board memory. Selection is one building block of larger-scale computation. Our aim is to get a full implementation of the simulator on the GPU. In such case, the transfers across PCI-Express bus will be close to zero.

The *selection stage* on the GPU takes about *195 msec.* This is 12 times faster than the *selection stage* on the CPU which takes *2345 msec.* We have used the NVIDIA GPU Tesla C1060 which has 240 execution cores and 4GB of device memory, plugged in a computer server with a Intel Core2 Quad CPU and 8GB of RAM, using the 32bits ubuntu server as Operating System.

Our experimental results demonstrate the results we expect to see: a massively-parallel problem such as selection of the rules in a P system with active membranes achieves faster running times on a massively-parallel architecture such as GPU.

7 Conclusions and Future Work

In this paper, we have presented a simulator for the class of recognizer P systems with active membranes using CUDA. The membrane computation has double parallel nature. The first level of parallelism is presented by the objects inside the membranes, and the second one is presented between membranes. Hence, we have simulated these P systems in a platform which provides those levels of parallelism. This platform is the GPU, with parallelism between thread blocks and threads. Besides, we have used a programming language called *P-Lingua* to generate a solution to the N-Queens problem, in order to test our simulator.

Using the power and parallelism that provides the GPU to simulate P systems with active membranes is a new concept in the development applications for membrane computing. Even the GPU is not a cellular machine, its features help the researches to accelerate their simulations allowing the consolidation of the cellular machines as alternative to the traditional machines.

The first version of the simulator is presented for P systems with active membranes, specifically, we have developed the *selection stage* of the simulator. In forthcoming versions, we will try to include the execution version in the GPU. This issue allows a completely parallel execution on the GPU, avoiding CPU-GPU transfers in every step, which degrades system performance.

On the other hand, we shall adapt our simulator to use the resources available on the GPU at maximum. To develop general purpose programs on the GPU is easier than several years ago with tools such as CUDA. However, extracting the maximum performance on the GPU is still hard, so we need to make a deep analysis to obtain the maximum performance available for our simulator.

It is also important to point out that this simulator is limited by the resources available on the GPU as well as the CPU (RAM, Device Memory, CPU, GPU). This limits the size of the instances of **NP**-complete problems whose solutions can be successfully simulated. In the following version of the simulator, we will try to reduce the memory requirements for the simulator in order to be able to simulate bigger instances of **NP**-complete problems. Moreover, it would be interesting to design heuristics to accelerate the computations of our simulator.

Although, the massively parallel environment that provides the GPUs is good enough for the simulator, we need to go beyond. The newest cluster of GPUs provides a higher massively parallel environment, so we will attempt to scale to those systems to obtain better performance in our simulated codes.

Acknowledgement

The first three 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 "Proyecto de Excelencia con Investigador de Reconocida Valía" of the Junta de Andalucía under grant P08–TIC04200. The last three authors acknowledge the support of the project from the Spanish MEC and Euro-

pean Commission FEDER funds under grants "Consolider Ingenio-2010 CSD2006-00046" and "TIN2006-15516-C04-03", as well as by the EU FP7 NoE HiPEAC IST-217068.

References

1. A. Alhazov, M.J. Pérez–Jiménez: Uniform solution of QSAT using polarizationless active membranes. In J. Durand-Lose, M. Margenstern, eds., *Machines, Computations, and Universality.* LNCS 4664, Springer, 2007, 122–133.
2. I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan: Brook for GPUs: stream computing on graphics hardware. SIGGRAPH '04, ACM Press (2004), 777–786.
3. G. Ciobanu, Gh. Păun, M.J. Pérez–Jiménez, eds.: *Applications of Membrane Computing.* Springer, 2006.
4. D. Díaz–Pernil, I. Pérez–Hurtado, M.J. Pérez–Jiménez, A. Riscos–Núñez: P–Lingua: A programming language for membrane computing. In D. Díaz–Pernil, M.A. Gutiérrez–Naranjo, C. Graciani–Díaz, Gh. Păun, I. Pérez–Hurtado, A. Riscos–Núñez, eds., *Proceedings of the 6th Brainstorming Week on Membrane Computing*, Sevilla, Fénix Editora, 2008, 135–155.
5. M. García–Quismondo, R. Gutiérrez–Escudero, I. Pérez–Hurtado, M.J. Pérez–Jiménez: P–Lingua 2.0: added features and first applications. In this volume.
6. M. Garland, S.L. Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, V. Volkov: Parallel computing experiences with CUDA. *IEEE Micro*, 28, 4 (2008), 13–27.
7. N.K. Govindaraju, D. Manocha: Cache–efficient numerical algorithms using graphics hardware. *Parallel Comput.*, 33, 10–11 (2007), 663–684.
8. M.A. Gutiérrez–Naranjo, M.A. Martínez–del–Amor, I. Pérez–Hurtado, M.J. Pérez–Jiménez: Solving the $N - Queens$ puzzle with P systems. In this volume.
9. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez: Available membrane computing software. In G. Ciobanu, Gh. Păun, M.J. Péréz–Jiménez (eds.) *Applications of Membrane Computing*, Springer–Verlag, 2006. Chapter 15 (2006), 411–436.
10. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez: Towards a programming language in cellular computing. *Electronic Notes in Theoretical Computer Science,* 123 (2005), 93–110.
11. M. Harris, S. Sengupta, J.D. Owens: Parallel prefix sum (Scan) with CUDA. *GPU Gems*, 3 (2007).
12. T.D. Hartley, U. Catalyurek, A. Ruiz, F. Igual, R. Mayo, M. Ujaldon: Biomedical image analysis on a cooperative cluster of GPUs and multicores. *ICS '08: Proceedings of the 22nd annual international conference on Supercomputing*, ACM, 2008, 15–25.
13. M.D. Lam, E.E. Rothberg, M.E. Wolf: The cache performance and optimizations of blocked algorithms. *ASPLOS-IV: Proceedings of the fourth international conference on Architectural support for programming languages and operating systems*, ACM, 1991, 63–74.
14. E. Lindholm, J. Nickolls, S. Oberman, J. Montrym: NVIDIA Tesla. A unified graphics and computing architecture. *IEEE Micro*, 28, 2 (2008), pp. 39–55.

15. W.R. Mark, R.S. Glanville, K. Akeley, M.J. Kilgard: Cg – a system for programming graphics hardware in a C–like language. *SIGGRAPH '03*, ACM, 2003, 896–907.
16. J. Michalakes, M. Vachharajani: GPU acceleration of numerical weather prediction. *IPDPS*, 2008, 1–7.
17. J. Nickolls, I. Buck, M. Garland, K. Skadron: Scalable parallel programming with CUDA. *Queue*, 6, 2 (2008), pp. 40–53.
18. J D. Owens, M. Houston, D. Luebke, S. Green, J.E. Stone, J.C. Phillips: Gpu computing. *Proceedings of the IEEE,* 96, 5 (2008), 879–899.
19. J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A.E. Lefohn, T.J. Purcell: A survey of general–purpose computation on graphics hardware. *Computer Graphics Forum*, 26, 1 (2007), pp. 80–113.
20. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143, and Turku Center for Computer Science-TUCS Report No 208.
21. M. J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini: Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2, 3 (2003), 265–285.
22. M. J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini: A polynomial complexity class in P systems using membrane division. In E. Csuhaj–Varjú, C. Kintala, D. Wotschke, G. Vaszil, eds., *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, Budapest, 2003, 284–294.
23. A. Ruiz, M. Ujaldon, J.A. Andrades, J. Becerra, K. Huang, T. Pan, J.H. Saltz: The GPU on biomedical image processing for color and phenotype analysis. *BIBE,* 2007, 1124–1128.
24. S. Ryoo, C. Rodrigues, S. Baghsorkhi, S. Stone, D. Kirk, W. mei Hwu: Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2008, 73–82.
25. S. Ryoo, C.I. Rodrigues, S.S. Stone, J.A. Stratton, Sain-Zee Ueng, S.S. Baghsorkhi, W.W. Hwu: Program optimization carving for GPU computing. *J. Parallel Distrib. Comput.*, 68, 10 (2008), 1389–1401.
26. NVIDIA CUDA Programming Guide 2.0, 2008: `http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf`
27. GPGPU organization. World Wide Web electronic publication: `www.gpgpu.org`
28. NVIDIA CUDA. World Wide Web electronic publication: `www.nvidia.com/cuda`

Sleep-Awake Switch with Spiking Neural P Systems: A Basic Proposal and New Issues

Jack Mario Mingo

Computer Science Department
University Carlos III of Madrid
Avda. de la Universidad Carlos III, 22, Colmenarejo (Madrid), Spain
jmingo@inf.uc3m.es

Summary. Spiking Neural P Systems are a kind of Membrane Systems developed with the aim of incorporating ideas from biological systems, known as spiking neurons, in the computational field. Initially, these systems were designed to take concepts of the neural science based on action potentials with the purpose of testing its possibilities from a computational point of view and not to be used as neurological models. In this work, a basic approach in the opposite sense is reviewed by means of the application of such systems on a well-known biological phenomenon. This phenomenon refers to the fluctuations among neural circuits which are responsible for swapping between awake-asleep states. This basic approach is analyzed and new issues are exposed.

1 Introduction

Sleep is a highly organized and actively induced cerebral state with different stages [11]. It has been observed two kinds of sleep: REM-sleep and non-REM sleep, each one with a number of specific features. Specifically, non-REM sleep comprises four stages. Each stage is defined according to its activity in the electroencephalogram (EGG). Stage 1 contains alternative periods of alpha activity (8-12Hz), irregular speed activity and theta activity (3.5-7.5Hz). Stage 2 does not show alpha activity in the EGG although in this stage appears a phenomenon called *sleep spindles* (bursts of 12-14Hz sinusoidal waves) and, sometimes, high-voltage biphasic waves called K complexes. Stage 3 consists of delta activity (less than 3.5Hz) during part of its time (20-50%) and stage 4 consists of delta activity during the most time (more than 50%). Approximately 90 minutes after sleep starts human beings fall in non-REM sleep which is characterized by rapid eye movements, unsynchronize EGG, a nearly complete inhibition of skeletal muscle tone (atonia). The brain temperature and metabolic rate are high, equal to or greater than during the waking state and some sexual activity can be present as well.

From a neural point of view, although sleep is controlled by the great part of the encephalon [4], there is a particularly important zone. It is the *ventrolat-*

eral preoptic nucleus (VLPO)which is located rostral to the hypothalamus. Some anatomical studies have shown that VLPO contains neurons that inhibit the system responsible of activating the brainstem and the forebrain. On the other hand, VLPO receives inhibitory afferents from the same regions that it inhibits. As Saper suggested [12] this reciprocal inhibition might be the basis to establish the transition between sleep and wake states. Reciprocal inhibition also is a feature in an electronic circuit called *flip-flop switch*. A flip-flop switch can be either *on* or *off*. According to Saper and collaborators' model, either VLPO is active and inhibits regions which induce wakefulness or regions that induce wakefulness are actives and inhibit VLPO. Like these regions are reciprocally inhibited it would not be possible that they were active at the same time.

There are some models that show how this switch might perform. On one hand, for example, Carlson's model [4] and Saper's model [12] are further schemes that show the information flow among neural groups along the circuits. On the other hand, several mathematical models have been proposed. A good review about these models is presented in [9]. With the aim to include some dynamical and structural aspects of the flip-flop switch a computational model that claim to describe the Saper's model is reviewed and analyzed in this paper. The model is based on *Spiking Neural P Systems* [6]. This type of computational model is part of the *Membrane Computing* [10] and its starting point consists on adding concepts typical of the neuronal computation based on spiking with the goal of testing its possibilities from a computational viewpoint. Nevertheless, this paper shows a Spiking Neural P system oriented in the opposite way. The system describes a specific neurophysiological mechanism called the sleep-wake switch.

The paper is organized as follows. Section 2 describes the neural control of slow waves sleep and the sleep-wake switch. Section 3 reviews a basic model of the sleep-wake switch with Spiking Neural P Systems. Section 4 shows results of the basic model. Finally, section 5 analyzes the proposed basic model and comments new issues to develop in the future.

2 Neural Control of Slow Waves Sleep

2.1 Non-REM stages

In non-REM sleep there is a low neuronal activity and both the metabolic function and the temperature of the brain are on its lowest level [11]. Besides, the sympathetic flow, heart rate and blood pressure decreases. Inversely, parasympathetic activity is increased while the muscle tone and reflexes remain intact.

Non-REM sleep is divided in four stages. Stage 1 represents the transition from wakefulness to sleep state. It lasts several minutes. When an individual is awoken shows an activity in the EGG with low voltage ($10 - 30\mu$V and 16-25Hz). As they relax, individuals show alpha activity around of $20 - 40\mu$V and 10Hz. This stage shows some activity on the muscles but there is no rapid eye movement, rather the sleeper shows slow eye movement and his EGG is characterized by a

low voltage and mixed frequencies. Stage 2 reveals bursts of sinusoidal waves called *sleep spindles* and biphasic waves called *K complexes*. These impulses are presented in an episodic way in front of a continuous activity in the EGG with low voltage. Stage 3 is characterized by a EGG which shows slow delta waves (0.5-2Hz) and high amplitude. Finally, in stage 4 this slow waves are increased and domain the EGG. In human beings stage 3 and stage 4 are known as *slow wave sleep*.

2.2 The sleep switch

The circuits in the brain that are responsible to regulate sleep and to produce wakefulness include cell groups in the brainstem, the hypothalamus and the basal forebrain [13] which are very important to activate the cerebral cortex and the thalamus. Neurons of these groups are inhibited during the sleep by a neural group that produces GABA (gammaminobutiric acid; an inhibitory neurotransmitter that seems to be widely distributed by all the encephalon and the spinal cord. It appears in many synaptic communications). This group corresponds with the *ventrolateral preoptic nucleus*. The reciprocal inhibition between both circuits acts like a switch and defines sleep and awake states. These states are discreet and they have sharp transitions among them.

To understand better how this switch performs it is very useful to look at the awake system and sleep system separately. We cite here the Saper's works [12] [13]. Regarding the first system, several studies carried out in the 70's and 80's showed an ascending activation pathway which induce the wakefulness state. The pathway has two main branches. The first one is an ascending branch directed toward the thalamus and it activates the thalamic relay neurons that are crucial for transmission of information to the cerebral cortex [13]. A mayor input of this thalamic relay neurons comes from a pair of neuron groups that produce acetylcholine (ACh): the *pedunculopontine tegmental nuclei* (PPT) and the *laterodorsal tegmental nuclei* (LDT). Neurons on these groups are more active during wakefulness and REM sleep and much less active during non-REM sleep when cortical activity is decreased. The second branch in the ascending activation system avoids the thalamus and activates neurons in the *lateral hypothalamus* (LHA) and the *basal forebrain* (BF). This second route starts from monoaminergic neurons in the upper brainstem and the caudal hypothalamus including the *locus coeruleus* (LC) which contains noradrenaline (NA), the *dorsal and median raphe nuclei* (DR) which contains serotonin (5-HT), the *ventral periaqueductal grey matter* which contains dopamine (DA) and the *tuberomammillary nucleus* containing histamine (HIS). The input to the cerebral cortex is augmented thanks to the lateral hypothalamic peptidergic neurons containing orexine or hypocretin (ORX), the melanin-concentrating hormone (MCH) and the basal forebrain neurons which contain GABA. Neurons in the monoaminergic nuclei have the property to fire faster during the wakefulness, slowing down during the non-REM sleep and stopping during REM sleep. Figures 1 and 2 show a schematic drawing with the mentioned systems.

During the 80's and 90's several researchers began to show interest for the inputs to the monoaminergic cells and found out that VLPO sent signals toward

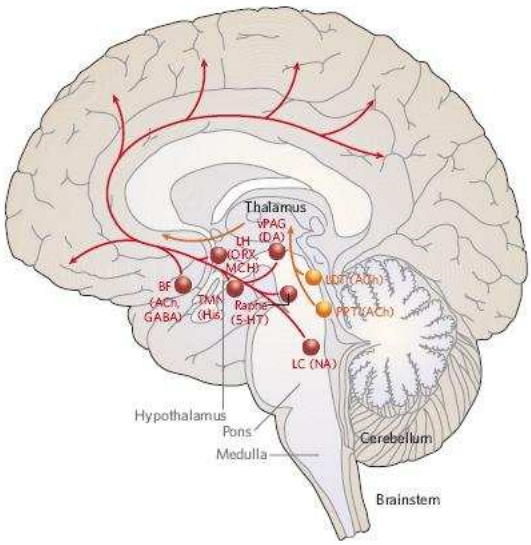


Fig. 1. Scheme with the main items in the Ascending Arousal System (AAS). Please see [13] for details

the main cells of the hypothalamus and the brainstem that are active during the wakefulness. Neurons in the VLPO are mainly active during sleep and they contain inhibitory neurotransmitters like GABA. These neurons form a dense group and a extended part more diffuse. Some of the studies showed that lesions in the dense group disrupted the non-REM sleep and lesions in the extended group did the same with the REM sleep. Experiments also showed that VLPO was innervated by the very monoaminergic systems that it innervates during sleep.

Reciprocal inhibition between sleep system and ascending arousal system acts like a circuit usually known as flip-flop in engineering [12]. A switch can be *on* or *off*. So, either VLPO is active and inhibits regions that induce wakefulness or these regions are actives and inhibit VLPO. This oscillator mechanism tries to avoid intermediate states because it is considered an adaptive advantage whether an animal is either asleep or awake. Saper and collaborators [12] suggested that an important function of hypocretinergic neurons situated in the lateral hypothalamus consist in helping to stabilize the oscillator. When these neurons are actives they induce wakefulness and inhibit sleep. The following schematic drawing in figure 3 shows the model proposed by Saper [13] in order to explain the sleep switch.

2.3 Homeostatic control of sleep

Nowadays is known that hypocretinergic neurons do not receive inhibitory afferents from each part of the oscillator, so that activation of these parts do not affect

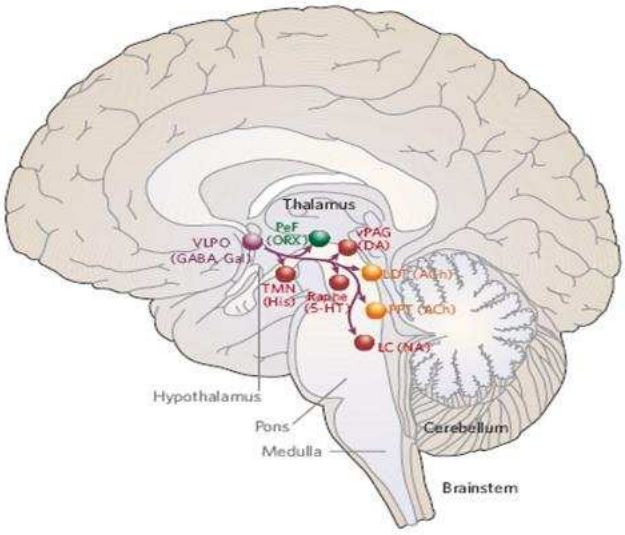


Fig. 2. Scheme with the projections from VLPO to (AAS). Please see [13] for details

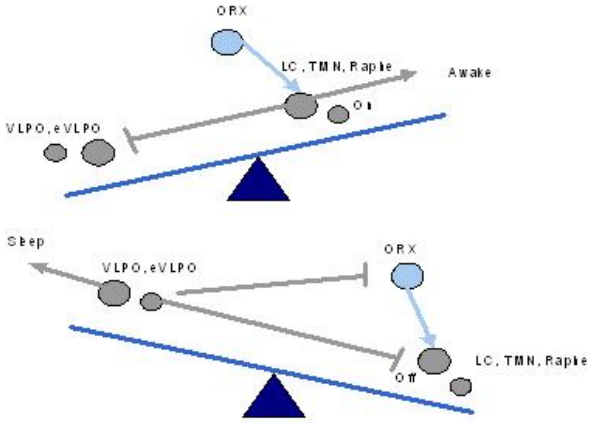


Fig. 3. Schematic drawing of the sleep switch proposed by Saper. Please see [13] for details

them [4]. If these neurons induce wakefulness they would hold the oscillator on. A question arise: how is sleep induced? Benington and cols. [3] suggested that a neurotransmitter called *adenosine* might play an essential role on the sleep control system. Its theory means that when neurons are getting especially actives

adenosine is accumulated. This substance acts like an inhibitory modulator and it produces an effect opposite to wakefulness. As Carlson suggests [4], if the VLPO is a critic region to generate sleep and the accumulated adenosine is a key factor to produce sleepiness it could be possible that this substance activates the VLPO. The proposed hypothesis suggests that adenosine favor sleep because it inhibits the neurons that usually inhibit the VPLO.

2.4 Circadian control of sleep

Several neurophysiological studies have confirmed a strong impact of 24-hour circadian cycle on the sleep control system. As Saper writes [13] "The *suprachiasmatic nucleus* (SCN) serves as the brain's master clock". Neurons in the SCN fire following a 24-hours cycle. The relation between SCN and the sleep control system has been studied [4] and the results show that SCN has projections to the VLPO or neurons containing orexin. However, the most outputs are directed toward the *adjacent subparaventricular zone* (SPZ) and the *dorsomedial nucleus of the hypothalamus* (DMH). The SPZ contains a ventral part (vSPZ) and a dorsal part (dSPZ) and it presents limited projections toward the VLPO, the neurons containing orexin and other elements in the sleep-wake system. Nevertheless, DMH is a main target because that region receives a lots of the afferents from the SPZ. Finally, the DMH is a main source of inputs to the VLPO and neurons containing orexin and it is very important in the sleep-wake regulatory system. The projections from DMH to VLPO comes from neurons containing GABA (therefore, they inhibit the sleep) while the projections toward the LHA are originated in neurons that contain glutamate (they act like exciters). Figure 4 shows the projections between all cited items as was proposed in [4].

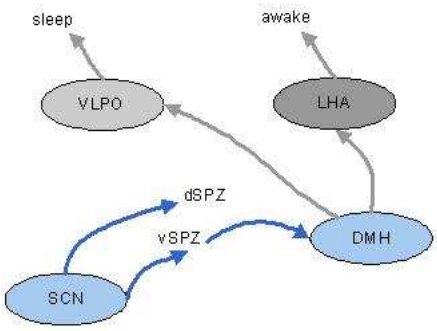


Fig. 4. Relation between the SCN and the sleep-wake regulatory system

3 Sleep-Wake Cycle with Spiking Neural P Systems: A Basic Model

Spiking Neural P Systems (SN P systems) were defined in [6] with the aim of introducing concepts typical of the spiking neurons [7], [5] into membrane computing. The standard model of SN P systems only considered excitatory rules but this configuration is not realistic to model the sleep-wake system because of the inhibitory nature of some synapses between neural groups. Starting from the standard model some variants has been proposed with the aim of modeling different situations. For example, in [1] an SN P system with extended rules was defined. A extended rule considers the possibility to send spikes along the axon with different magnitudes at different moments of time. Another idea about inhibitory connections among cells was slightly described in the same work. Bearing in mind these ideas in [8] an *SN P System with inhibitory rules* is described in the following way.

Definition 1. *A SN P System with inhibitory rules of degree m is a construct*

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, out_1, \ldots, out_n)$$

where

- $O = \{a\}$ *is the alphabet (the object a is called spike);*
- $\sigma_1, \ldots, \sigma_m$ *are neurons such as $\sigma_i = (n_i, R_i)$, with $1 \leq i \leq m$, means:*
  - $n_i \geq 0$ *is the initial number of spikes inside the neuron*
  - R_i *is a finite set of rules with the general form:*

    $$E/a^c \rightarrow a^p; d; t$$

    where E is a regular expression and it only uses the symbol a, $c \geq 1$ and p, $d \geq 0$, with $c \geq p$; besides, if $p = 0$ then $d = 0$. If the rule is excitatory then $t = 1$ and if the rule is inhibitory then $t = -1$
- $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, *with $(i, i) \notin syn$ for $1 \leq i \leq m$, are the synapses*
- $out_1, \ldots, out_n$ *represents output neurons with $1 \leq n \leq m$.*

As is usual in the SN P Systems literature these models can be represented by means of a graph with arcs between nodes. In an SN P System with inhibitory rules we can draw inhibitory rules with discontinuous lines and excitatory rules with continuous lines. Main differences between standard SN P systems and SN P systems with inhibitory rules as they have been defined are: a) the possibility of several output neurons, and b) the definition of inhibitory rules. Excitatory rules act like they do it in a standard SN P system and the inhibitory rules are interpreted in the following way: *if a neuron σ_i has an inhibitory connection with a neuron σ_j then spikes arriving from σ_i close the neuron σ_j during a step of time* (because of $t = -1$ in an inhibitory rule).

In the basic model proposed in [8] there are established the following simplifications:

- The *ascending arousal system* in the brainstem and the forebrain are necessaries to accumulate adenosine as consequence of a long activity of their neurons. This means that when the organism is awoken the adenosine is accumulated. The neural groups which are responsible to accumulate this necessity are called *Accumulator System.*
- The *dorsomedial nucleus of the hypothalamus* (DMH) is the most active part in the *suprachiasmatic nucleus* (SCN) and it is responsible to control sleep-wake transitions following a 24-hour cycle. This neural group is called *DMH System* and its goal is to provide a motivation to awake.
- The *Acumulator System* and the *DMH system* act as activators for the asleep and awake states but once they have activated their neural groups they stop to fire and neurons in the activation system (neurons of the groups LHA, TMN, LC and DR) and neurons in the regulator-VLPO combined system start to fire while the system remains sleeping or awakening.
- Bearing in mind the previous suppositions the DMH system and the LHA are the neural groups that control the awake state and the Acumulator system and the Regulator System are the neural groups that control the asleep state.

Figure 5 shows the possible connections that control sleep-wake switch from a neurological viewpoint. The connections were explained in Section 2.

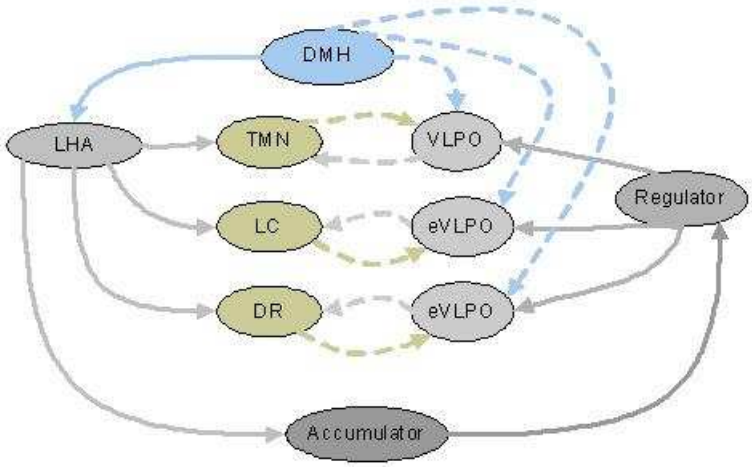


Fig. 5. Schematic drawing of the flip-flop circuit

In Figure 5, *LHA, DMH, TMN, LC, DR, VLPO (VLPO dense group), eVLPO (VLPO extended group)* are neural groups defined previously in section two. *Acumulator* represents the region where the necessity to sleep is accumulated and

Regulator would be a component of the Acumulator and its goal is to feed neurons in the VLPO. Both, *accumulator* and *regulator* groups are suppositions in the model because, from a neural point of view, regions that are responsible of accumulating adenosine are not known.

Starting from the previous figure and the SN P system with inhibitory rules a basic model for sleep-wake system is shown in Figure 6 where the inhibitory rules are represented as discontinuous lines and the excitatory rules as continuous lines.

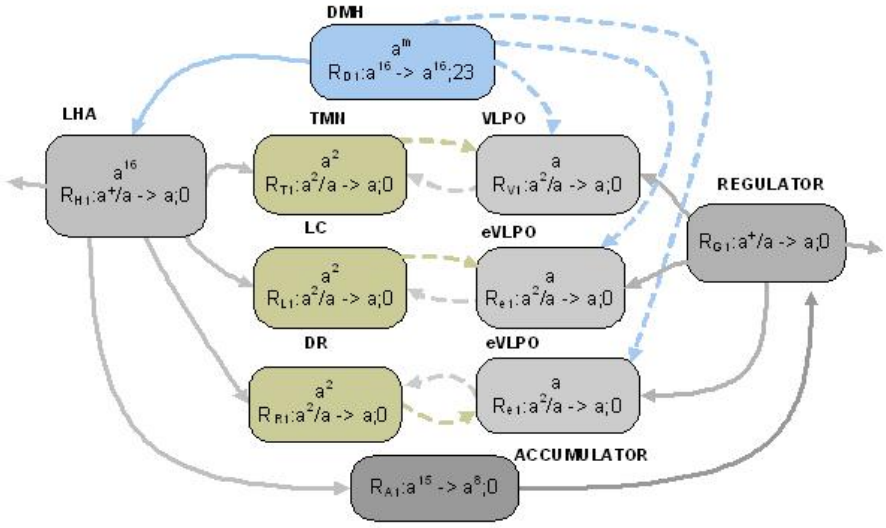


Fig. 6. Sleep-Wake basic switch with a Spiking P System

4 Results of the Basic Model

To analyze conveniently the system several suppositions must be taken into account:

- The system supposes that an individual is awoken during 16 hours and is sleeping during 8 hours. Another configurations, for example, 18 hours awake/6 hours sleep, are possible but always a fixed period is maintained.
- Each step of time in the system represents one hour of real time.

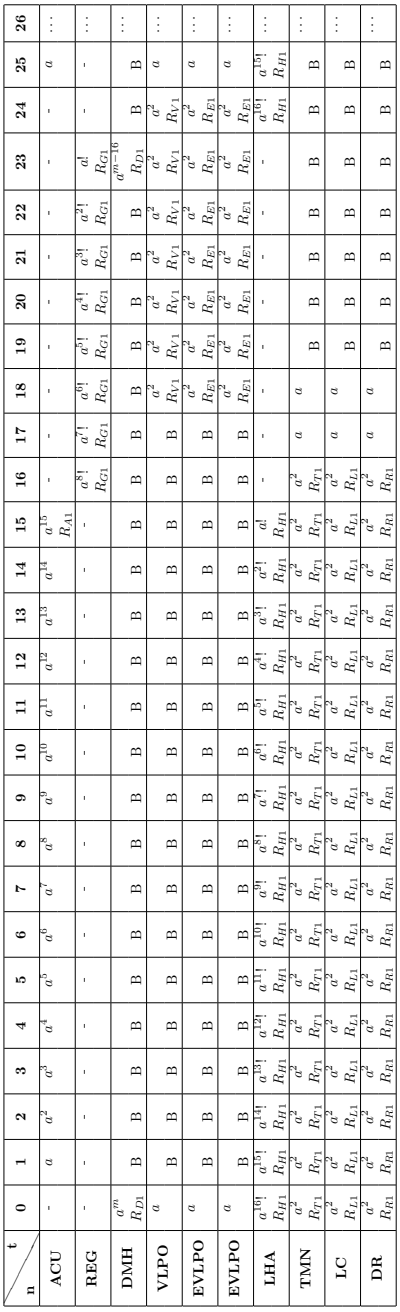
The basic model described previously can be applied starting from an initial configuration. Table 7 shows results when the system starts in awake state. First row on each square shows initial spikes and the second one represents the rule

applied. An exclamation symbol close to spikes means that the spike gets out from the network to the environment and it is useful to identify two states: *on* or *off*. When neurons in the LHA are firing the system represents wakefulness and if neurons in the *regulator system* are actives the system falls asleep. A brief explanation shows these situations. In wakefulness the neurons in the activation system (LHA, TMN, LC and DR) are firing and they select and execute a rule ($a^+/a \rightarrow a; 0$ is selected in LHA and $a^2/a \rightarrow a; 0$ in TMN, LC and DR). The activated rule in LHA, TMN and LC locks during a step of time neurons in the VLPO, both the dense group and the extended group, because of inhibitory rules among them. A lock is shown in the table with the **B** symbol. DMH system acts like the system's clock. In $t = 0$ DMH contains **m** spikes ($m >> 16$ is a necessary supposition if system does not finish) and it applies its only rule ($a^{16} \rightarrow a^{16}; 23$). This rule sends 16 spikes after 23 step of time because a daily cycle lasts 24 hours and the system tries to simulate that situation. This way, a complete cycle in the system lasts 24 steps of time (from 0 to 23). The only rule in DMH serves out to activate neurons in the LHA each 24 hours (steps of time). For its part, neurons in LHA are actives during 16 hours (steps of time). Along this period the necessity of sleep is accumulated in the *accumulator system*. When 15 steps of time have been consumed a state switch is started, the system gets to sleep and the *accumulator system* sends 8 spikes to the *regulator system* (it uses the rule $a^{15} \rightarrow a^8; 0$). To use this rule means that 7 spikes are losts and sleep state only lasts 8 hours (steps of time). Now, the *regulator system* controls sleep state executing its rule $a^+/a \rightarrow a; 0$. Once the system is sleeping, neurons of the dense and extended group are actives and execute the rule $a^2/a \rightarrow a; 0$. Besides, this rule locks neurons in TMN, LC and DR groups during a step of time. When $t = 23$, DMH is open again and it emits 16 spikes to LHA. Then, the system comes back to wakefulness and the process is repeated.

Notation: **n:** neural groups, **t:** steps of time, **ACU:** accumulator system, **REG:** regulator system, **DMH:** dorsomedial nucleus of the hypothalamus, **VPO:** dense group of the ventrolateral preoptic nucleus, **eVLPO:** extended group of the ventrolateral preoptic nucleus, **LHA:** neurons in the lateral hypothalamus, **TMN:** neurons in the tuberomammillary nucleus, **LC:** neurons in the locus coeruleus and **DR:** neurons in the dorsal and median raphe nuclei

5 Comments and New Issues

A computational basic model for sleep-wake switch is examined in this paper. The goal was to model a neurophysiological process by means of a computational device such as Spiking Neural P Systems. Traditionally, this type of computational mechanisms include biological concepts in order to test its possibilities from a computational point of view but this paper tries to apply them in a well-known neural process. In order to apply SN P Systems to this process a new definition of extended rules was necessary and this work proposes a definition for inhibitory rules with the aim of modeling the inhibitory connections among neural groups



n \ t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
ACU	-	a	a^2	a^3	a^4	a^5	a^6	a^7	a^8	a^9	a^{10}	a^{11}	a^{12}	a^{13}	a^{14}	a^{15} R_{A1}	-	-	-	-	-	-	-	-	-	a	$\cdots$
REG	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	$a^8 \mid$ R_{G1}	$a^7 \mid$ R_{G1}	$a^6 \mid$ R_{G1}	$a^5 \mid$ R_{G1}	$a^4 \mid$ R_{G1}	$a^3 \mid$ R_{G1}	$a^2 \mid$ R_{G1}	$a \mid$ R_{G1}	-	-	$\cdots$
DMH	a^m R_{D1}	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	a^{m-16} R_{D1}	B	B	$\cdots$
VLPO	a	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	a^2 R_{V1}	a^2 R_{V1}	a^2 R_{V1}	a^2 R_{V1}	a^2 R_{V1}	a^2 R_{V1}	a^2 R_{V1}	a	$\cdots$
EVLPO	a	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a	$\cdots$
EVLPO	a	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a^2 R_{E1}	a	$\cdots$
LHA	$a^{161} \mid$ R_{H1}	$a^{151} \mid$ R_{H1}	$a^{141} \mid$ R_{H1}	$a^{131} \mid$ R_{H1}	$a^{121} \mid$ R_{H1}	$a^{111} \mid$ R_{H1}	$a^{101} \mid$ R_{H1}	$a^{91} \mid$ R_{H1}	$a^{81} \mid$ R_{H1}	$a^{71} \mid$ R_{H1}	$a^{61} \mid$ R_{H1}	$a^{51} \mid$ R_{H1}	$a^{41} \mid$ R_{H1}	$a^{31} \mid$ R_{H1}	$a^{21} \mid$ R_{H1}	$a^1 \mid$ R_{H1}	-	-	-	-	-	-	-	$a^{61} \mid$ R_{H1}	$a^{151} \mid$ R_{H1}	$\cdots$
TMN	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a^2 R_{T1}	a	a	B	B	B	B	B	B	$\cdots$
LC	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a^2 R_{L1}	a	a	B	B	B	B	B	B	$\cdots$
DR	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a^2 R_{R1}	a	a	B	B	B	B	B	B	$\cdots$

Fig. 7. Results of SN P System simulating the sleep-awake switch

which are involved in the sleep-awake transitions. Starting from an SN P System with inhibitory rules a basic model is built with two important constraints:

1. the system supposes that an individual is slept or is awoken during fixed periods previously established.
2. the system splits the sleep control in two subsystems: *accumulator* and *regulator*. This supposition produces a delay in transition between states. Specifically, a step of time is lost from sleep to wakefulness and vice versa because accumulator and regulator subsystems can not start to perform at the same time. This is the reason why the rule $a^{15} \rightarrow a^8; 0$ in the accumulator uses 15 as number instead of 16.

In spite of this constraints the system can represent in a clear way a complex biological process and it defines formally such process. This formally definition would let possible implementations once an appropiate software was developed. Clarity of reading is also an advantage feature of SN P Systems in front of other alternatives like mathematical models based on equations, for example.

However, a number of interesting questions arises about the natural process and the basic model that it would be precise to analyze and solve in order to come near the model and the biological reality.

- The sleep and wakefulness states with fixed period are a very simplification because, obviously, neither human beings nor animals maintain along their life an established amount of time on each activity. Moreover, an individual does not sleep at all a day or more if really he or she cannot do it. This kind of flexibility is difficult to implement in the basic model even in SN P Systems as they are currently defined.
- Because the system considers one hour as step of time, the transition between sleep and wakefulness is completed slower than it occurs really in the biological process. Shorter steps can be defined in the model but then the model would be larger and tedious to show results in a table format. A possibility would be implement steps of time with different magnitudes but this question is not considered by currents SN P Systems.
- The inhibitory mechanism proposed by the model has several constraints and their effects are easily visible looking at the table. When a transition sleep-awake is produced either the *regulator system* or the *LHA neuron groups* fires (an exclamation symbol appears close to spikes). That way, the system represents that a state is active but, during one or two steps, the other main neuron groups in the dominant state does not fire because they still are locked by neurons in the opposite control system. For example, when $t = 16$ and $t = 17$, which is a sleep state, neurons in VLPO (dense and extended group) are locked (a B symbol appears in the square) because neurons in TMN, LC and DR are executing their rules yet. A similar situation occurs when $t = 24$ and $t = 25$ for the wakefulness state. This is a problem originated in the definition of the SN P System because a step of time is consumed since a spike gets out of a

neuron and comes in another one. Maybe, a best definition of inhibitory rules should be attended.

- Sleep or wakefulness states in the biological processes are a behavior but in the model they are simulated as firing in neuron groups. The model associates *LHA neurons* and *regulator system* as the output neurons in a way that is different of usual SN P systems. A best approach would be if a special cell or item in the SN P system performed this function.
- SN P systems generally send and receive signals from a state to another one but the system performs like a closed system in the sense that everything must be in the system. This is a drawback in this case because, for example, the *suprachiasmatic nucleus* does not receive only inputs of the considered neurons but other items. The basic model has replaced this fact with the delay property that SN P systems incorporate and the axiom ($m >> 16$) to simulate a non finite execution but better ideas on this matter would be a great contribution.
- SN P systems usually work with individual neurons and the basic model represents neuron groups as a single cell. A complex and interesting question to solve is concerned with how the model would be modified to work with several neurons on each group, for example, several LHA, TMN, LC, DR and VLPO cells.
- Besides the sleep-wake switch another biological process that has been studied from a neurological point of view are the transitions between non-REM and REM sleep. This transition involves practically the same neuron groups but it adds more complexity and connections among items. Contributions about this topic would also be interesting.

References

1. A. Alhazov, R. Freund, M. Oswald, M. Slavkovik: Extended Spiking Neural P Systems Generating Strings and Vectors of Non-Negative Integers. In H.J. Hoogeboom, Gh. Păun, G. Rozenberg, eds., *Workshop on Membrane Computing*, WMC7, Leiden, the Netherlands 2006, LNCS 4361, Springer, 2007, 123–134.
2. I.I. Ardelean, D. Besozzi: On Modelling Ion Fluxes Across Biological Membranes with P Systems. *Proceedings of the third brainstorming week on Membrane Computing*, 2005.
3. J.H. Benington, S.K. Koldali, H.C. Heller: Monoaminergic and cholinergic modulation of REM-sleep timing in rats. *Brain Research*, 681 (1995), 141–146.
4. N. Carlson: *Physiology of the Behavior*. 8th Edition. Addison Wesly, 2006.
5. W. Gerstner, W. Kistler: *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge Univ. Press, 2002.
6. M. Ionescu, Gh. Păun, T. Yokomori: Spiking Neural P Systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
7. W. Mass, C. Bishop, eds.: *Pulsed Neural Networks*. MIT Press, Cambridge 1999.
8. J.M. Mingo: Una Aproximacin al Interruptor del Sueo Mediante Spiking Neural P Systems (only in Spanish). Not published.

9. M. Nakao, A. Karashima, N. Katayama: Mathematical models of regulatory mechanisms of sleep-wake rhythms. *Cellular and Molecular Life Science*, 64 (2007), 1236–1243.
10. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
11. A. Rechtschaffen, J. Siegel: Sleep and Dreaming. Principles of Neuroscience. Fourth Edition, Edited by E.R. Kandel, J.H. Schwartz and T.M. Jessel, McGraw-Hill, New York, 2000, 936–947.
12. C.B. Saper, T.C. Chou, T.E. Scammell: The sleep switch: hypothalamic control of sleep and wakefulness. *Trends in Neurosciences*, 24, 12 (2001), 726–731.
13. C.B. Saper, T.E. Scammell, J. Lu: Hypothalamic regulation of sleep and circadian rhythms. *Nature*, 437 (2005), 1257–1263.

The Computational Complexity of Uniformity and Semi-uniformity in Membrane Systems

Niall Murphy[1] and Damien Woods[2]

[1] Department of Computer Science,
National University of Ireland, Maynooth, Ireland
http://www.cs.nuim.ie/∼nmurphy/
nmurphy@cs.nuim.ie
[2] Department of Computer Science and Artificial Intelligence,
University of Seville, Spain
http://www.cs.us.es/∼dwoods/
dwoods@us.es

Summary. We investigate computing models that are presented as families of finite computing devices with a uniformity condition on the entire family. Examples include circuits, membrane systems, DNA computers, cellular automata, tile assembly systems, and so on. However, in this list there are actually two distinct kinds of uniformity conditions.

The first is the most common and well-understood, where each input length is mapped to a single computing device that computes on the finite set of inputs of that length. The second, called semi-uniformity, is where each input is mapped to a computing device for that input. The former notion is well-known and used in circuit complexity, while the latter notion is frequently found in literature on nature-inspired computing models, from the past 20 years or so.

Are these two notions distinct or not? For many models it has been found that these notion are in fact the same, in the sense that the choice of uniformity or semi-uniformity leads to characterisations of the same complexity classes. Here, we buck this trend and show that these notions are actually distinct: we give classes of uniform membrane systems that are strictly weaker than their semi-uniform counterparts. This solves a known open problem in the theory of membrane systems.

1 Introduction

In his famous 1984 paper on DNA computing [1], Adleman mapped a specific instance of the travelling salesman problem (TSP) to a set of DNA strands, and then used well-known biomolecular techniques to solve the problem. To assert generality for his algorithm, one would define a (simple) mapping from arbitrary TSP instances to sets of DNA strings. Then, in order to claim that this mapping is not doing the essential computation, it would have to be easily computable

(e.g. logspace computable). Circuit uniformity provides a well-established framework where we map each input length $n \in \mathbb{N}$ to a circuit $c_n \in C$, with a suitably simple mapping. However, Adleman did something different, he mapped a specific *instance* of the problem to a computing device. We call this notion *semi-uniformity*, and in fact a large number of computation models use semi-uniformity. This raises the immediate question of whether the notions of uniformity and semi-uniformity are equivalent.

It has been shown in a number of models that whether one chooses to use uniformity or semi-uniformity does not affect the power of the model. However, in this paper we show that these notions are not equivalent. We prove that choosing one notion over another gives characterisations of completely different complexity classes, including known distinct classes.

We prove this result for a computational model called membrane systems (also known as P-systems) [15]. Membrane computing is a branch of natural computing which defines computation models that are inspired by the structure and function of living cells. The membrane computing model is sufficiently formal that this question can be clearly stated, e.g. it is stated as Open Problem C in [16].

Why is this result surprising? Every class of problems solved by a uniform family of devices is contained in the analogous semi-uniform class, since one is a restriction of the other. However, in all membrane system models studied to date, the classes of problems solved by semi-uniform and uniform families turned out to be equal [4, 10, 19]. Specifically, if we want to solve some problem, by specifying a family of membrane systems (or some other model), it is often much easier to first use the more general notion of semi-uniformity, and then subsequently try to find a uniform solution. In almost all cases where a semi-uniform family was given for some problem [3, 11, 13, 19], at a later point a uniform version of the same result was published [2, 4, 13]. Here we prove that this improvement is not always possible.

Since our main result proves something general about families of finite devices we would hope that, in the future, it can be applied to other computational models, besides membrane systems. Why? Firstly, our results are proved by converting the membrane system into a directed acyclic graph. Input acceptance is then rephrased as a graph reachability problem and this gives a very general tool that can be applied to other computational models (where we can find analogous graph representations). Secondly, the result concerns a general concept (uniformity/semi-uniformity) that is independent of particular formalisms. Besides membrane systems and circuits, some other models that use notions of uniformity and semi-uniformity include families of neural networks, molecular and DNA computers, tile assembly systems and cellular automata [6, 8, 12, 17, 18]. Our results could conceivably be applied to these models.

We now briefly observe what happens when we relate the notion of semi-uniformity to circuit complexity. We can easily define semi-uniformity for circuits. If the complexity class of the semi-uniformity function contains the prediction problem for circuits in the resulting family, then the semi-uniformity condition

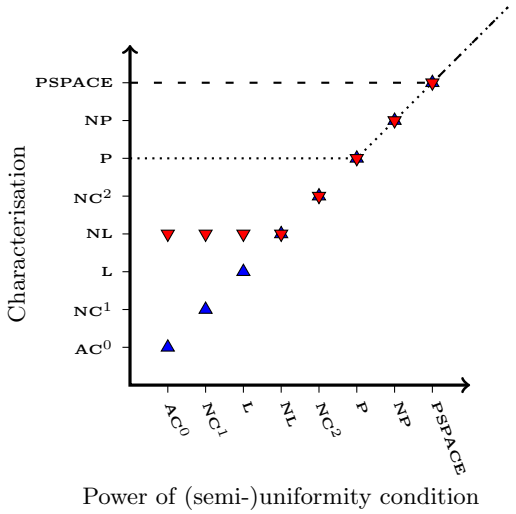


Fig. 1. Complexity classes that are characterised by the membrane systems studied in this paper. Characterisations by uniform systems are denoted by ▲, and semi-uniform by ▼. For example, Theorem 1 is illustrated by the fact that $\mathbf{AC}^0$-uniform systems characterise $\mathbf{AC}^0$, and that $\mathbf{AC}^0$-semi-uniform systems characterise **NL**. The previously known **P** [10] (indicated by ·····) and **PSPACE** [4, 20] (indicated by - -) results, where semi-uniform and uniform classes have the same power are also shown.

characterises the power of the model. If the semi-uniformity function is computable in the class that is characterised by the prediction problem for circuits in the resulting family, then we get the known characterisations for the analogous uniformity condition. However, in the uniform case it is not obvious what happens when we increase the uniformity beyond the power of the circuit, for example **P**-uniform $\mathbf{AC}^0 = \mathbf{AC}^0$ is an open problem [5]. Furthermore we should note that the uniformity condition in membrane systems preprocesses the input (as well as creating the device) and so is a seemingly different notion than circuit uniformity. If we add an analogous preprocessing step to circuits we see similar results as proven here for membrane systems: as soon as the preprocessing goes beyond the power of the circuit, we can ignore the circuit and let the preprocessing solve the problem. With preprocessing below the power of the circuit, the answer depends on the particular circuit model. In fact, if we restrict ourselves to polynomially sized circuits with only OR gates, we would see analogous results to those presented here, (i.e. our work shows that this circuit model is computationally equivalent to the $\mathcal{AM}^0_{-d}$ membrane systems discussed in this work).

1.1 Statement of result

We show that a class of problems, that is characterised by $\mathbf{AC}^0$-uniform membrane systems of a certain type, is a strict subset of another class that is characterised by $\mathbf{AC}^0$-semi-uniformity systems of the same type. Besides their respective use of uniformity and semi-uniformity, both models are identical, so this shows that for the membrane systems we consider, semi-uniformity is a strictly stronger notion than uniformity. Specifically, we show that the uniform systems characterise $\mathbf{AC}^0$ and the semi-uniform systems characterise **NL**, two classes known to be distinct. In the notation of membrane systems this is written as follows (explanations of notation are found in Section 2).

Theorem 1. $\mathbf{AC}^0 = (\mathbf{AC}^0, \mathbf{AC}^0)\text{–}\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} \subsetneq (\mathbf{AC}^0)\text{–}\mathbf{P}\mathrm{MC}^*_{\mathcal{AM}^0_{-d}} = \mathbf{NL}$

The left hand equality is proved in this paper, while the right hand equality was given in [11]. In Figure 1, Theorem 1 is illustrated by the leftmost pair of triangles. Essentially, the figure shows that if we use $\mathbf{AC}^0$ uniformity, the systems characterise $\mathbf{AC}^0$, while with $\mathbf{AC}^0$ semi-uniformity they characterise **NL**.

In fact we can also state a more general result for a number of complexity classes below **NL**, for brevity we keep the list short.

Theorem 2. *Let* $C \in \{\mathbf{AC}^0, \mathbf{NC}^1, \mathbf{L}\}$ *and assuming* $\mathbf{NC}^1 \subsetneq \mathbf{L} \subsetneq \mathbf{NL}$ *then* $C = (C, C)\text{–}\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} \subsetneq (C)\text{–}\mathbf{P}\mathrm{MC}^*_{\mathcal{AM}^0_{-d}} = \mathbf{NL}$

This shows that, roughly speaking, uniform membrane systems are essentially powerless, they are as weak and as strong as their uniformity condition. In Figure 1, Theorem 2 is illustrated by the triangles to the left of (and including) the uniformity condition **L**.

The essential ideas behind the proof of these theorems are as follows. First, we convert the (complicated looking) membrane systems into a directed acyclic graph called a dependency graph. Acceptance of an input word in some membrane system is equivalent to reachability in the corresponding dependency graph. We observe that for the class of systems that we consider, it is possible to make a number of simplifications to the model (and the dependency graph) without changing the power. In the semi-uniform case, even with these simplifications, the membrane systems have **NL** power. We then go on to prove that in the uniform case, the systems are severely crippled. We show this by proving that even though an arbitrary membrane system's dependency graph may have an **NL**-complete reachability problem, in fact there is an equivalent membrane system where reachability on the dependency graph is in $\mathbf{AC}^0$. This, along with some other tools, is used to show that if the power of the uniformity notion is $\mathbf{AC}^0$ or more, then the power of the entire family of systems is determined by the power of the uniformity.

2 Preliminaries

In this section we define membrane systems and some complexity classes, these definitions are based on those from [9, 13, 14, 15, 20].

The set of all multisets over a set A is denoted by $\mathrm{MS}(A)$. Let $G = (V, E)$ be a directed graph with $x, y, z \in V$. Then let $\mathsf{path}(x, y)$ be true if $x = y$, or $\exists\ z$ s.t $\mathsf{path}(x, z)$ and $\mathsf{path}(z, y)$. Otherwise $\mathsf{path}(x, y)$ is false.

2.1 Active membrane systems

Active membrane systems are a class of membrane systems with membrane division rules. Here division rules act only on elementary membranes, which are membranes that do not contain other membranes (i.e. leaves in the membrane structure).[3] To prove the results in this paper, we convert membrane systems into directed graphs. Thus, in this section, we provide some necessary membrane system definitions, but omit specific example of membrane systems.

Definition 3. *An active membrane system without charges is a tuple*

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

where,

1. $m \geq 1$ *is the initial number of membranes;*
2. O *is the alphabet of objects,* Σ *is the input alphabet,* $\Sigma \subset O$*;*
3. H *is the finite set of labels for the membranes;*
4. μ *is a membrane structure in the form of a tree, consisting of* m *membranes (nodes), labelled with elements of* H*. The parent of all membranes (the root node) is called the "environment" and has label* $env \in H$*;*
5. $w_1, \ldots, w_m$ *are strings over* O*, describing the multisets of objects placed in the* m *regions of* μ*.*
6. R *is a finite set of developmental rules, of the following forms:*
   a) $[\ a \rightarrow u\ ]_h$*, for* $h \in H,\ a \in O,\ u \in O^*$ *(object evolution)*
   b) $a[\ ]_h \rightarrow [\ b\ ]_h$*, for* $h \in H,\ a, b \in O$ *(communication in)*
   c) $[\ a\ ]_h \rightarrow [\ ]_h\ b$*, for* $h \in H,\ a, b \in O$ *(communication out)*
   d) $[\ a\ ]_h \rightarrow b$*, for* $h \in H,\ a, b \in O$ *(membrane dissolution)*
   (e) $[\ a\ ]_h \rightarrow [\ b\ ]_h\ [\ c\ ]_h$*, for* $h \in H,\ a, b, c \in O$ *(elementary membrane division).*

These rules are applied according to the following principles:

- All the rules are applied in a maximally parallel manner. That is, in one step, one object of a membrane is used by at most one rule (chosen in a non-deterministic way), but any object which can evolve by one rule of any form, must evolve.

[3] The more complicated non-elementary membrane division rule is also considered in the literature (where membranes containing other membranes can divide and replicate all of their substructure). All results in this paper hold when we permit non-elementary division, however we omit this detail as it adds unnecessary complications to our definitions and proofs.

- If at the same time a membrane labelled with h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that first the evolution rules of type (a) are used, and then the division is produced. This process takes only one step.
- The rules associated with membranes labelled with h are used for membranes with that label. At one step, a membrane can be the subject of only one rule of types (b)–(e).

2.2 Recogniser membrane systems

We recall [9] that a computation of a membrane system is a sequence of configurations such that each configuration (except the initial one) is obtained from the previous one by a transition (one-step maximally parallel application of the rules). Membrane systems are non-deterministic, therefore on a given input there are multiple possible computations. A computation that reaches a configuration where no more rules are applicable to the existing objects and membranes is called a halting computation.

Definition 4 ([9]). *A* recognizer membrane system *is a membrane system that, on each computation, outputs either object* `yes` *or object* `no` *(but not both), this occurs only when no further rules are applicable.*

2.3 Complexity classes

Consider a decision problem X, i.e. a set of instances $X = \{x_1, x_2, \ldots\}$ over some finite alphabet such that to each x_i there is an unique answer "yes" or "no". We say that a *family* of membrane systems solves a decision problem if each instance of the problem is solved by some family member. We denote by $|x| = n$ the length of any instance $x \in X$. Throughout this paper, $\mathbf{AC}^0$ circuits are **DLOGTIME**-uniform, polynomial sized (in input length n), constant depth, circuits with AND, OR and NOT gates, and unbounded fanin [7].

Definition 5. *Let $\mathcal{D}$ be a class of membrane systems and let $t : \mathbb{N} \to \mathbb{N}$ be a total function. The class of* problems solved by (e, f)-uniform families of membrane systems *of type $\mathcal{D}$ in time t, denoted (e, f)–$\mathrm{MC}_{\mathcal{D}}(t)$, contains all problems X such that:*

- *There exists an* f-uniform *family of membrane systems, $\mathbf{\Pi}_X = \{\Pi_X(1), \Pi_X(2), \ldots\}$ of type $\mathcal{D}$: that is, there exists a function $f : \{1\}^* \to \mathbf{\Pi}_X$ such that $f(1^n) = \Pi_X(n)$.*
- *There exists an input encoding function $e : X \to \mathrm{MS}(\Sigma)$ such that $e(x)$ is the input multiset, where $|x| = n$, and the input multiset is placed in a specific (input) membrane of $\Pi_X(n)$.*
- *$\mathbf{\Pi}_X$ is* sound *and* complete *with respect to problem X: $\Pi_X(n)$ starting with the encoding $e(x)$ of input $x \in X, |x| = n$, accepts iff the answer to x is "yes".*

- $\mathbf{\Pi}_X$ *is t-efficient:* $\Pi_X(n)$ *always halts in at most* $t(n)$ *steps.*

Definition 5 describes (e, f)-uniform families (i.e. with input) and we generalise this to define (h)-*semi-uniform families of membrane systems* $\mathbf{\Pi}_X = \{\Pi_X(x_1), \Pi_X(x_2), \ldots\}$ where there exists a function $h : X \to \mathbf{\Pi}_X$ such that $h(x) = \Pi_X(x)$. Here a single function (rather than two) is used to construct the semi-uniform membrane family, and so the problem instance is encoded using objects, membranes, and rules. Also, for each instance of $x \in X$ we have a (potentially unique) membrane system, a clear departure from the spirit of circuit uniformity. The resulting class of problems is denoted by (h)–$\mathrm{MC}^*_{\mathcal{D}}(t)$.

We often refer to $\mathbf{AC}^0$ uniform or logspace uniform (or semi-uniform) families of membrane systems which indicates that the functions e and f (or h) are $\mathbf{AC}^0$ or logspace computable functions.

We define (e, f)–$\mathbf{P}\mathrm{MC}_{\mathcal{D}}$ (and (h)–$\mathbf{P}\mathrm{MC}^*_{\mathcal{D}}$) as the class of problems solvable by (e, f)-uniform (respectively (h)-semi-uniform) families of membrane systems in polynomial time. We let $\mathcal{AM}^0$ denote the class of membrane systems that obey Definitions 3 and 4. We let $\mathcal{AM}^0_{-d}$ denote the class of membrane systems that obey Definition 3 but where rule (d) is forbidden, and Definition 4.

We let $(\mathbf{AC}^0)$–$\mathbf{P}\mathrm{MC}^*_{\mathcal{AM}^0_{-d}}$ denote the class of problems solvable by $\mathbf{AC}^0$-semi-uniform families of membrane systems in polynomial time with no dissolution rules.

Remark 6. A membrane system is said to be *confluent* if it is both sound and complete. That is, a membrane system Π is *confluent* if all computations of Π with the same input x (properly encoded) give the same result; either always accepting or else always rejecting.

In a confluent membrane system, given a fixed initial configuration, the system non-deterministically chooses one from a number of valid configuration sequences, but all of the reachable configuration sequences must lead to the same result, either all accepting or all rejecting.

3 Dependency graphs

A *dependency graph* (first introduced in [9]) represents the rules of membrane systems as a directed acyclic graph (DAG). For many proofs, this representation is significantly simpler and as such is an indispensable tool for characterising the computational complexity of membrane systems (without type (d) dissolution rules).

The dependency graph for a membrane system Π (without type (d) dissolution rules) is a directed graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \mathtt{in}, \mathtt{yes}, \mathtt{no})$ where $\mathtt{in} \subseteq V_{\mathcal{G}}$ represents the input multiset, and $\mathtt{yes}, \mathtt{no} \in V_{\mathcal{G}}$, represent the accepting and rejecting objects respectively. Each vertex $a \in V_{\mathcal{G}}$ is a pair $a = (o, h) \in O \times H$, where O is the set of objects in Π, and H is the set of membrane labels in Π. An edge (a, b)

exists in $E_{\mathcal{G}}$ if there is a developmental rule in Π such that the left hand side of the rule has the same object-membrane pair as a and the right hand side has an object-membrane pair matching b. In this paper, no membrane dissolution (type (d)) rules are allowed, and so the parent/child relationships of membranes in the structure tree cannot change during the computation. Thus when creating the edges for communication rules (types (b) and (c)) we can find the parent and child membranes for these rules and these choices remain correct for the entire computation (for example, to represent the rule $a[\ ]_h \to [a]_h$, that communicates an object a into a membrane of label h, it is only necessary to calculate the parent of h *one time* in the construction of the dependency graph).

For a number of previous results, it was sufficient to construct the graph $\mathcal{G}$ from Π in polynomial time [9]. For the results in this paper, we make the observation that $\mathcal{G}$ can be constructed from Π in $\mathbf{AC}^0$ (see Appendix A).

4 Proof of main result

The equality on the right hand side of Theorem 1 states that certain $(\mathbf{AC}^0)$-semi-uniform systems characterise **NL**. This was shown in [11], we quote the result:

Theorem 7 ([11]). $(\mathbf{AC}^0)$–$\mathbf{P}\mathrm{MC}^*_{\mathcal{AM}^0_{-d}} = \mathbf{NL}$

In rest of this paper, we prove the left hand side equality of Theorem 1, that is, we show that the analogous $(\mathbf{AC}^0, \mathbf{AC}^0)$-uniform systems characterise $\mathbf{AC}^0$. We begin by giving two normal forms for the membrane systems that are considered in this paper.

4.1 Normal forms

Lemma 8. *Any confluent $\mathcal{AM}^0_{-d}$ membrane system Π, with m membranes, is simulated by a $\mathcal{AM}^0_{-d}$ membrane system Π', that (i) has exactly one membrane and (ii) uses only rules of type (a). (By* simulate *we mean that the latter system accepts x iff the former does.)*

Proof (sketch). Given membrane system Π we construct its dependency graph $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}}, \mathtt{in}, \mathtt{yes}, \mathtt{no})$. We observe that we can convert $\mathcal{G}$ into a new membrane system $\Pi' = \Pi_{\mathcal{G}}$ by simply converting the edges of the graph into object evolution rules. Specifically, the set of objects of $\Pi_{\mathcal{G}}$ is $O_{\mathcal{G}} = V_{\mathcal{G}}$, and there is a single (environment) membrane of label *env*. The rules of $\Pi_{\mathcal{G}}$ are $\{[\, v \to \mathrm{str}(v)]_{env} \mid\ v \in V_{\mathcal{G}}\}$ where $\mathrm{str}(v)$ is the string formed by concatenating the elements of the set $\{s \mid (v, s) \in E_{\mathcal{G}}\}$. The vertices $\mathtt{yes}, \mathtt{no}$ are mapped to the `yes` and `no` objects respectively, and the set of vertices `in` becomes the input multiset of objects (actually an input *set*). This construction of $\Pi_{\mathcal{G}}$ (from $\mathcal{G}$) is $\mathbf{AC}^0$-computable.

For correctness, notice that the dependency graphs of Π and $\Pi_{\mathcal{G}}$ are isomorphic, so one accepts iff only the other does. Furthermore, $\Pi' = \Pi_{\mathcal{G}}$ has exactly one membrane with label *env*, and uses only type (a) rules (object evolution). □

Lemma 9. *Any confluent $\mathcal{AM}^0_{-d}$ membrane system Π, which has, as usual, multisets of objects in each membrane is simulated by another $\mathcal{AM}^0_{-d}$ membrane system Π', which has sets of objects in each membrane. (By* simulate *we mean that the latter system accepts x iff the former does.)*

Proof (sketch). We observe that in a dependency graph, $\mathcal{G}$, the multiset of objects is encoded as a *set* of vertices, so no information is stored regarding object multiplicities. Thus if we convert $\mathcal{G}$ into a new membrane system, Π' (as in the proof of the previous lemma), there are no rules in Π' with a right hand side with more than one instance of each object. The resulting system Π' accepts iff Π accepts since the dependency graphs of both systems are isomorphic. □

4.2 Uniformity is not equal to semi-uniformity

The following theorem is key to the proof of our main results (Theorems 1 and 2). Roughly speaking, Theorem 10 states that in uniform membrane systems of the type we consider, the uniformity condition dominates the computational power of the system. By letting $E = F = \mathbf{AC}^0$, the statement of Theorem 10 gives us the left hand side equality in Theorem 1. By letting $E = F \in \{\mathbf{AC}^0, \mathbf{NC}^1, \mathbf{L}\}$ we get the left hand side of Theorem 2. The remaining classes quoted in the theorem serve to illustrate Figure 1.

Theorem 10. *Let $E, F \in \{\mathbf{AC}^0, \mathbf{NC}^1, \mathbf{L}, \mathbf{NL}, \mathbf{NC}^2, \mathbf{P}, \mathbf{NP}, \mathbf{PSPACE}\}$ and let $F \subseteq E$. Then (E, F)–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} = E$.*

Proof. Let $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$ be the dependency graph of confluent recogniser membrane system Π from the class $\mathcal{AM}^0_{-d}$. We define the following subsets of the vertices of $V_\mathcal{G}$. Let $V_{\mathcal{G}_{\mathtt{yes}}} = \{v \mid v \in V_\mathcal{G}, \mathsf{path}(v, \mathtt{yes})\}$, $V_{\mathcal{G}_{\mathtt{no}}} = \{v \mid v \in V_\mathcal{G}, \mathsf{path}(v, \mathtt{no})\}$, and $V_{\mathcal{G}_{\mathrm{other}}} = V_\mathcal{G} \backslash (V_{\mathcal{G}_{\mathtt{yes}}} \cup V_{\mathcal{G}_{\mathtt{no}}})$.

We claim that $V_{\mathcal{G}_{\mathtt{yes}}} \cap V_{\mathcal{G}_{\mathtt{no}}} = \emptyset$. Assume otherwise, and let vertex $v \in V_{\mathcal{G}_{\mathtt{yes}}} \cap V_{\mathcal{G}_{\mathtt{no}}}$. This implies that $\mathsf{path}(\mathtt{in}, \mathtt{yes})$ and $\mathsf{path}(\mathtt{in}, \mathtt{no})$ are both true, which contradicts Definition 4 which states that *only* a `yes` or *only* a `no` object may be output by the system Π.

Next we claim that for confluent recogniser membrane systems Π from the class $\mathcal{AM}^0_{-d}$, a size-two input alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ is both necessary and sufficient, in the sense that this restriction does not alter the computing power of the system Π. Again, consider $\mathcal{G} = (V_\mathcal{G}, E_\mathcal{G})$, the dependency graph of Π. The "input" vertices $\mathtt{in} \subseteq V_\mathcal{G}$, represent the input objects of Π. On the one hand, it is necessary that $|\ \mathtt{in}\ | \geq 2$. This follows from the fact that it is necessary that both `yes` and `no` are reachable, and the fact that $V_{\mathcal{G}_{\mathtt{yes}}} \cap V_{\mathcal{G}_{\mathtt{no}}} = \emptyset$. Thus we need one vertex in $V_{\mathcal{G}_{\mathtt{yes}}}$ and another vertex in $V_{\mathcal{G}_{\mathtt{no}}}$. On the other hand, it can be seen that a single vertex from each set $V_{\mathcal{G}_{\mathtt{yes}}}, V_{\mathcal{G}_{\mathtt{no}}}$ is sufficient as follows. Given a set S of input vertices in $V_{\mathcal{G}_{\mathtt{yes}}}$, there is another system with an extra vertex, where all edges from this extra vertex lead to all vertices in S. A vertex can be analogously added for $V_{\mathcal{G}_{\mathtt{no}}}$. So even though membrane system Π may have multiple input objects, there is a

Π' that is equivalent in all respects except that there are exactly two input objects some extra rules. In particular, Π' accepts input x iff Π does.

The previous argument permits us to consider only those systems that have two input objects $\{\mathtt{a}, \mathtt{b}\}$. Thus we restrict attention to the case that the input encoding function is of the form $e : X \to \{\mathtt{a}, \mathtt{b}\}$. We say that e is a characteristic function with range $\{\mathtt{a}, \mathtt{b}\}$.

Let (e, f)–$\mathfrak{P}$ be a (e, f)-uniform family of confluent recogniser membrane systems from the class $\mathcal{AM}^0_{-d}$ that solves problem X. We claim that there exists a family (e, f')–$\mathfrak{P}$ that also solves X but uses a uniformity function f' that produces membrane systems of a very restricted form. Consider the dependency graph of the membrane system $f(n) = \Pi_X(n)$. In terms of reachability from $\{\mathtt{a}, \mathtt{b}\}$ to $\{\mathtt{yes}, \mathtt{no}\}$ in this graph (which corresponds to accepting/rejecting in the membrane system), the essential property is whether $\mathtt{path}(\mathtt{a}, \mathtt{yes})$ is true or $\mathtt{path}(\mathtt{a}, \mathtt{no})$ is true. However, this essential property is captured by the following (extremely simple) pair of dependency graphs: $\mathcal{G}_1 = (V_\mathcal{G}, E_{\mathcal{G}_1})$ and $\mathcal{G}_2 = (V_\mathcal{G}, E_{\mathcal{G}_2})$ where $V_\mathcal{G} = \{\mathtt{a}, \mathtt{b}, \mathtt{yes}, \mathtt{no}\}$, $E_{\mathcal{G}_1} = \{(\mathtt{a}, \mathtt{yes}), (\mathtt{b}, \mathtt{no})\}$, and $E_{\mathcal{G}_2} = \{(\mathtt{a}, \mathtt{no}), (\mathtt{b}, \mathtt{yes})\}$. Therefore if there is a family (e, f)–$\mathfrak{P}$ that solves X, where f represents valid, but arbitrary, dependency graphs, then there is another family of the form (e, f')–$\mathfrak{P}$ that also solves X and is identical in every way except that f' represents dependency graphs of the restricted form just described.

Now we prove the upper bound (e, f)–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} \subseteq E$, where e, f are respectively computable in E, F, with $F \subseteq E$, and the classes E, F are as given in the statement. As we have just shown, for each problem X in the class (e, f)–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}}$ there is a family of the restricted form (e, f')–$\mathfrak{P}$ that solves X. To simulate (e, f')–$\mathfrak{P}$ with a given input $x \in X$ we compute the pair $\left(e(x), f'(1^{|x|})\right)$. The range of the function f' is two membrane systems, which correspond to the two (restricted) dependency graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ from above, and whose reachability problems are (trivially) in the weakest E that we consider ($E = \mathbf{AC}^0$). Furthermore, since we only consider the case where $F \subseteq E$ then we know that f' itself is computable in E and we have (e, f)–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}} \subseteq E$.

The lower bound $E \subseteq (e, f)$–$\mathbf{P}\mathrm{MC}_{\mathcal{AM}^0_{-d}}$ is easy to show. We use the fact, shown above, that e is a characteristic function with access to the input word. Thus the following simple family computes any problem from E: function $e(x)$ outputs $\mathtt{a}$ if x is a positive instance of X and $\mathtt{b}$ if x is a negative instance of X, and f simply maps $\mathtt{a}$ to $\mathtt{yes}$ and $\mathtt{b}$ to $\mathtt{no}$. □

Acknowledgements

Niall Murphy is funded by the Irish Research Council for Science, Engineering and Technology. Damien Woods is supported by a Project of Excellence from the Junta de Andaluca, grant number TIC-581.

References

1. Leonard Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.
2. Artiom Alhazov, Carlos Martín-Vide, and Linqiang Pan. Solving a PSPACE-complete problem by recognizing P Systems with restricted active membranes. *Fundamenta Informaticae*, 58(2):67–77, 2003.
3. Artiom Alhazov and Linqiang Pan. Polarizationless P Systems with active membranes. *Grammars*, 7:141–159, 2004.
4. Artiom Alhazov and Mario J. Prez-Jimnez. Uniform solution to QSAT using polarizationless active membranes. In Jrme Durand-Lose and Maurice Margenstern, editors, *Machines, Computations and Universality (MCU)*, volume 4664 of *LNCS*, pages 122–133, Orlans, France, September 2007. Springer.
5. Eric Allender. Applications of time-bounded Kolmogorov complexity in complexity theory. In Osamu Watanabe, editor, *Kolmogorov Complexity and Computational Complexity*, chapter 1, pages 4–22. Springer, 1992.
6. Martyn Amos. *Theoretical and Experimental DNA Computation*. Natural Computing Series. Springer, 2005.
7. David A. Mix Barrington, Neil Immerman, and Howard Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41(3):274–306, 1990.
8. Matthew Cook, David Soloveichik, Erik Winfree, and Jehoshua Bruck. Programmability of chemical reaction networks. Technical report, Caltech Parallel and Distributed Systems Group [http://caltechparadise.library.caltech.edu/perl/oai2] (United States), 2008. In submission.
9. Miguel A. Gutirrez-Naranjo, Mario J. Prez-Jimnez, Agustn Riscos-Nez, and Francisco J. Romero-Campero. Computational efficiency of dissolution rules in membrane systems. *International Journal of Computer Mathematics*, 83(7):593–611, 2006.
10. Niall Murphy and Damien Woods. Active membrane systems without charges and using only symmetric elementary division characterise P. *8th International Workshop on Membrane Computing, LNCS*, 4860:367–384, 2007.
11. Niall Murphy and Damien Woods. A characterisation of NL using membrane systems without charges and dissolution. *Unconventional Computing, 7th International Conference, UC 2008, LNCS*, 5204:164–176, 2008.
12. Ian Parberry. *Circuit complexity and neural networks*. MIT Press, 1994.
13. Mario J. Prez-Jimnez, Alvaro Romero-Jimnez, and Fernando Sancho-Caparrini. Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2(3):265–285, 2003.
14. Gheorghe Pun. P Systems with active membranes: Attacking NP-Complete problems. *Journal of Automata, Languages and Combinatorics*, 6(1):75–90, 2001.
15. Gheorghe Pun. *Membrane Computing*. Springer-Verlag, Berlin, 2002.
16. Gheorghe Pun. Further twenty six open problems in membrane computing. In *Proceedings of the Third Brainstorming Week on Membrane Computing, Sevilla (Spain)*, pages 249–262. Fnix Editoria, January 2005.
17. David Soloveichik and Erik Winfree. The computational power of Benenson automata. *Theoretical Computer Science*, 344:279–297, 2005.
18. David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM J. Comput.*, 36(6):1544–1569, 2007.
19. Petr Sosk. The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2(3):287–298, August 2003.

20. Petr Sosk and Alfonso Rodrguez-Patn. Membrane computing and complexity theory: A characterization of PSPACE. *Journal of Computer and System Sciences*, 73(1):137–152, 2007.

Appendix A

A.1 Constructing dependency graphs in $\mathbf{AC^0}$

We are given a binary string x that encodes a membrane system, Π. To make a dependency graph from a membrane system requires a constant number of parallel steps that are as follows. First, a row of circuits identifies all communication (type (b) and (c)) rules and uses the (static) membrane structure to determine the correct parent membranes, then writes out (a binary encoding of) edges representing these rules. Next, a row of circuits writes out all edges representing division (type (e)) rules, for example a rule $[\ a\ ]_h \rightarrow [\ b\ ][\ c\ ]$ becomes the edges $(a_h, b_h), (a_h, c_h)$ in the dependency graph. In the final step we deal with evolution (type (a) rules) where it is possible to have polynomially many copies of polynomially many distinct objects on the right hand side of a rule (e.g. $[\ a\ ]_h \rightarrow [\ bcbbcdee\cdots z\ ]_h$). To write out edges for these rules in constant time we take advantage of the fact that we require at most one edge for each object-membrane pair in $O \times H$. We have a circuit for each element of $\{o_h \mid o \in O, h \in H\}$. The circuit for o_h takes as input (an encoding of) all rules in R whose left hand side is of the form $[o]_h$. The circuit then, in a parallel manner, masks (an encoding of) the right hand side of the rule (for example $[bbcdc]_h$) with the encoding of each object in O, (in the example, masking for (encoded) b would produce (encoded) $bb000$). All encoded objects in the string are then ORed together so that if there was at least one copy of that object in the system we obtain a single instance of it. The circuit being unique for a specific left hand side $[o]_h$ now writes out an encoding of the edge (o_h, b_h) and an encoding of all other edges for objects that existed on the right hand side of this rule in parallel.

Structured Modeling with Hyperdag P Systems: Part A

Radu Nicolescu, Michael J. Dinneen, Yun-Bum Kim

Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
radu.nicolescu@cs.auckland.ac.nz

Summary. P systems provide a computational model based on the structure and interaction of living cells [9]. A P system consists of a hierarchical nesting of cell-like membranes, which can be visualized as a rooted tree.

Although the P systems are computationally complete, many real world models, e.g., from socio-economic systems, databases, operating systems, distributed systems, seem to require more expressive power than provided by tree structures. Many such systems have a primary tree-like structure completed with shared or secondary communication channels. Modeling these as tree-based systems, while theoretically possible, is not very appealing, because it typically needs artificial extensions that introduce additional complexities, nonexistent in the originals.

In this paper we propose and define a new model that combines structure and flexibility, called *hyperdag P systems*, in short, *hP systems*, which extend the definition of conventional P systems, by allowing dags, interpreted as hypergraphs, instead of trees, as models for the membrane structure.

We investigate the relation between our hP systems and neural P systems. Despite using an apparently less powerful structure, i.e., a dag instead of a general graph, we argue that hP systems have essentially the same computational power as tissue and neural P systems. We argue that hP systems offer a structured approach to membrane-based modeling that is often closer to the behavior and underlying structure of the modeled objects.

Additionally, we enable dynamical changes of the rewriting modes (e.g., to alternate between determinism and parallelism) and of the transfer modes (e.g., the switch between unicast or broadcast). In contrast, classical P systems, both tree and graph based P systems, seem to focus on a statical approach.

We support our view with a simple but realistic example, inspired from computer networking, modeled as a hP system with a shared communication line (broadcast channel). In Part B of this paper we will explore this model further and support it with a more extensive set of examples.

1 Introduction

P systems provide a distributed computational model, based on the structure and interaction of living cells, first introduced by G. Păun in 1998 [8]. The model was initially based on transition rules, but was later expanded into a large family of related models. Essentially, all versions of P systems have a structure consisting of cell-like membranes and a set of rules that govern their evolution over time.

Many of the "classical" versions use a structure where membranes correspond to nodes in a rooted tree. Such a structure is often visualized as Venn diagram where nesting denotes a parent/child relationship. For example, Figure 1 [10] shows the same P system structure with 9 membranes, labeled as $1, \ldots, 9$, both as a rooted tree and as a Venn diagram.

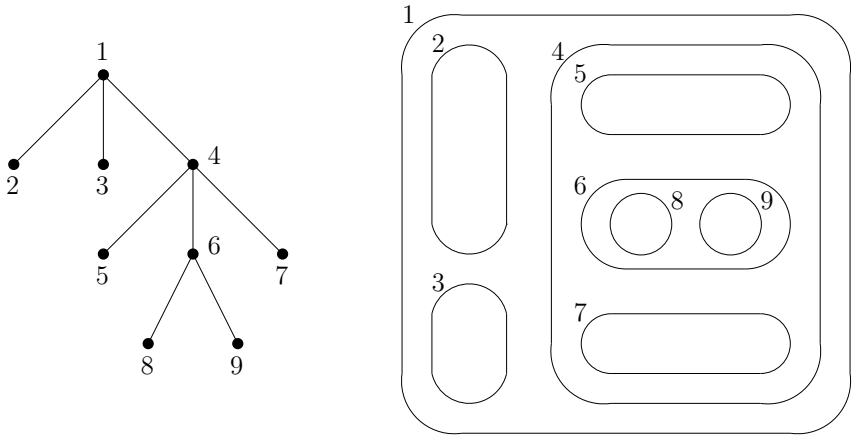


Fig. 1. A P system structure represented as a tree and as a Venn diagram.

More, recently, tissue P systems [7] and neural P systems [9], here abbreviated as tP and nP systems, respectively, have been introduced, partially to overcome the limitations of the tree model. Essentially, these systems organize their cells in an arbitrary digraph. For example, ignoring for the moment the actual contents of cells (states, objects, rules), Figure 2 illustrates the membrane structure of a simple tP or nP system, consisting of 3 cells, $\sigma_1, \sigma_2, \sigma_3$, where cell σ_1 is designated as the output cell.

A large variety of rules have been used to describe the operational behavior of P systems, the main ones being: multiset rewriting rules, communication rules and membrane handling rules. Essentially, transition P systems and nP systems use multiset rewriting rules, P systems with symport/antiport operate by communicating immutable objects, P systems with active membranes combine all three type rules. For a comprehensive overview and more details, we refer the reader to [9, 10].

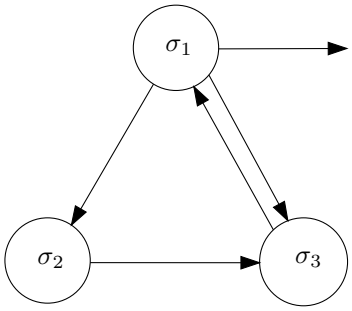


Fig. 2. A tP/nP system structure represented as a digraph.

Besides theoretical computer science and biology, P systems have been applied to a variety of other domains, ranging from linguistics [5] to theoretically efficient solutions of NP-complete problems [14], or to model distributed algorithms [3, 6]. The underlying tree structure provides good support for reasoning and formal verification, good potential for efficient implementation on multi-core architectures, and an excellent visualization, very appealing to practitioners.

Although the P systems are computationally complete, many real world models seem to require more expressive power, essentially trees augmented by shared or secondary communication channels. For example, the notion of a processing node having an unique parent is not true for (a) computer networks where a computer could simultaneously be attached to several subnets (e.g., to an Ethernet bus and to a wireless cell), (b) living organisms may be the result of multiple inheritance (e.g., the evolutionary "tree" is not really a tree, because of lateral gene transfer [4]) and (c) socio-economic scenarios where a player is often connected to and influenced by more than one factors [11, 12, 13].

Modeling these as tree-based systems, while theoretically possible, is not very appealing. Simulating shared or secondary channels requires artificial mechanisms that will ripple data up and down the tree, via a common ancestor. This could of course limit the merits of using a formal model. Tissue and neural P systems have been introduced to model such cases [7, 9]; details on neural P systems, in short, *nP systems*, are given in Section 3. However, these extensions are based on general graphs and, while allowing any direct communications, they also tend to obscure the structures already present in the modeled objects, limiting the advantages that a more structured approach could provide. Verification is more difficult without a clear modularization of concerns, practical parallel implementation could be less efficient, if the locality of reference is not enforced, and visualizations are not very meaningful, unless the primary structure is clearly emphasized.

We do not think that we have to choose between structure and flexibility. We propose a solution that seems to combine both, i.e., flexibility without sacrificing the advantages of a structured approach.

Our main contribution in this paper is to propose a new model for P systems, called *hyperdag P systems*, in short, *hP systems*, that allows more flexible communications than tree-based models, while preserving a strong hierarchical structure. This model, defined in Section 4, (a) extends the tree structure of classical P systems to directed acyclic graphs (dags), (b) augments the operational rules of nP systems with broadcast facilities (via a *go-sibling* transfer tag), and (c) enables dynamical changes of the rewriting modes (e.g., to alternate between determinism and parallelism) and of the transfer modes (e.g., to switch between unicast or broadcast). In contrast, classical P systems, both tree and graph based P systems, seem to focus on a statical approach.

We investigate the relation between our hP systems and neural P systems. Despite using an apparently less powerful structure, we show in Section 5 that our simple dag model has the same computational power as graph-based tissue and neural P systems.

We argue that hP systems offer a structured approach to membrane-based modeling that is often closer to the behavior and underlying structure of the modeled objects. Because our extensions address the membrane topology, not the rules model, they can be applied to a variety of P system flavors, including transition systems and symport/antiport systems.

We support our view with a realistic example (see Examples 8 and 9), inspired from computer networking, modeled as a hP system with a shared communication line (broadcast channel).

Classical P systems allow a "nice" planar visualization, where the parent/child relationships between membranes are represented by Venn-like diagrams. We show in Section 6 that the extended membrane structure of hP systems can still be visualized by hierarchically nested planar regions.

In this article we will restrict ourselves to P systems based on multiset rewriting rules, such as used by transition P systems and nP systems. However, because our extensions address the membrane topology, not the rules model, they can be applied to a variety of other P system flavors.

2 Preliminaries

A (binary) *relation* R over two sets X and Y is a subset of their Cartesian product, $R \subseteq X \times Y$. For $A \subseteq X$ and $B \subseteq Y$, we set $R(A) = \{y \in Y \mid \exists x \in A, (x, y) \in R\}$, $R^{-1}(B) = \{x \in X \mid \exists y \in B, (x, y) \in R\}$.

A *digraph* (directed graph) G is a pair (X, A), where X is a finite set of elements called *nodes* (or *vertices*), and A is a binary relation $A \subseteq X \times X$, of elements called *arcs*. For an arc $(x, y) \in A$, x is a *predecessor* of y and y is a *successor* of x. A length $n-1$ *path* is a sequence of n distinct nodes $x_1, \ldots, x_n$, such that $\{(x_1, x_2), \ldots, (x_{n-1}, x_n)\} \subseteq A$. A *cycle* is a path $x_1, \ldots, x_n$, where $n \geq 1$ and $(x_n, x_1) \in A$.

A *dag* (directed acyclic graph) is a digraph (X, A) without cycles. For $x \in X$, $A^{-1}(x) = A^{-1}(\{x\})$ are x's *parents*, $A(x) = A(\{x\})$ are x's *children*, and

$A(A^{-1}(x))\backslash\{x\} = A(A^{-1}(\{x\}))\backslash\{x\}$ are x's *siblings* (siblings defines a symmetric relation). A node $x \in X$ is a *source* iff $|A^{-1}(x)| = 0$, and $x \in X$ is a *sink* iff $|A(x)| = 0$. The *height* of a node x is the maximum length of all paths from x to a sink node. An arc (x, y) is *transitive* if there exists a path $x_1, \ldots, x_n$, with $x_1 = x$, $x_n = y$ and $n > 2$. dags without transitive arcs are here called *canonical*.

A (rooted unordered) *tree* is a dag with exactly one source, called *root*, and all other nodes have exactly one parent (predecessor). Sinks in a tree are also called *leaves*. A *topological order* of a dag is a linear reordering of vertices, in which each vertex x comes before all its children vertices $A(x)$.

Dags and trees are typically represented with parent-child arcs on the top-down axis, i.e., sources/roots up and sinks/leaves down. Figure 3 shows a simple dag.

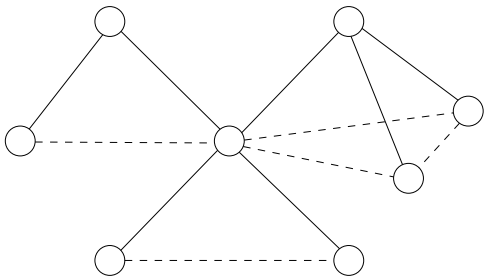


Fig. 3. A simple dag. The parent-child axis is up-down. Here, plain lines indicate parent-child relations and dashed lines indicate siblings.

We consider a variant hypergraph definition, based on multisets, as an extension of the classical definition [1], which is based on sets. A *hypergraph* is here a pair (X, E), where X is a finite set of elements called *nodes* (or *vertices*), and E is a finite *multiset* of subsets of X, i.e., $e \in E \Leftrightarrow e \subseteq X$. By using a multiset of edges, instead of a more conventional set of edges, we introduce an *intensional* element, where two *extensionally* equivalent hyperedges (i.e., hyperedges containing the same nodes) are not necessarily equal. A *graph* is a set based hypergraph where hyperedges are known as *edges* and contain exactly two nodes. Alternatively, a graph (X, E) can be interpreted as a digraph (X, A), where $A = \{(x, y) \mid \{x, y\} \in E\}$. Hypergraphs (set or multiset based) can be represented by planar diagrams, where hyperedges are represented as regions delimited by images of Jordan curves (simple closed curves) [2].

With the above hypergraph definition, a height 1 dag (X, A) can be interpreted as a hypergraph (X, E), where E is the multiset $E = \{A(x) \mid |A^{-1}(x)| = 0\}$. For example, Figure 4 represents, side by side, the dag $D = (\{a, b, c, d, e, f\}, \{(d, a), (d, b), (d, c), (e, b), (e, c), (f, b), (f, c)\})$ and its corresponding hypergraph $H = (\{a, b, c\}, \{d, e, f\})$, where $d = \{a, b, c\}, e = \{b, c\}, f = \{b, c\}$. Note that the apparently empty differences of regions are needed in the case of *multiset* based hypergraphs, to support the *intensional* (as opposed to the *extensional*) aspect:

here $e \neq f$, despite containing the same nodes, b and c, and neither e nor f is included in d.

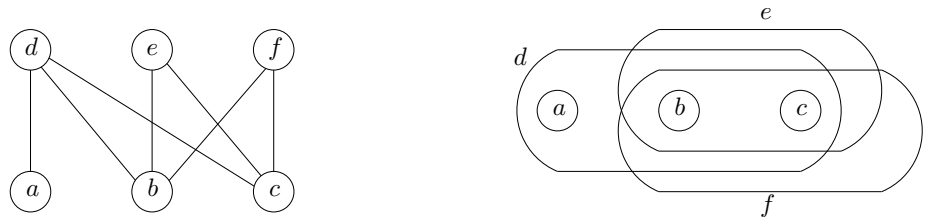


Fig. 4. A simple height 1 dag and its corresponding hypergraph representation.

Generalizing the above hypergraph definition, a height n *generalized hypergraph* is a system (X, E), recursively built via a sequence of n hypergraphs $(X_1, E_1), \ldots, (X_n, E_n)$ where $X_1 = X$, $X_i \cap E_i = \emptyset, X_{i+1} = X_i \cup E_i$, $e \cap E_i \neq \emptyset$ for $\forall e \in E_{i+1}$ and $E = \bigcup_{i \in \{1,\ldots,n\}} E_i$. An arbitrary height n dag can be represented by a height n generalized hypergraph, where the hypergraph nodes correspond to dag sinks, and height i hyperedges correspond to height i dag nodes, for $i \in \{1, \ldots, n\}$.

We will later see that any generalized hypergraph that corresponds to a non-transitive dag can also be represented by hierarchically nested planar regions delimited by Jordan curves, where arcs are represented by direct nesting. For example, Figure 5 shows a height 2 dag and its corresponding height 2 hypergraph (X, E), where $X = X_1 = \{a, b, c, d, e\}, E_1 = \{f, g, h\}, E_2 = \{i\}, E = \{f, g, h, i\}$.

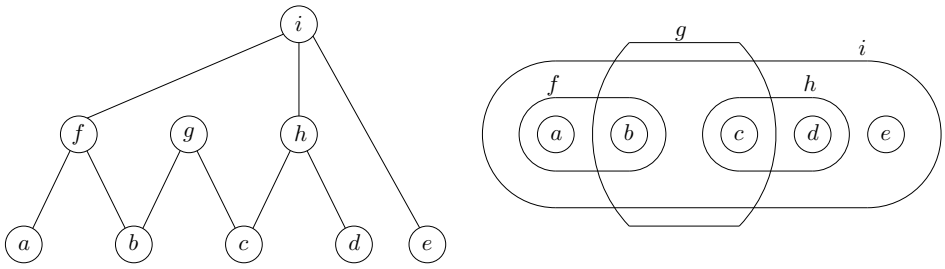


Fig. 5. A height 2 dag and its corresponding height 2 hypergraph.

An *alphabet* O is a finite non-empty sets of *objects*. We will assume that these alphabets are implicitly ordered. *Multisets* over an alphabet O are represented as strings over O, such as $o_1^{n_1} \ldots o_k^{n_k}$, where $o_i \in O$, $n_i \geq 0$, and, in the canonical form, letters appear in sorted order, i.e., $o_1 < \cdots < o_k$, and $n_i \geq 1$. The set of all multisets is denoted by O^*. For this representation, two strings are equivalent if they become equal after sorting, e.g., a^2cbd^0a and a^3bc are equivalent representations of the same multiset $\{a, a, a, b, c\}$. Under this convention, the empty string λ

represents the empty multiset, and string concatenation represents multiset union, e.g., $(a^2c) \cdot (ab) = a^3bc$.

3 Neural P Systems

In this paper we present the definition of neural P systems as given in [9], that coincide with an early definition of tissue P systems as given in [7]. We define the following sets of tagged objects: $O_{go} = \{(a, go) \mid a \in O\}$, $O_{out} = \{(a, out) \mid a \in O\}$, and we set $O_{tot} = O \cup O_{go} \cup O_{out}$. For simplicity, we will use subscripts for these tagged objects, such as a_{go} for (a, go) and a_{out} for (a, out). We also define projection homomorphisms, here denoted in postfix notation: $|_O, |_{go}, |_{out} : O_{tot}^* \to O^*$, by $o|_O = o, o_{go}|_{go} = o, o_{out}|_{out} = o$ for $o \in O$, and otherwise λ. For example, $a^2a_{go}^3b^4b_{go}|_{go} = a^3b$.

Definition 1 (Neural P systems [9, 7]). *A* neural P system *(of degree $m \geq 1$) is a system: $\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_{out})$, where:*

1. *O is an ordered finite non-empty alphabet of* objects*;*
2. *$\sigma_1, \ldots, \sigma_m$ are* cells*, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i)$, $1 \leq i \leq m$, where:*
   - *Q_i is a finite set (of* states*),*
   - *$s_{i,0} \in Q_i$ is the* initial state*,*
   - *$w_{i,0} \in O^*$ is the* initial multiset *of objects,*
   - *P_i is a finite set of multiset rewriting* rules *of the form $sx \to s'x'y_{go}z_{out}$, where $s, s' \in Q_i$, $x, x' \in O^*$, $y_{go} \in O_{go}^*$ and $z_{out} \in O_{out}^*$, with the restriction that $z_{out} = \lambda$ for all $i \in \{1, \ldots, m\} \backslash \{i_{out}\}$.*
3. *syn is a set of* digraph *arcs on $\{1, \ldots, m\}$, i.e., $syn \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$, representing* unidirectional *communication channels between cells, known as* synapses*;*
4. *$i_{out} \in \{1, \ldots, m\}$ indicates the* output cell*, the only cell allowed to send objects to the "environment".*

Example 1. To illustrate the operational behavior of nP systems, consider again the example of Figure 2, expanded now with states, rules and objects, see Figure 6. For simplicity, in this example only cell σ_1 provides rules. More formally, this nP system can be defined as the system $\Pi_1 = (O, \sigma_1, \sigma_2, \sigma_3, syn, i_{out})$, where:

- $O = \{a, b, c, d\}$;
- $\sigma_1 = (\{s, t\}, s, a^2, \ \{sa \to sdb_{go}c_{go}, sa \to sd, s \to t, td \to td_{out}\})$;
- $\sigma_2 = (\{s\}, s, \lambda, \emptyset)$;
- $\sigma_3 = (\{s\}, s, \lambda, \emptyset)$;
- $syn = \{(1, 2), (1, 3), (2, 3), (3, 1)\}$;
- $i_{out} = 1$.

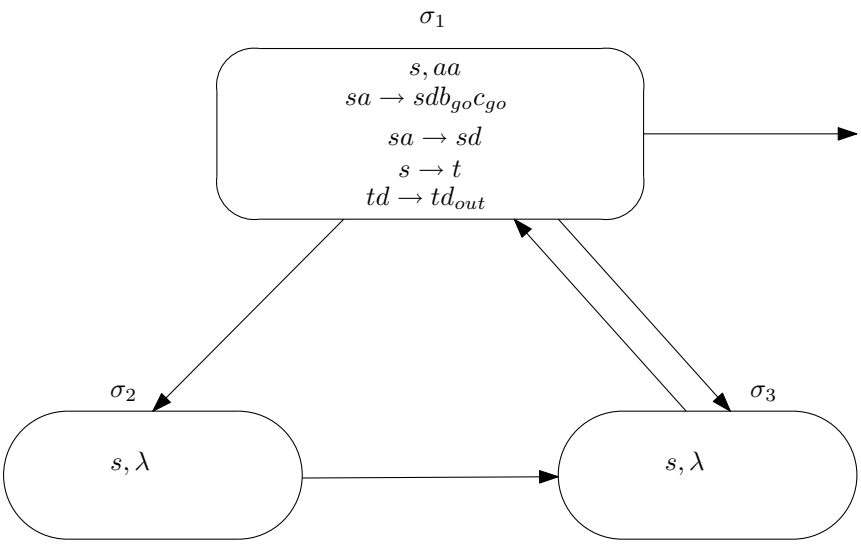


Fig. 6. Π_1, a simple nP system with states, objects and rules.

Neural P systems operate as indicated by multiset rewriting rules. A rewriting rule takes the existing state and objects and generates a new state and new objects, where some of the generated objects are tagged for *communication*, i.e., for transfer to neighboring cells along existing synapses. Objects that need to be transferred to a neighboring cell are tagged with *go* and objects that need to be output in the environment are tagged with *out* (in this definition, this is only possible from the i_{out} node).

Definition 2 (Configurations [9, 7]). *A* configuration *of the nP system Π is an m-tuple of the form $(s_1w_1, \ldots, s_mw_m)$, with $s_i \in Q_i$ and $w_i \in O^*$, for $1 \leq i \leq m$. The m-tuple $(s_{1,0}w_{1,0}, \ldots, s_{m,0}w_{m,0})$ is the* initial configuration *of Π.*

Example 2. For example, the initial configuration of the nP system Π_1 in Figure 6, is $C_0 = (saa, s\lambda, s\lambda)$.

Definition 3 (Rewrite and transfer modes [9, 7]). *Neural P systems have three* modes of rewriting objects*, inside a cell,* min (minimum), par (parallel), max (maximum)*, and three* modes of transferring objects*, from a given cell to another cell,* repl (replicate), one, spread*. For each nP system, the rewriting and transfer modes are* fixed *from start and apply to all rewriting and transition steps, as defined below.*

Definition 4 (Rewriting steps [9, 7]). *For each cell σ_i with $s, s' \in Q_i$, $x \in O^*$, $y \in O^*_{tot}$, we define a* rewriting step*, denoted by $\Rightarrow_\alpha$, where $\alpha \in \{min, par, max\}$:*

- *$sx \Rightarrow_{min} s'y$ iff $sw \to s'w' \in P_i$, $w \subseteq x$, and $y = (x - w) \cup w'$;*

- $sx \Rightarrow_{par} s'y$ *iff* $sw \rightarrow s'w' \in P_i$, $w^k \subseteq x$, $w^{k+1} \nsubseteq x$, *for some* $k \geq 1$, *and* $y = (x - w^k) \cup w'^k$;
- $sx \Rightarrow_{max} s'y$ *iff* $sw_1 \rightarrow s'w'_1, \ldots, sw_k \rightarrow s'w'_k \in P_i$, $k \geq 1$, *such that* $w_1 \ldots w_k \subseteq x, y = (x - w_1 \ldots w_k) \cup w'_1 \ldots w'_k$, *and there is no* $sw \rightarrow s'w' \in P_i$, *such that* $w_1 \ldots w_k w \subseteq x$ *(note that rules selected arbitrarily can be combined only if they start from the same state* s *and end in the same state* s'*).*

Example 3. As an example, considering cell 1 from the nP system Π_1, illustrated in Figure 6, the following rewriting steps are possible:

- $sa^n d^k \Rightarrow_{min} sa^{n-1} b_{go} c_{go} d^k$
- $sa^n d^k \Rightarrow_{min} sa^{n-1} d^{k+1}$
- $sa^n d^k \Rightarrow_{min} ta^n d^k$
- $ta^n d^k \Rightarrow_{min} ta^n d^{k-1} d_{out}$
- $sa^n d^k \Rightarrow_{par} sb^n_{go} c^n_{go} d^{k+n}$
- $sa^n d^k \Rightarrow_{par} sd^{k+n}$
- $sa^n d^k \Rightarrow_{par} ta^n d^k$
- $ta^n d^k \Rightarrow_{par} ta^n d^k_{out}$
- $sa^n d^k \Rightarrow_{max} sb^l_{go} c^l_{go} d^{k+n}$ with $0 \leq l \leq n$
- $sa^n d^k \Rightarrow_{max} ta^n d^k$
- $ta^n d^k \Rightarrow_{max} ta^n d^k_{out}$

We now define a *transition step between two configurations*, denoted by $\Rightarrow_{\alpha,\beta}$, where α is an *object processing mode* and β is an *object transfer mode*. Essentially, for a transition step we apply a rewriting step in each cell and we send to the neighbors all objects tagged for transfer.

Definition 5 (Transition steps, adapted from [9, 7]). *Given two configurations* $C_1 = (s_1 w_1, \ldots, s_m w_m)$ *and* $C_2 = (s'_1 w''_1, \ldots, s'_m w''_m)$, *we write* $C_1 \Rightarrow_{\alpha,\beta} C_2$, *for* $\alpha \in \{min, par, max\}$, $\beta \in \{repl, one, spread\}$, *if the conditions below are met.*

First, we apply rewriting steps (as defined in Definition 4) on each cell, i.e., $s_i w_i \Rightarrow_\alpha s'_i w'_i$, $1 \leq i \leq m$.

Secondly, we define $z_{j,k}$, *the outgoing object multisets from* j *to* k, *where* $j \in \{1, \ldots, m\}$ *and* $k \in syn(j)$*:*

- *If* $\beta = repl$, *then*
  - $z_{j,k} = w'_j|_{go}$, *for* $k \in syn(j)$;
- *If* $\beta = one$, *then*
  - $z_{j,k_j} = w'_j|_{go}$, *for an arbitrary* $k_j \in syn(j)$, *and* $z_{j,k} = \lambda$ *for* $k \in syn(j) \backslash \{k_j\}$;
- *If* $\beta = spread$, *then*
  - $\{z_{j,k}\}_{k \in syn(j)}$ *is an arbitrary multiset partition of* $w'_j|_{go}$.

Finally, we set $w''_i = w'_i|_O \cup \bigcup_{j \in syn^{-1}(i)} z_{j,i}$, *for* $i \in \{1, \ldots, m\}$.

Example 4. As an illustration, considering again the nP system Π_1, given in Figure 6, the following are examples of possible transfer steps:

- $(sa^n d^k, s, s) \Rightarrow_{min,repl} (sa^{n-1}d^{k+1}, sbc, sbc)$
- $(sa^n d^k, s, s) \Rightarrow_{min,repl} (sa^{n-1}d^{k+1}, s, s)$
- $(sa^n d^k, s, s) \Rightarrow_{min,one} (sa^{n-1}d^{k+1}, sbc, s)$
- $(sa^n d^k, s, s) \Rightarrow_{min,one} (sa^{n-1}d^{k+1}, s, sbc)$
- $(sa^n d^k, s, s) \Rightarrow_{min,spread} (sa^{n-1}d^{k+1}, sbc, s)$
- $(sa^n d^k, s, s) \Rightarrow_{min,spread} (sa^{n-1}d^{k+1}, sb, sc)$
- $(sa^n d^k, s, s) \Rightarrow_{min,spread} (sa^{n-1}d^{k+1}, sc, sb)$
- $(sa^n d^k, s, s) \Rightarrow_{min,spread} (sa^{n-1}d^{k+1}, s, sbc)$

Definition 6 (Halting and results [9, 7]). *If no more transitions are possible, the nP system halts. For halted nP system Π, the computational result is the multiset that was cumulatively sent* out *(to the "environment") from the output cell i_{out}. The numerical result is the vector $N_{\alpha,\beta}(\Pi)$ consisting of the object multiplicities in the multiset result, where $\alpha \in \{min, par, max\}$ and $\beta \in \{repl, one, spread\}$.*

Example 5. For example, if a nP system Π, over the alphabet $\{a, b, c, d\}$, sends out the multiset a^2cd^3 and then halts, then its numerical result is vector $N_{\alpha,\beta}(\Pi) = (2, 0, 1, 3)$.

Example 6. We replicate here another, perhaps more interesting example, originally given in [7]. Consider the following nP system, see Figure 7, $\Pi_2 = (O, \sigma_1, \sigma_2, \sigma_3, syn, i_{out})$, where:

- $O = \{a\}$;
- $\sigma_1 = (\{s\}, s, a, \{sa \to sa_{go}, sa \to sa_{out}\})$;
- $\sigma_2 = (\{s\}, s, \lambda, \{sa \to sa_{go}\})$;
- $\sigma_3 = (\{s\}, s, \lambda, \{sa \to sa_{go}\})$;
- $syn = \{(1, 2), (1, 3), (2, 1), (3, 1)\}$;
- $i_{out} = 1$.

The following results are straightforward:

$$
\begin{aligned}
N_{min,repl}(\Pi_2) &= \{(n) \mid n \geq 1\},\\
N_{min,\beta}(\Pi_2) &= \{(1)\}, \text{ for } \beta \in \{one, spread\},\\
N_{par,repl}(\Pi_2) &= \{(2^n) \mid n \geq 0\},\\
N_{par,\beta}(\Pi_2) &= \{(1)\}, \text{ for } \beta \in \{one, spread\},\\
N_{max,repl}(\Pi_2) &= \{(n) \mid n \geq 1\},\\
N_{max,\beta}(\Pi_2) &= \{(1)\}, \text{ for } \beta \in \{one, spread\}.
\end{aligned}
$$

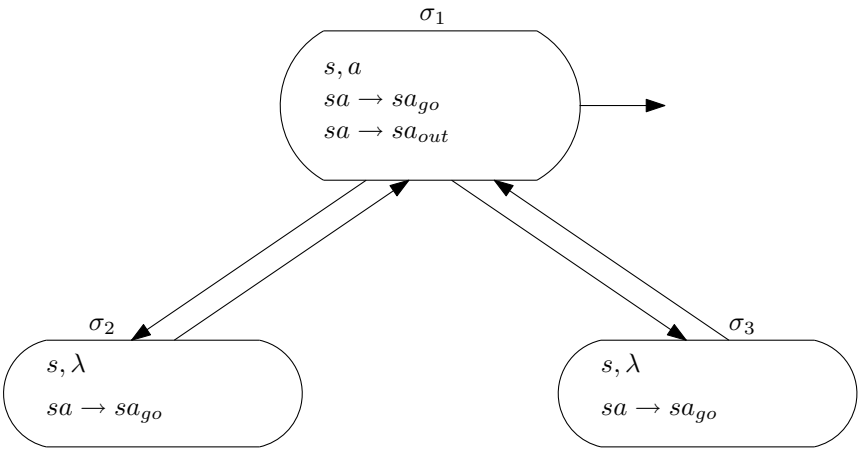


Fig. 7. Π_2, another simple nP system.

4 Hyperdag P Systems

We define hP systems as essentially nP systems (see Section 3), where the underlying digraph is a dag, with several adjustments. Besides the existing *go, out* tags, we consider three other object tags:

1. *go-parent*, abbreviated by the symbol $\uparrow$, indicating objects that will be sent to parents;
2. *go-child*, abbreviated by the symbol $\downarrow$, indicating objects that will be sent to children;
3. *go-sibling*, abbreviated by the symbol $\leftrightarrow$, indicating objects that will be sent to siblings;

The precise semantics of these tags will be explained below when we detail the hP object transfer modes. In fact, we could also discard the *go* tag, as it corresponds to the union of these news tags (*go-parent, go-child, go-sibling*); however, we will keep it here, for its concise expressive power. We use similar notation as nP systems for these new tags $O_\uparrow, O_\downarrow, O_\leftrightarrow$, and postfix projections $|_\uparrow, |_\downarrow, |_\leftrightarrow$.

Other extension tags, including addressing mechanisms (such as from/to/via tags) are possible, and indeed seem natural, but this is beyond the scope of this article.

Definition 7 (Hyperdag P systems). *A* hP system *(of degree m) is a system: $\Pi = (O, \sigma_1, \ldots, \sigma_m, \delta, I_{out})$, where:*

1. O is an ordered finite non-empty alphabet of objects*;*
2. $\sigma_1, \ldots, \sigma_m$ are cells*, of the form $\sigma_i = (Q_i, s_{i,0}, w_{i,0}, P_i)$, $1 \leq i \leq m$, where:*
- *Q_i is a finite set (of* states*),*
- *$s_{i,0} \in Q_i$ is the* initial state*,*

- $w_{i,0} \in O^*$ *is the* initial multiset *of objects,*
- P_i *is a finite set of multiset rewriting* rules *of the form* $sx \rightarrow s'x'u_{\uparrow}v_{\downarrow}w_{\leftrightarrow}y_{go}z_{out}$*, where* $s, s' \in Q_i$*,* $x, x' \in O^*$*,* $u_{\uparrow} \in O^*_{\uparrow}$ *,* $v_{\downarrow} \in O^*_{\downarrow}$ *,* $w_{\leftrightarrow} \in O^*_{\leftrightarrow}$ *,* $y_{go} \in O^*_{go}$ *and* $z_{out} \in O^*_{out}$*, with the restriction that* $z_{out} = \lambda$ *for all* $i \in \{1, \ldots, m\} \backslash I_{out}$*,*

3. δ *is a set of dag parent/child arcs on* $\{1, \ldots, m\}$*, i.e.,* $\delta \subseteq \{1, \ldots, m\} \times \{1, \ldots, m\}$*, representing* bidirectional *communication channels between cells;*
4. $I_{out} \subseteq \{1, \ldots, m\}$ *indicates the* output cells*, the only cells allowed to send objects to the "environment".*

The essential novelty of our proposal is to replace the arbitrary arc set *syn* by a more structured arcs set δ (dag), or, otherwise interpreted, as a generalized multiset-based hypergraph. This interpretation has actually suggested the name of our proposal, hyperdag P systems, and their abbreviation hP.

The changes in the rules format are mostly adaptations needed by the new topological structure. Here we have reused and enhanced the rewriting rules used by nP systems [9]. However, we could adopt and adapt any other rule set, from other variants or extensions of P systems, such as, rewriting, antiport/symport or boundary rules [10].

Definitions of *configurations*, *transitions*, *computations* and *results of computations* in hP systems are similar to definitions used for nP systems (Section 3), with the following essential additions/differences, here informally stated:

- The *rewrite mode* α and *transfer mode* β could but need not be fixed from the start—they may vary, for each cell σ_i and state $s \in Q_i$.
- If *object transfer mode* is *repl* (this is a deterministic step):
  - the objects tagged with $\uparrow$ will be sent to all the parents, replicated as necessary
  - the objects tagged with $\downarrow$ will be sent to all the children, replicated as necessary
  - the objects tagged with $\leftrightarrow$ will be sent to all the siblings, of all sibling groups, replicated as necessary
- If *object transfer mode* is *one* (this is a nondeterministic step):
  - the objects tagged with $\uparrow$ will be sent to one of the parents, arbitrarily chosen
  - the objects tagged with $\downarrow$ will be sent to one of the children, arbitrarily chosen
  - the objects tagged with $\leftrightarrow$ will be sent to one of the siblings, of one of the sibling groups, arbitrarily chosen
- If *object transfer mode* is *spread* (this is a nondeterministic step):
  - the objects tagged with $\uparrow$ will be split into submultisets and distributed among the parents, in an arbitrary way
  - the objects tagged with $\downarrow$ will be split into submultisets and distributed among the children, in an arbitrary way

- the objects tagged with $\leftrightarrow$ will be split into submultisets and distributed among the siblings and sibling groups, in an arbitrary way

Figure 8 schematically shows the possible transfers of objects from a membrane i, having two children, two parents, hence two sibling groups, with one sibling in the first group and two siblings in the other. The above mentioned transfer modes will select one, some or all the illustrated transfer targets, deterministically (*repl*) or nondeterministically (*one, spread*).

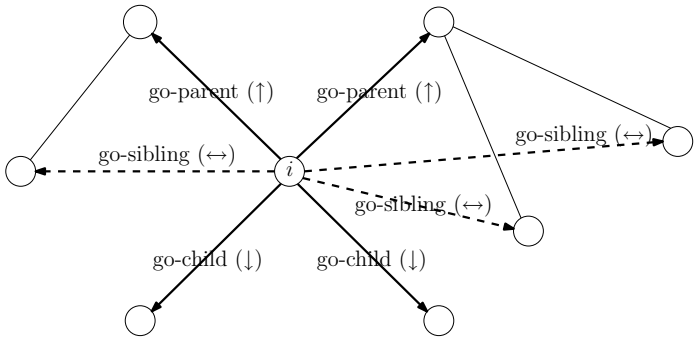


Fig. 8. Transfer modes in a hP system. The parent-child axis is top-down. Plain lines indicate parent-child relations and dashed lines indicate siblings. Arrows at the end of long thick lines, plain or dashed, indicate possible transfer directions from node i.

More formal definitions follow.

Definition 8 (Configurations). *A* configuration *of the hP system Π is an m-tuple of the form $(s_1w_1, \ldots, s_mw_m)$, with $s_i \in Q_i$ and $w_i \in O^*$, for $1 \leq i \leq m$. The m-tuple $(s_{1,0}w_{1,0}, \ldots, s_{m,0}w_{m,0})$ is the* initial configuration *of Π.*

Definition 9 (Rewrite and transfer modes). *For a hP system of degree m,*

- *the* object rewriting mode *is a function*

$$\alpha : \bigcup_{i \in \{1,\ldots,m\}} \{i\} \times Q_i \to \{min, par, max\} \ .$$

- *the* object transfer mode *is a function*

$$\beta : \bigcup_{i \in \{1,\ldots,m\}} \{i\} \times Q_i \to \{repl, one, spread\} \ .$$

Definition 10 (Rewriting steps). *For each cell σ_i with $s, s' \in Q_i$, $x \in O^*$, $y \in O^*_{tot}$, we define a* rewriting step*, denoted by $\Rightarrow_\alpha$, where $\alpha = \alpha(i, s) \in \{min, par, max\}$.*

- $sx \Rightarrow_{min} s'y$ *iff* $sw \rightarrow s'w' \in P_i$, $w \subseteq x$, *and* $y = (x - w) \cup w'$;
- $sx \Rightarrow_{par} s'y$ *iff* $sw \rightarrow s'w' \in P_i$, $w^k \subseteq x$, $w^{k+1} \nsubseteq x$, *for some* $k \geq 1$, *and* $y = (x - w^k) \cup w'^k$;
- $sx \Rightarrow_{max} s'y$ *iff* $sw_1 \rightarrow s'w'_1, \ldots, sw_k \rightarrow s'w'_k \in P_i$, $k \geq 1$, *such that* $w_1 \ldots w_k \subseteq x, y = (x - w_1 \ldots w_k) \cup w'_1 \ldots w'_k$, *and there is no* $sw \rightarrow s'w' \in P_i$, *such that* $w_1 \ldots w_k w \subseteq x$ *(note that rules can be combined only if they start from the same state* s *and end in the same state* s'*).*

Definition 11 (Transition steps). *Given two configurations* $C_1 = (s_1 w_1, \ldots, s_m w_m)$ *and* $C_2 = (s'_1 w''_1, \ldots, s'_m w''_m)$, *we write* $C_1 \Rightarrow_{\alpha,\beta} C_2$, *for* α *and* β *(as defined in Definition 9) if the conditions below are met.*

First, we apply rewriting steps (as defined in Definition 10) on each cell, i.e., $s_i w_i \Rightarrow_{\alpha(i,s_i)} s'_i w'_i$, $1 \leq i \leq m$.

Secondly, we define $z^{\uparrow}_{j,k}$, $z^{\downarrow}_{j,k}$, $z^{\leftrightarrow}_{j,k}$, *the outgoing multisets from* j *to* k, *where* $j \in \{1, \ldots, m\}$ *and, respectively,* $k \in \delta^{-1}(j)$, $k \in \delta(j)$, $k \in \delta(\delta^{-1}(j))\backslash\{j\}$*:*

- *If* $\beta(j, s_j) = repl$, *then*
  - $z^{\uparrow}_{j,k} = w'_j|_{\uparrow}$, *for* $k \in \delta^{-1}(j)$;
  - $z^{\downarrow}_{j,k} = w'_j|_{\downarrow}$, *for* $k \in \delta(j)$;
  - $z^{\leftrightarrow}_{j,k} = w'_j|_{\leftrightarrow}$, *for* $k \in \delta(\delta^{-1}(j))\backslash\{j\}$.
- *If* $\beta(j, s_j) = one$, *then*
  - $z^{\uparrow}_{j,k_j} = w'_j|_{\uparrow}$, *for an arbitrary* $k_j \in \delta^{-1}(j)$, *and* $z^{\uparrow}_{j,k} = \lambda$ *for* $k \in \delta^{-1}(j)\backslash\{k_j\}$;
  - $z^{\downarrow}_{j,k_j} = w'_j|_{\downarrow}$, *for an arbitrary* $k_j \in \delta(j)$, *and* $z^{\downarrow}_{j,k} = \lambda$ *for* $k \in \delta(j)\backslash\{k_j\}$;
  - $z^{\leftrightarrow}_{j,k_j} = w'_j|_{\leftrightarrow}$, *for an arbitrary* $k_j \in \delta(\delta^{-1}(j))\backslash\{j\}$, *and* $z^{\leftrightarrow}_{j,k} = \lambda$ *for* $k \in \delta(\delta^{-1}(j))\backslash\{j, k_j\}$.
- *If* $\beta(j, s_j) = spread$, *then*
  - $\{z^{\uparrow}_{j,k}\}_{k \in \delta^{-1}(j)}$ *is an arbitrary multiset partition of* $w'_j|_{\uparrow}$;
  - $\{z^{\downarrow}_{j,k}\}_{k \in \delta(j)}$ *is an arbitrary multiset partition of* $w'_j|_{\downarrow}$;
  - $\{z^{\leftrightarrow}_{j,k}\}_{k \in \delta(\delta^{-1}(j))\backslash\{j\}}$ *is an arbitrary multiset partition of* $w'_j|_{\leftrightarrow}$.

Finally, we set $w''_i = w'_i|_O \cup \bigcup_{j \in \delta^{-1}(i)} z^{\uparrow}_{j,i} \cup \bigcup_{j \in \delta(i)} z^{\downarrow}_{j,i} \cup \bigcup_{j \in \delta(\delta^{-1}(i))\backslash\{i\}} z^{\leftrightarrow}_{j,i}$, *for* $i \in \{1, \ldots, m\}$.

Definition 12 (Halting and results). *If no more transitions are possible, the hP system halts. For halted hP system, the computational result is the multiset that was cumulatively sent* out *(to the "environment") from the output cells* I_{out}. *The numerical result is the set of vectors consisting of the object multiplicities in the multiset result.*

Example 7. As examples, consider two hP systems, Π_3 and Π_4, both functional equivalent a functional equivalent of the earlier Π_2 nP system.

$\Pi_3 = (O, \sigma_1, \sigma_2, \sigma_3, \delta, I_{out})$, see Figure 9, where:

- $O = \{a\}$;
- $\sigma_1 = (\{s\}, s, a, \{sa \to sa_{\downarrow}, sa \to sa_{out}\})$;
- $\sigma_2 = (\{s\}, s, \lambda, \{sa \to sa_{\uparrow}\})$;
- $\sigma_3 = (\{s\}, s, \lambda, \{sa \to sa_{\uparrow}\})$;
- $\delta = \{(1,2), (1,3)\}$;
- $I_{out} = \{1\}$.

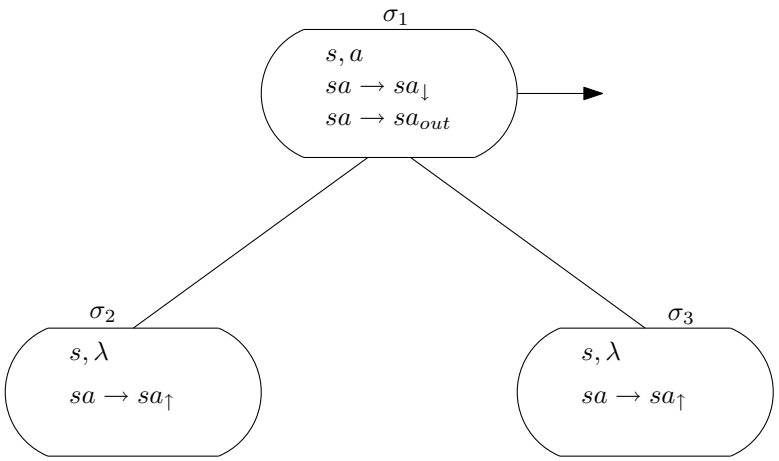


Fig. 9. Π_3, a simple hP system (equivalent to the Π_2 nP system of Figure 7).

$\Pi_4 = (O, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5, \delta, I_{out})$, see Figure 10, where:

- $O = \{a\}$;
- $\sigma_1 = (\{s\}, s, a, \{sa \to sa_{\leftrightarrow}, sa \to sa_{out}\})$;
- $\sigma_2 = (\{s\}, s, \lambda, \{sa \to sa_{\leftrightarrow}\})$;
- $\sigma_3 = (\{s\}, s, \lambda, \{sa \to sa_{\leftrightarrow}\})$;
- $\sigma_4 = (\{s\}, s, \lambda, \emptyset)$;
- $\sigma_5 = (\{s\}, s, \lambda, \emptyset)$;
- $\delta = \{(4,1), (4,2), (5,1), (5,3)\}$;
- $I_{out} = \{1\}$.

5 Relations Between P Systems, Neural P Systems and Hyperdag P Systems

Theorem 1 (Hyperdag P systems include non-dissolving P systems). *Any non-dissolving transition P system can be simulated by a hP system, with the same number of steps and object transfers.*

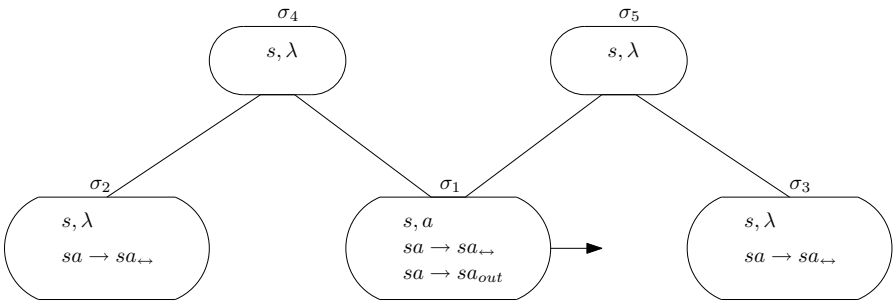


Fig. 10. Π_4, another simple hP system (equivalent to the Π_2 nP system of Figure 7).

Proof. Given a non-dissolving, transition P system Π [10], we build a functionally equivalent hP system H by the following transformation f. Essentially, we use the same elements, with minor adjustments. As the underlying structure, we can reuse the rooted tree structure of the P systems, because any rooted tree is a dag.

$\Pi = (O, C, \mu, w_1, \ldots, w_m, R_1, \ldots, R_m, i_o)$, $f(\Pi) = (O', \sigma'_1, \ldots, \sigma'_m, \delta, I_{out})$.

- $O' = O$;
- $\sigma'_i = (Q'_i, s'_{i,0}, w'_{i,0}, P'_i)$, $1 \leq i \leq m$, where:
  - $Q'_i = \{s\}$, where s is any symbol $\notin O$;
  - $s'_{i,0} = s$;
  - $w'_{i,0} = w_i$;
  - $P'_i = \{su \to sv' \mid u \to v \in R_i\}$, where v' is a translation of v by the following homomorphism: $(O \cup O \times Tar)^* \to O^*$, such that $a \to a$, $(a, here) \to a$, $(a, out) \to a_{\uparrow}$, $(a, in) \to a_{\downarrow}$;
- $\delta = \mu$;
- $I_{out} = \{i_o\}$;
- The object rewrite mode is the *max* constant function, i.e., $\alpha(i, s) = max$, for $i \in \{1, \ldots, m\}, s \in Q_i$;
- The object transfer mode is the *spread* constant function, i.e., $\beta(i, s) = spread$, for $i \in \{1, \ldots, m\}, s \in Q_i$.

Tags go-child($\downarrow$), go-parent($\uparrow$) correspond to P system target indications in, out, respectively. An empty tag corresponds to P system target indication $here$. Object rewrite and transfer modes of hP systems are a superset of object rewrite and transfer mode of P systems.

We omit here the rest of the proof which is now straightforward but lengthy.

Remark 1. We leave open the case of dissolving P systems, which can be simulated, but not properly subsumed by hP systems.

Proving that hP systems also cover nP systems appears more daunting. However, here we will use a natural interpretation of hP systems, where the bulk of the computing will be done by the sink nodes, and the upper nodes (parents) will function mostly as communication channels.

Remark 2. The combination of *go-sibling* ($\leftrightarrow$) with *repl* object transfer mode enable the efficient modeling of a communication *bus*, using only one hyperedge or, in the corresponding dag, n arcs. In contrast, any formal systems that use graph edges (or digraph arcs) to model 1:1 communication channels will need $n(n-1)$ separate edges (or $2n(n-1)$ arcs) to model the associated complete subgraph (clique). It is expected that this modeling improvement will also translate into a complexity advantage, if we use the number of messages measure. In hP systems, a local broadcast needs only one message to siblings, while needing $n-1$ messages in graph/digraph based systems.

Example 8. Figure 11 shows the structure of an hP system that models a computer network. Four computers are connected to "Ethernet Bus 1", the other four computers are connected to "Ethernet Bus 2", while two of the first group and two of the second group are at the same time connected to a wireless cell. In this figure we also suggest that "Ethernet Bus 1" and "Ethernet Bus 2" are themselves connected to a higher level communication hub, in a generalized hypergraph.

Example 9. Figure 12 shows the computer network of Figure 11, modeled as a graph (if we omit arrows) or as a digraph (if we consider the arrows). Note that the graph/digraph models, such as nP, do not support the grouping concept, i.e., there is no direct way to mark the nodes a, b, c, d as being part of the "Ethernet Bus 1", etc.

We can now sketch the proof of the theorem comparing hP systems and nP systems.

Theorem 2 (Hyperdag P systems can simulate bidirectional nP systems).
Any bidirectional nP system can be simulated by a hP system, with the same number of steps and object transfers.

Proof. Given a bidirectional nP system Π, we build a functionally equivalent hP system H by the following transformation f. As the underlying structure, we use a dag of height 1, where the cells are sink nodes, and the *syn* arcs are reified as height 1 nodes.

Without loss of generality, we assume that in the nP systems synapses are distinct from cells.

$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, i_{out})$, $f(\Pi) = (O', \sigma'_1, \ldots, \sigma'_{m+|syn|}, \delta, I_{out})$.

- $O' = O$;
- $\sigma'_i = f(\sigma_i)$, for $i \in \{1, \ldots, m\}$, where:
  - $Q'_i = Q_i$;
  - $s'_{i,0} = s_{i,0}$;
  - $w'_{i,0} = w_{i,0}$;
  - $P'_i = \{u \to v' \mid u \to v \in P_i\}$, where v' is a translation of v by the following homomorphism: $O^*_{tot} \to O^*$, such that $a \to a$, $a_{go} \to a_{\leftrightarrow}$, $a_{out} \to a_{out}$;

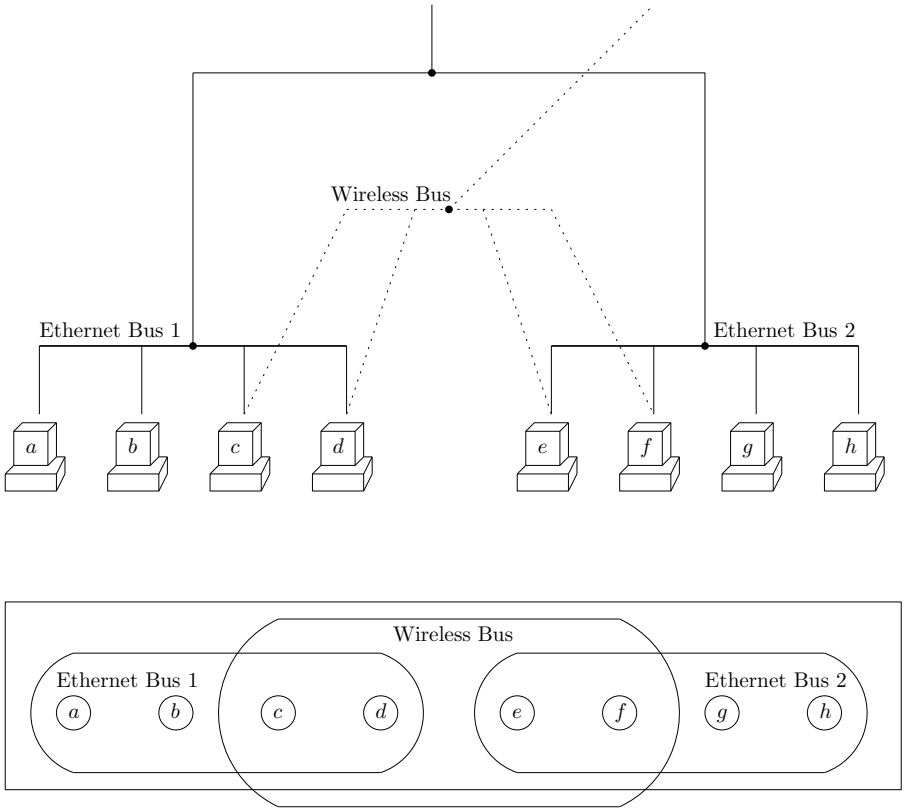


Fig. 11. A computer network and its corresponding hP/hypergraph representation.

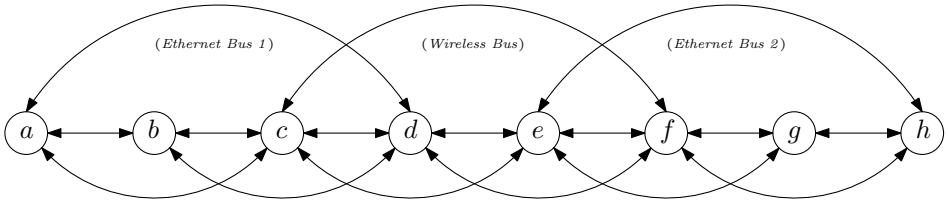


Fig. 12. The graph/digraph representations of the computer network of Figure 11.

- $\sigma'_{m+1}, \ldots, \sigma'_{m+|syn|}$ is an arbitrary ordering of elements in syn;
- $\delta = \{(e, u), (e, v) \mid e = (u, v) \in syn\}$;
- $I_{out} = \{i_{out}\}$;
- The object rewrite mode is a constant function, i.e., $\alpha(i, s) = \alpha_0$, for $i \in \{1, \ldots, m + |syn|\}, s \in Q_i$, where $\alpha_0 \in \{min, par, max\}$;

- The object transfer mode is a constant function, i.e., $\beta(i,s) = \beta_0$, for $i \in \{1,\ldots,m+|syn|\}, s \in Q_i$, where $\beta_0 \in \{repl, one, spread\}$.

Here the nodes corresponding to synapses are inactive as they just link neighboring cells.

Essentially, the cells keep their original nP rules, with minor adjustments. A *go* tag in nP rules corresponds to a sibling($\leftrightarrow$) tag in hP rules. Object rewrite and transfer modes of hP systems are a superset of object rewrite and transfer modes of nP systems.

We omit here the rest of the proof which is now straightforward but lengthy.

Remark 3. We leave open the case of non-bidirectional nP systems, which can be simulated, but not properly subsumed by hP systems.

6 Planar Representation of hP Systems

Classical tree-based P systems allow a "nice" planar representation, where the parent/child relationships between membranes are represented by Venn-like diagrams. Can we extend this representation to cover our dag-based hP systems?

In this section we will show that any hP system structurally based on a canonical dag can still be *intensionally* represented by hierarchically nested planar regions, delimited by Jordan curves (simple closed curves). Conversely, we also show that any set of hierarchically nested planar regions delimited by Jordan curves can be interpreted as a canonical dag, which can form the structural basis of a number of hP systems.

We will first show how to represent a canonical dag as a set of hierarchically nested planar regions.

Algorithm 3 (Algorithm for visually representing a canonical dag)

Without loss of generality, we consider a canonical dag (V,δ) of order n, where vertices are topologically ordered according to the order implied by the arcs, by considering parents before the children, i.e., $V = \{v_i \mid i \in \{1,\ldots,n\}\}$, where $(v_i, v_{i+1}) \in \delta$. Figure 13 shows side by side a simple height 1 canonical dag and its corresponding hypergraph representation. Note the *intensional* representation (as opposed to the *extensional* one): v_2 is not totally included in v_1, although all vertices included in v_2, i.e., v_4 and v_5, are also included in v_1. A possible topological order is v_1, v_2, v_3, v_4, v_5.

For each vertex v_i, we associate a distance $\psi_i = \frac{1}{2^{(n-i+1)}}$, for $i \in \{1,\ldots,n\}$. For Figure 13, $\psi_i = \frac{1}{32}, \frac{1}{16}, \frac{1}{8}, \frac{1}{4}, \frac{1}{2}$, for $i \in \{1,\ldots,n\}$.

We process the vertices in reverse topological order $v_n,\ldots,v_1$, at each step i representing the current vertex v_i by a planar region R_i.

First, we set parallel horizontal axis X_o and X_p, vertically separated by distance $3(n-1)$. Secondly, we set points $o_1,\ldots,o_n$ on X_o, such that o_i and o_{i+1} are separated by distance 3, for $1 \leq i \leq n-1$. We define o_i as the *origin point* of v_i,

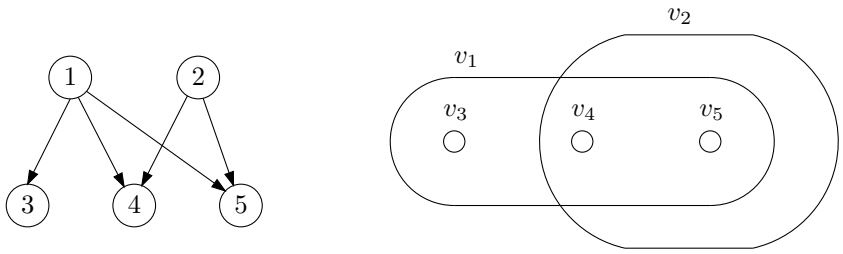


Fig. 13. A simple canonical dag and its corresponding hypergraph representation.

and write $o_i = origin(v_i)$. Finally, we set points $p_1, \ldots, p_n$ on X_p, such that p_i and p_{i+1} are separated by distance 3, for $1 \leq i \leq n-1$. We define p_i as the *corridor point* of v_i.

Figure 14 shows the construction of X_o, X_p, o_i and p_i, for the dag of Figure 13, where $n = 5$.

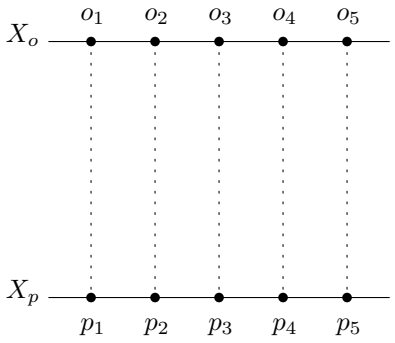


Fig. 14. Construction of X_o, X_c, o_i and p_i, for the dag of Figure 13, where $n = 5$.

If the current vertex v_i is a sink, then R_i is a circle with with radius $\frac{1}{2}$ centered at o_i.

If the current vertex v_i is a non-sink, then R_i is constructed as follows: Assume that the children of v_i are $w_1, \ldots, w_{n_i}$, and their (already created) regions are $S_1, \ldots, S_{n_i}$. Consider line segments $l_0, l_1, \ldots, l_{n_i}$, where l_0 is bounded by o_i and p_i, and l_j is bounded by p_i and $origin(w_j)$, for $j \in \{1, \ldots, n_i\}$. Let $L_0, L_1, \ldots, L_{n_i}$, $T_1, \ldots, T_{n_i}$ be the regions enclosed by Jordan curves around l_0, $l_1, \ldots, l_{n_i}$, $S_1, \ldots, S_{n_i}$, at a distance ψ_i, and let $R'_i = L_0 \cup \bigcup_{j=1,\ldots,n_i} L_j \cup \bigcup_{j=1,\ldots,n_i} T_j$. We define R_i as the external contour of R'_i. This definition will discard all internal holes, if any, without introducing any additional containment relations between our regions. The details of our construction guarantee that no internal hole will ever contain an origin point. □

Figure 15 shows an intermediate step (left) and the final step (right) of applying Algorithm 3 on Figure 13. The representation of Figure 15 is topologically

equivalent to the hypergraph representation of Figure 13 (right). Figure 16 shows the side by side, another dag and its corresponding planar region representation; internal holes are represented by dotted lines. Our objective here was not to create "nice" visualizations, but to prove that it is possible to represent an arbitrary canonical dag, i.e., an arbitrary hP system structurally based on a canonical dag, by hierarchically nested planar regions.

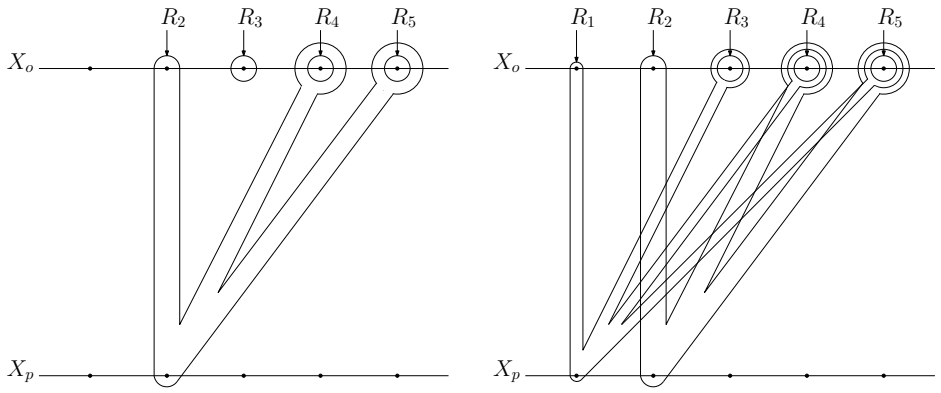


Fig. 15. An intermediate step and the final step of applying Algorithm 3 on Figure 13.

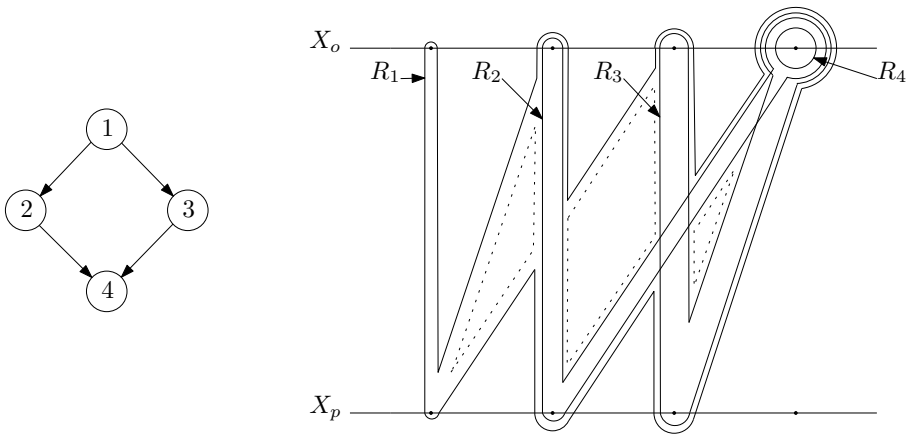


Fig. 16. A height two dag and its corresponding representation, built by Algorithm 3.

We will next show that, for any finite set of hierarchically nested planar regions, we can build a corresponding canonical dag (i.e., the underlying structure of a hP system).

Algorithm 4 (Algorithm for building a canonical dag from finite set of hierarchically nested planar regions)

Assume that we have n hierarchically nested planar regions,

1. Label each planar region by R_i, $i \in \{1, \ldots, n\}$,
2. If R_i directly nests R_j then draw an arc from a vertex v_i to a vertex v_j, $i, j \in \{1, \ldots, n\}$, $i \neq j$.

□

We now show that a canonical graph produced from Algorithm 4 does not contain any cycles. Our proof is by contradiction. Let us assume a directed graph G produced from Algorithm 4 contains a cycle $v_i, \ldots, v_k, \ldots, v_i$. Then every vertex in a cycle has an incoming arc. If vertex v_k is a maximal element in a cycle, with respect to direct nesting, then its corresponding planar region R_k have the largest region area among planar regions in a cycle. Since no other planar region in a cycle can contain R_k, there are no arc incident to vertex v_k. Hence, there is no cycle in G.

Remark 4. We leave open the problem of representing dags (that is hP systems) that contain transitive arcs.

7 Summary

We have proposed a new model, as an extension of P systems, that provides a better communication structure and we believe is often more convenient for modeling real-world applications based on tree structures augmented with secondary or shared communication channels.

We have shown that hP systems functionally extends the basic functionality of transition P systems and neural P systems, even though the underlying structure of hP systems is different. In the dag/hypergraph model of hP systems we can have a natural separation of computing cells (sinks) from communication cells (hyperedges). This model also allows us to easily represent multiple inheritance and/or to distribute computational results (as specified by a dag) amongst several different parts of a membrane structure.

We note that the operational behavior of hP systems is separate from the topological structure of a membrane system. In this paper, we illustrated hP systems using the computational rules of nP systems, where multisets of objects are repeatedly changed within cells, by using a fixed set of multiset rewriting rules, or transferred between cells, using several possible transfer modes.

Finally, we provided a intuitive visualization of hP systems, by showing that any set of hierarchically nested planar regions (which represents any set of cells ordered by containment) is equivalent to, or modeled by, a dag without transitive arcs. We provided simple algorithms to translate between these two interpretations.

Part B of this paper will explore this model further and support it with a more extensive set of examples.

References

1. C. Berge: *Hypergraphs. Combinatorics of Finite Sets*. Elsevier Science Publishers, 1989.
2. C. Carathéodory: *Theory of Functions of a Complex Variable*. Vol. 1, Chelsea Publishing Company, 1954.
3. G. Ciobanu: Distributed algorithms over communicating membrane systems. *BioSystems*, 70, 2 (2003), 123–133.
4. W.F. Doolittle: Uprooting the tree of life. *Scientific American*, 282 (2000), 90–95.
5. T.-O. Ishdorj, M. Ionescu: Replicative-distribution rules in P systems with active membranes. *Proceedings of the First International Colloquium on Theoretical Aspects of Computing (ICTAC 2004)*, 68–83.
6. C. Li: *Master Thesis*. The University of Auckland, Supervisor: R. Nicolescu, 2008.
7. C. Martín-Vide, G. Păun, J. Pazos, A. Rodríguez-Patón: Tissue P systems. *Theoretical Computer Science*, 296, 2 (2003), 295–326.
8. G. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143 (and Turku Center for Computer Science, TUCS Report 208, November 1998).
9. G. Păun: *Membrane Computing–An Introduction*. Springer-Verlag, 2002.
10. G. Păun: Introduction to membrane computing. *Proceeding of the First Brainstorming Workshop on Uncertainty in Membrane Computing*, 2004, 17–65.
11. G. Păun, R.A. Păun: Membrane computing as a framework for modeling economic processes. *Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2005)*, 11–18.
12. M. Slikker, A. Van Den Nouweland: *Social and Economic Networks in Cooperative Game Theory*. Kluwer Academic Publishers, 2001.
13. V.I. Voloshin: *Coloring Mixed Hypergraphs: Theory, Algorithms and Applications*. American Mathematical Society, 2002.
14. C. Zandron, C. Ferretti, G. Mauri: Solving NP-complete problems using P systems with active membranes. *Proceedings of the Second International Conference on Unconventional Models of Computation*, 2000, 289–301.

Spiking Neural P Systems and Modularization of Complex Networks from Cortical Neural Network to Social Networks

Adam Obtułowicz

Institute of Mathematics, Polish Academy of Sciences
Śniadeckich 8, P.O.B. 21, 00-956 Warsaw, Poland
A.Obtulowicz@impan.pl

Summary. An idea of modularization of complex networks (from cortial neural net, Internet computer network, to market and social networks) is explained and some its topic motivations are presented. Then some known modularization algorithms and modular architectures (constructions) of complex networks are discussed in the context of possible applications of spiking neural P systems in order to improve these modularization algorithms and to analyze massively parallel processes in networks of modular architecture.

1 Introduction

The aim of this paper is to discuss certain interconnections between spiking neural P systems [16], [28], and an idea of modularization of complex networks from cortial neural net, Internet computer network, to market and social networks, where the idea of modularization comprises modular architectures (structures or constructions) of those networks and modularization algorithms for retrieving modular structure of networks. The interconnections are understood here as proposals of application of spiking neural P systems to improve the modularization algorithms and to analyze massively parallel processes in networks of modular architecture or construction (emergence of new modules, etc., [9], [23]).

In Section 2 we explain the idea of modularization of complex networks and some its topic motivations. In Section 3 we outline open problems concerning improvement of some modularization algorithms by application of spiking neural P systems and investigations of massively parallel processes in networks of modular construction by applying these P systems.

2 Modularization of Complex Networks and Its Topic Motivations

A *modularization*[1] of a complex network or a graph is understood to be a decomposition or a partition of the underlying set of nodes of the network or the graph, respectively, into subsets called *modules*, often identified with subnetworks determined by these subsets and treated as autonomous processing units in cooperation with other units (on a higher level if abstraction). A collection of modules of a given network or a graph can be also a subject of modularization, i.e. a decomposition into subcollections of modules, etc., where the resulting subcollections of modules are called *higher level modules*.

There are many reasons, motivations, and practical applications of modularization and we outline here some topics:

1) cortial neural network is modularized from anatomical, physiological, and scale (or magnitude) reasons, see, e.g., [24] or [27] for more references, into
   - cortial minicolumns which are modules consisting of neurons,
   - cortial hypercolumns which are some sets of minicolumns,
   - cortial areas which are some sets of hypercolumns,

   where cortial hypercolumns and areas are higher level modules,
2) natural self-modularization of cortial neural network into neuronal groups during evolution process described by M. G. Edelman's Theory of Neuronal Group Selection (Neuronal Darwinism), see [18], [17] for a spiking neural network version,
3) modularization of cortial neural network into assemblies of neurons appears useful for neuronal representation of cognitive functions and processes because:
   - a single neuron behavior is less certain or more noisy than a behavior of an assembly of neurons,
   - the number of synaptic connections of a single neuron with other neurons is smaller than that of an assembly of neurons with other assemblies of neurons,
   - according to M. Kaiser [20] hierarchical cluster (higher level module) architecture "may provide the structural basis for the stable and diverse functional patterns observed in cortial networks",
4) emergence (or extraction) of community structures in social networks, biological networks, and Internet computer network is a modularization of these networks discussed by M. J. E. Newman, [6], [7], [26], [29], see also applications of similar modularization in city planning discussed by Ch. Alexander [2],
5) modularization of artificial cortial-like networks for image processing, e.g., regularization for improving segmentation, see J. A. Anderson and P. Sutton, cf. [21], applications of an idea of a Network of Networks (NoN),
6) modularization which gives rise to hierarchical and fractal graphs and networks, [25], [29], [30], [33], to compress the information contained in large complex networks.

[1] The term 'modularization' is used e.g., in [19].

The higher level modules and their motivation are also discussed in [10], [11], [12], [22], [32], [31].

3 Modularization Algorithms and Modular Architectures

In the cases 2)–5) algorithms of modularization are considered, i.e. algorithms of distinguishing or extraction of modules, see e.g., [3], [18]. Thus one asks for those spiking neural P systems which could realize these algorithms through massive parallelism of computations providing

- efficiency of computation,
- those computation processes which could be close (from simulation reason) to real distributed processes of emergence of neuronal groups (see [18]) or community structures in social networks, where distributed processes of emergence of neuronal groups are massively parallel processes of simultaneous emergence of many those groups which are autonomous understood that, e.g., each group has at least one neuron which does not belong to other groups.

These spiking neural P systems could give rise to constructing new brain-based devices (robots) similar to those which belong to the family Darwin due to M.G. Edelman [8]. The new brain-based devices could simulate maturing processes, where emergence of neuronal groups and groups of groups give rise to new cognitive functions.

We show now an example of a link between modularization algorithms and spiking neural P systems which suggests the proposed above applications of these P systems. Namely, basing on the algorithm for identification of neuronal groups described in [18] we outline a method of extraction of a process of simultaneous emergence of many neuronal groups from a process generated by a spiking neural P system.

Let $\mathcal{S}\ [\ \mathcal{C} > \mathcal{S}'$ be the next state relation determined by simultaneous application (in maximal parallelism mode) of the rules of a spiking neural P system, where $\mathcal{S}, \mathcal{S}'$ are spike contents of neurons of the system and $\mathcal{C}$ is the set of those synapses of the system which are activated to transform $\mathcal{S}$ into $\mathcal{S}'$ according to some maximal consistent set of the rules of the system during a unit of time. For a finite process generated by a spiking neural P system and represented by

$$\mathcal{S}_0\ [\ \mathcal{C}_1 > \mathcal{S}_1\ [\ \mathcal{C}_2 > \mathcal{S}_2 \ldots \mathcal{S}_{n-1}\ [\ \mathcal{C}_n > \mathcal{S}_n \quad (n > 2) \tag{1}$$

we extract from it a process of simultaneous emergence of many neuronal groups which is represented by a sequence $\mathcal{G}_1 \ldots \mathcal{G}_{i^*}$ of sets of synapses of the system such that

- $\mathcal{G}_1$ is the set of maximal (with respect to inclusion relation of sets) subsets x of $\mathcal{C}_1$ such that the synapses in x have a common source neuron which is a counterpart of an anchor neuron, see the first step of the algorithm in [18],

- for $i > 1$ we define $\mathcal{G}_i$ to be the set of maximal sets in the collection

$$\mathcal{K}_i = \Big\{ y \;\Big|\; x \subsetneq y = x \cup \{ s \in \mathcal{C}_i \,|\, \text{the source neuron of synapse } s \text{ is the target neuron of some synapse in } x \} \text{ for some } x \in \mathcal{G}_{i-1} \Big\}$$

  until this collection is non-empty or, equivalently, until $i = i^*$, where i^* is the greatest number for which $\mathcal{K}_{i^*}$ is non-empty.

The elements of $\mathcal{G}_{i^*}$ represent neural circuits which correspond to neuronal groups emerging simultaneously in the process represented by (1).

Since the networks and their modularization discussed in 2)–4) are also approached by using probabilistic and statistical methods of clustering (see [1]) and by using random graphs (understood as in the B. Bolobas book [4]), it is worth to discuss a concept of a stochastic (or random) spiking neural P system whose synaptic connections form a random graph.

Besides the new applications of spiking neural P systems suggested above one could ask for an application of M.-A. Guttiérez-Naranjo and M. Pérez-Jiménez models for Hebbian learning with spiking neural P systems [15] to explain in a new way

- temporal correlation hypothesis of visual feature integration [14], also dealing with modularization, where modules are neural assemblies emerging in distributed processes like the processes of emergence of neuronal groups described above, for the connections of neuronal groups and binding (some generalization of feature integration), see [17].
- emergence of brain cognitive functions, where some modularizations of cortial neural network are considered, see [5], [13].

We propose to use higher-level networks with neighbourhood graphs, introduced in [27], as a precise description of results of some modularizations of networks and modular architectures including higher level modules.

References

1. Albert, R., Barabási, A.-L., *Statistical mechanics of complex networks*, Reviews of Modern Physics 74 (2002), pp. 47–97.
2. Alexander, Ch., *Notes on the Synthesis of Form*, second ed., Harvard Univ. Press, Cambridge, MA, 1971.
3. Boccaletti, S., et al., *Complex networks*: *Structure and dynamics*, Physics Reports 424 (2006), pp. 175–308.
4. Bolobas, B., *Random Graphs*, Academic Press, 1985.
5. Changeux, J.-P., Dehaene, S., *The neuronal workspace model*: *Conscious processing and learning in Learning Theory and Behaviour*, in: Learning Memory. A Comprehensive Reference, ed. R. Menzel, vol. 1, 2008, pp. 729–758.

6. Clauset, A., Moore, Ch., Newman, M. E. J., *Structural inference of hierarchies in networks*, in: Proceedings of 23rd International Conference on Machine Learning, Pittsburgh, PA, 2006.
7. Clauset, A., Moore, Ch., Newman, M. E. J., *Hierarchical structure and prediction of missing links in networks*, Nature 453 (2008), pp. 98–101.
8. Edelman, G. M., *Learning in and from brain-based devices*, Science 318 (16 November 2007), pp. 1103–1105.
9. Fernando, Ch., Karishma, K. K., Szathmary, E., *Copying and evolution of neuronal topology*, PLOS One 3 (November 2008), Issue 11, e3775.
10. Fingelkurts, An. A., Fingelkurts, Al. A., *Operational architectonics of perception and cognition* (*a principle of self-organized metastable brain states*), presented at the VI Parmenides Workshop of Institute of Medical Psychology of Munich University, Elba, Italy, April 5 to 10, 2003.
11. Fingelkurts, An. A., Fingelkurts, Al. A., *Mapping of brain operational architectonics*, in: Focus on Brain Mapping Research, ed. F. J. Chen, 2005, pp. 59–98.
12. Fingelkurts, An. A., Fingelkurts, Al. A., Kahkonen, S., *New perspectives in pharmaco-electroencephalography*, Progress in Neuro-Psychopharmacology & Biological Psychiatry 29 (2005), pp. 193–199.
13. Goldman-Rakic, P. S., *Topography of cognition*: *parallel distributed networks in primate association cortex*, Annual Rev. Neurosc. 11 (1988), pp. 137–156.
14. Gray, C. M., *The temporal correlation hypothesis of visual feature integration still alive and well*, Neuron 24 (1999), pp. 31–47.
15. Gutierez-Naranjo, M. A., Perez-Jimenez, M. J., *A spiking neural P systems based model for Hebbian learning*, in: Proceedings of 9th Workshop on Membrane Computing, Edinburgh, July 28 – July 31, 2008, ed. P. Frisco et al., Technical Report HW-MASC-TR-0061, School of Mathematical and Computer Sciences, Heriot–Watt University, Edinburgh, UK, 2008, pp. 189–207.
16. Ionescu, M., Păun, Gh., Yokomori, Y., *Spiking neural P systems*, Fund. Inform. 71 (2006), pp. 279–308.
17. Izhikevich, E. M., *Polychronization*: *computation with spikes*, Neuronal Computation 18 (2006), pp. 245–282.
18. Izhikevich, E. M., Gally, J. A., Edelman, G. M., *Spike-timing dynamics of neuronal groups*, Cerebral Cortex 14 (2004), pp. 933–944.
19. Johansson, Ch., Lasner, A., *A hierarchical brain inspired computing systems*, in: Proceedings NOLTA 2006, 11–14 September 2006, Bologna, Italy, pp. 599–602.
20. Kaiser, M., *Brain architecture*: *a design for natural computation*, Phil. Trans. R. Soc. A 365 (2007), pp. 3033–3045.
21. Ling Guan, Anderson, J. A., Sutton, J. P., *A network of networks processing model for image regularization*, IEEE Transactions on Neural Networks 8 (1997), No. 1, pp. 169–174.
22. Mason, R. D., Robertson, W., *Mapping hierarchical neural networks to VLSI hardware*, Neural Networks 8 (1995), pp. 905–913.
23. Moore, Ch., Ghoshal, G., Newman, M. E. J., *Exact solution for models of evolving networks with addition and deletion of nodes*, arXiv:cond-mat/0604069v1 [cond-math.stat-mech], 4 April 2006.
24. Mountcastle, V. B., *The columnar organization of the neocortex*, Brain 120 (1997), pp. 701–722.
25. Müller-Linow, M., Hilgetag, C. C., Hütt, M.-T., *Organization of excitable dynamics in hierarchical biological networks*, PLOS Computational Biology 4 (2008), Issue 9, e100190.

26. Newman, M. E. J., *The structure and function of complex networks*, SIAM Review 45 (2003), pp. 167–256.
27. Obtułowicz, A., *On mathematical modeling of anatomical assembly, spatial features, and functional organization of cortex by application of hereditarily finite sets*, in: Proceedings of 9th Workshop on Membrane Computing, Edinburgh, July 28 – July 31, 2008, ed. P. Frisco et al., Technical Report HW-MASC-TR-0061, School of Mathematical and Computer Sciences, Heriot–Watt University, Edinburgh, UK, 2008, pp. 371–382.
28. Păun, Gh., Perez-Jimenez, M. J., *Spiking neural P systems. Recent results, research topics*, in: 6th Brainstorming Week on Membrane Computing, Sevilla 2008, web page.
29. Ravasz, E., Barabási, A.-L., *Hierarchical organization in complex networks*, Physical Review E 67 (2003), 026112.
30. Sporns, O., *Small–world connectivity, motif composition, and complexity of fractal neuronal connections*, BioSystems 85 (2006), pp. 55–64.
31. Sporns, O., Chialvo, D. R., Kaiser, M., Hilgetag, C. C., *Organization, development and function of complex brain networks*, Trends in Cognitive Sciences 8 (2004), pp. 418–425.
32. Sporns, O., Tononi, G., Edelman, G. M., *Theoretical neuroanatomy: relating anatomical and functional connectivity in graphs and cortial connection matrices*, Cerebral Cortex 10 (2000), pp. 127–141.
33. Wuchty, S., Ravasz, E., Barabási, A.-L., *The Architecture of Biological Networks.*

The Discovery of Initial Fluxes of Metabolic P Systems

Roberto Pagliarini, Vincenzo Manca

Verona University
Computer Science Department
Strada Le Grazie 15, 37134 Verona, Italy.
{roberto.pagliarini,vincenzo.manca}@univr.it

Summary. A central issue in systems biology is the study of efficient methods to infer fluxes of biological reactions starting from experimental data. Among the different techniques proposed in the last years, in the theory of *Metabolic P systems Log-Gain* principles have been introduced, which prove to be helpful for deducing biological fluxes from temporal series of observed dynamics. However, crucial tasks remain to be performed for a complete suitable application of these principles. In particular the algebraic systems introduced by the Log-Gain principles require the knowledge of the initial fluxes associated with a set of biochemical reactions. In this paper we propose an algorithm for estimating initial fluxes, which is tested in two case studies.

1 Introduction

In the last years, the problem of reverse-engineering of biological phenomena from experimental data has spurred increasing interest in the scientific communities. For these reasons, many computational models inspired from biology have been given. Among these models, the *Metabolic P systems* [9, 10], shortly *MP systems*, proved to be relevant in the analysis of dynamics of biochemical processes [4, 12, 14, 13]. MP systems intend to model metabolic systems, that is, structures where matter of different types is subject to reactions, or transformations of various types. The importance of these computational models is their potential applicability to the reverse-engineering problem of biological phenomena. In fact, the MP systems introduce a theory, called *Log-Gain* [8], intrinsically related to the structure of these computational models.

This theory provides a method for constructing MP models of real phenomena from time-series of observed dynamics. In fact, given a real system which can be observed in its evolution, then almost all the elements occurring in the definition of MP system can be, in principle, deduced by macroscopic observations of the system [9].

The only component which cannot be directly measured is a set of regulation functions which state the reaction fluxes, that is, the amount of reactants transformed by the reactions at any state of the system. These functions depend on internal microscopic processes on which molecules are involved. However, Log-Gain theory provides a way for deducing them from time-series of the states of a given system along a sufficient number of observation instants.

A key point for achieving this task consists in the discovery of the fluxes associated to the passage of a metabolic system from two initial observation instants. In this paper an algorithm is proposed for estimating these initial fluxes from few steps of observation.

The present paper is organized as follows. Section 2 is devoted to the definition of Metabolic P Systems. Section 3 briefly recalls the Log-Gain theory. In Section 4 we describe the algorithm that solves our problem. Section 5 reports some experimental results obtained by the new framework. Some further remarks and directions for future researches are discussed in the last section.

2 Metabolic P Systems

MP systems are a special class of dynamical systems (the reader can find some details concerning dynamical aspects of MP systems in [11]), based on *P systems* [3, 16, 17, 18], which are related to metabolic processes. MP systems are essentially constituted by multiset grammars where rules are regulated by specific functions depending on the state of the system. From a Membrane Computing point of view, MP systems can be seen as deterministic mono-membrane P systems where the transitions between states are calculated by a suitable recurrent operation. In an MP system the overall variation, in a macroscopic time interval, of the whole system under investigation is considered. In this manner, the evolution law of the system consists in the knowledge of the contribution of each reaction in the passage between any two instants separated by such an interval. Therefore, dynamics is given at discrete steps, and at each step, it is ruled by a partition of matter among the reactions transforming it. The principle underlying the partitioning is called *mass partition principle*. This principle replaces the *mass action law*[1] of ODE systems. The mass partition principle defines the transformation rate of object populations rather than single objects, according to a suitable generalization of chemical laws [9].

[1] The foundation of this law is the theory of molecular collisions. The first formulation of this law, formulated by Waage and Guldberg [21], is the following: *"the rate of any given chemical reaction is proportional to the product of the concentrations of the reactants"*.

2.1 MP systems: a formal definition

The following definition introduces the MP systems in a formal way ($\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$ respectively denote the sets of natural, integer, and real numbers).

Definition 1 (MP system) *An MP system M is specified by the following construct:*

$$M = (X, R, V, H, \Phi, \nu, \mu, \tau, \delta)$$

where X, R and V are finite disjoint sets, and moreover the following conditions hold, with $n, m, k \in \mathbb{N}$:

- *$X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of substances. This set represents the types of molecules;*
- *$R = \{r_1, r_2, \ldots, r_m\}$ is a finite set of reactions. A reaction r is a pair of type $\alpha_r \rightarrow \beta_r$, where α_r identifies the multiset of the reactants (substrates) of r and β_r identifies the multiset of the products of r (λ represents the empty multiset). The stoichiometric matrix $\mathbb{A}$ of a set R of reactions over a set X of substances is $\mathbb{A} = (\mathbb{A}_{x,r} \mid x \in X, r \in R)$ with $\mathbb{A}_{x,r} = |\beta_r|_x - |\alpha_r|_x$, where $|\alpha_r|_x$ and $|\beta_r|_x$ respectively denote the number of occurrences of x in α_r and β_r. Of course, a reaction r can be seen as the vector $r = (\mathbb{A}_{x,r} \,|x \in X)$ of $\mathbb{Z}^n$. We also set $R_\alpha(x) = \{r \in R \mid x \in \alpha_r\}$, $R_\beta(x) = \{r \in R \mid x \in \beta_r\}$, and $R(x) = R_\alpha(x) \cup R_\beta(x)$;*
- *$V = \{v_1, v_2, \ldots, v_k\}$ is a finite set of parameters (such as pressure, temperature,...);*
- *$H = \{h_v \mid v \in V\}$ is a set of parameters evolution functions. The function $h_v : \mathbb{N} \rightarrow \mathbb{R}$ states the value of parameter v, and $H[i] = (h_v(i) \,|v \in V)$;*
- *$\Phi = \{\varphi_r \mid r \in R\}$ is the set of flux regulation maps, where, for each $r \in R$, $\varphi_r : \mathbb{R}^{n+k} \rightarrow \mathbb{R}$. Let $q \in \mathbb{R}^n$ be the vector of substances values and $s \in \mathbb{R}^k$ be the vector of parameters values. Then $(q, s) \in \mathbb{R}^{n+k}$ is the state of the system. We set by $U(q, s) = (\varphi_r(q, s) \mid r \in R)$ the flux vector in the state (q, s);*
- *ν is a natural number which specifies the number of molecules of a (conventional) mole of M;*
- *μ is a function which assigns, to each $x \in X$, the mass $\mu(x)$ of a mole of x (with respect with to some measure units);*
- *τ is the temporal interval between two consecutive observation steps;*
- *$X[i] = (x_1[i], x_2[i], \ldots, x_n[i])$, for each $i \in \mathbb{N}$, is the vector of substances values at the step i, and $X[0]$ are the initial values of substances. The dynamics $\delta : \mathbb{N} \rightarrow \mathbb{R}^n$ of the system is completely identified by the following recurrent equation, called Equational Metabolic Algorithm shortly EMA:*

$$X[i+1] = \mathbb{A} \times U(X[i], H[i]) + X[i] \tag{1}$$

where $\mathbb{A}$ is the stoichiometric matrix of reactions having dimension $n \times m$, while $\times$, $+$, are the usual matrix product and vector sum. We denote by $EMA[i]$ the system (1) at the step i. By using the formulation introduced above it is simple to note that we can obtain the vector $X[i+1]$ from vectors $X[i]$ and $U(X[i], X[i])$.

3 Log-Gain Theory: A Brief Recall

The starting point of the Log-Gain theory for MP systems [20] is the *Allometry Law* [2, 6] which assumes a proportion between the relative variations of the fluxes of a reaction r and the sum of relative variations of *tuners* of r, that is, magnitude influencing r.

The relative variation of a variable x is given, in differential notation and with respect to the time variable t, by $d(\lg x)/dt$. This explains the term "Log-Gain".

Given a dynamics of an MP system, we will use the following simplified notations, for $i \in \mathbb{N}$, and $r \in R$:

$$u_r[i] = \varphi_r(X[i], H[i]) \quad \text{and} \quad U[i] = (u_r[i] | r \in R) \tag{2}$$

Assuming to know the vectors $X[i]$ and $X[i+1]$, the equation (1) can be rewritten in the following form, which we called $ADA[i]$ (Avogadro and Dalton Aggregation [10]):

$$X[i+1] = \mathbb{A} \times U[i] + X[i] \tag{3}$$

The formula (3) expresses a system of n equations and m variables (n is the number of substances and m the number of reactions) which is assumed to have maximal rank. This supposition is not restrictive. In fact, if it does not hold the rows which are linearly depend on other rows are removed. Formally $ADA[i]$ is the same to system $EMA[i]$ introduced in Section 2. However, these two systems have dual interpretations. In fact, in $EMA[i]$, the vectors $U[i]$ and $X[i]$ are known, and the vector $X[i+1]$ is computed by means of them, while in $ADA[i]$, the vector $X[i+1] - X[i]$ is known and $U[i]$ is computed by solving the system, as we will see by formula (6).

Usually, in a biochemical phenomenon, the number of reactions is greater than the number of substances, and this means that the system (3) has more than one solution. Therefore, fluxes cannot be univocally deduced by means of ADA. The following principle [8] allows us to add more equations to the above system in order to get a univocally solvable system which could provide the flux vector.

Definition 2 (Discrete Log-Gain) *Let $(z[i] \,|\, i \in \mathbb{N})$ a real valued sequence. Then, the discrete log-gain of z is given by the following equation:*

$$Lg(z[i]) = \frac{z[i+1] - z[i]}{z[i]} \tag{4}$$

Principle 1 (Log-Gain regulation) *Let $U[i]$, for $i \geq 0$, be the vector of fluxes at step i. Then the Log-Gain regulation can be expressed in terms of matrix and vector operations:*

$$(U[i+1] - U[i])/U[i] = \mathbb{B} \times L[i] \quad + C \otimes P[i+1] \tag{5}$$

where:

- $\mathbb{B} = (p_{r,z} \,|r \in R, z \in X \cup V\,)$ *where* $p_{r,z} \in \{0,1\}$ *with* $p_{r,z} = 1$ *if* z *is a tuner of* r *and* $p_{r,z} = 0$ *otherwise;*
- $L[i] = (Lg(z[i]) \,|z \in X \cup V\,)$ *is the column vector of substances and parameters log-gains ;*
- $P[i+1]$ *is a column vector of values associated with the reactions and called (Log-Gain) offsets at step* $i+1$*;*
- $C = (c_r \,|r \in R)$*, where* $c_r = 1$ *if* $r \in R_0$*, while* $c_r = 0$ *otherwise, and* R_0 *is a subset of reactions having the Covering Offset Log Gain Property, that is, it is a set of* n *linear independent vectors of* $\mathbb{Z}^n$*;*
- $\times$ *denotes the usual matrix product;*
- $+, -, /, \otimes$ *denote the component-wise sum, subtraction, division and product*[2]*.*

If we assume to know the flux unit vector at step i and put together the equations (5) and (3) at steps i and $i+1$ respectively, we get the following linear system called *Offset Log-Gain Adjustment* module at step i, shortly $OLGA[i]$, in which the number of variables (here reported in bold font) is equal to the number of equations:

$$\begin{aligned} \mathbb{A} \times \mathbf{U[i+1]} &= X[i+2] - X[i+1] \\ (\mathbf{U[i+1]} - U[i])/U[i] &= \mathbb{B} \times L[i] \quad + \quad C \otimes \mathbf{P[i+1]} \end{aligned} \tag{6}$$

Now, if the vectors $X[i]$ and $V[i]$, for $0 \leq i \leq l$, where $l \in \mathbb{N}$, are obtained by experimental measures, then it is possible to solve $OLGA[i]$ for $i = 0, \ldots, l-1$, obtaining the vector $U[i]$ for $i \in [1, l-1]$.

4 An Algorithm for the Estimation of Initial Metabolic Fluxes

The method described in the previous section assumes the knowledge of the initial values of fluxes.

Problem 1 (Initial Fluxes Problem) *Given* $X[0]$ *and* $H[0]$*, find a flux vector* $U[0]$ *such that it satisfies the initial dynamics, that is:*

$$X[1] \cong \mathbb{A} \times U[0] + X[0]$$

where $\cong$ *means approximate equality.*

The algorithm given below circumvents the Initial Fluxes Problem by using the knowledge about the dynamics in the first evolution steps in order to approximate the amount of substances which is not transformed, we call *inertia* of the system (at a given step).

[2] Given two $n \times m$ matrices A and B, the operation $A \otimes B$ involves the action of multiplying component-wise each element of A by the corresponding element of B.

4.1 The proposed algorithm

Our approach is based on the assumption that if the inertia of each substance is known, then only a part of substances has to be partitioned among the reactions which require to consume them. The main steps of the algorithm are described in the following of this section.

Step 1.

The goal of the first step is to evaluate grossly the initial fluxes at the step 0 by assuming that they are proportional to the reactants, that is, for all $r \in R$:

$$\hat{u}_r[i] = k_r y_r[i] \tag{7}$$

where $k_r \in \mathbb{R}$, and $y_r[i]$ is the product of all substance quantities, at the step i, which are reactants for r. We suppose that if $\alpha_r = \lambda$ then $y_r = 1$, and we set

$$\hat{U}[i] = (\hat{u}_r[i] \mid r \in R) \tag{8}$$

For example, in a metabolic system with three kinds of substances, a, b, c, and with a set of reactions given in the first column of the Table 1, the relationships among the fluxes of these reactions and their reactants are reported in the second column of the Table 1.

Let us consider the following system, called *Local-Stoichiometric Module* at the

Reactions	Maps
$r_1 : a \to bc$	$k_{r_1} a$
$r_2 : b \to a$	$k_{r_2} b$
$r_3 : c \to ab$	$k_{r_3} c$
$r_4 : c \to cc$	$k_{r_4} c$

Table 1. Reactions and their flux regulation maps of the Local-Stoichiometric Module.

step i:

$$x[i+1] - x[i] = \sum_{r \in R(x)} \mathbb{A}_{x,r} \hat{u}_r[i] \quad \forall x \in X \tag{9}$$

If we assume that the constants k_r, with $r \in R$, do not sensibly change in few steps, then by applying the system (9) for a sufficient number of steps we can obtain a square linear system of dimension m having maximum rank. In the example reported in Table 1, we have a Local-Stoichiometric Module of 3 equations which initially has 4 unknowns. It has rank 3. At the second iteration of this module we get other 3 equations and the rank of Local-Stoichiometric Module is maximum. Thus, we can obtain a system of equation having unique solution. In general, if we start with the Local-Stoichiometric Module at the step 0 then we can compute the vector $\hat{U}[0] = (\hat{u}_r[0] \mid r \in R)$ by applying the Local-Stoichiometric module a suitable number of steps.

Step 2.

The aim of this step is to approximate the inertia of the system. We split this step in two sub-steps. In the first one we take n linear independent reactions, obtaining a set R_0, according to the Covering Offset Log Gain Property. Then, we use the set R_0 to obtain an $OLGA[1]$ module, with $U[0] = \hat{U}[0]$, where $\hat{U}[0]$ is the vector of fluxes computed in the previous step. We will indicate with $U^*[1] = (u_r^*[1] \mid r \in R)$ the solution of this system. However, if some elements of this vector is a negative real value, then we choose others n linear independent reactions and reapply the procedure above describe (it easy to prove that a positive vector must exist).

In the second sub-step we compute, for each $x \in X$, the inertia, indicated by $\bar{x}$, by applying the following equation:

$$\bar{x}[1] = x[1] - \sum_{r \in R_\alpha(x)} u_r^*[1], \quad \forall x \in X \tag{10}$$

Step 3

In the last step we obtain the vector of fluxes at the evolution step 1 by solving an optimization problem. In fact, the vector $U^\circ = (u_r^\circ \mid r \in R)$ we search has to be a strictly positive vector of $\mathbb{R}^m$ (positive in each component) which satisfies the following n equations:

$$x[1] - \bar{x}[1] = \sum_{r \in R_\alpha(x)} u_r^\circ[1], \quad \forall x \in X \tag{11}$$

and it is bounded, for each component, by the following constraint:

$$u_r^\circ[1] \leq \begin{cases} \min\left\{ \frac{x_j[1] - \bar{x}_j[1]}{|\alpha_r|_{x_j}} \mid x_j \in \alpha_r \right\} & \text{if} \quad \alpha_r \neq \lambda \\ k_r & \text{if} \quad \alpha_r = \lambda \end{cases} \tag{12}$$

and such that

$$U^\circ = \min_{\xi \in \mathbb{R}^m} \|\mathbb{A} \times \xi - (X[2] - X[1])\| \tag{13}$$

5 Experiments

In this section, in order to evaluate the performance of our algorithm, we apply it to two case studies: *i)* a synthetic oscillatory metabolic system, and *ii)* the Belousov-Zhabotinsky reaction [1, 7, 19, 22].

Reactions	Flux regulation maps
$r_1 : a \to aa$	$\varphi_1 = k_1 a/(k_1 + k_2 c + k_4 b + k_6)$
$r_2 : a \to b$	$\varphi_2 = k_2 ac/(k_1 + k_2 c + k_4 b + k_6)$
$r_3 : b \to \lambda$	$\varphi_3 = k_3 b/(k_3 + k_6)$
$r_4 : a \to c$	$\varphi_4 = k_4 ab/(k_1 + k_2 c + k_4 b + k_6)$
$r_5 : c \to \lambda$	$\varphi_5 = k_5 c/(k_5 + k_6)$
$X[0] = (100 \quad 100 \quad 1)$	$k_1 = k_3 = k_5 = 4, k_2 = k_4 = 0.02, k_6 = 100$

Table 2. Sirius' reactions and maps.

5.1 A synthetic metabolic system

Let us consider the synthetic non-cooperative metabolic system called Sirius and given in Table 2 [9]. Firstly, we generate the dynamics of this model for 1000 steps by using the flux regulation maps of Sirius. Then, we use our algorithm to approximate the vector of fluxes $U^{\circ}[1]$ at the evolution step 1. Starting from $U^{\circ}[1]$, by applying $OLGA[i]$ for $i = 1, 2, \ldots, 900$, we deduce the vectors $U[i]$, for $i = 2, 3, \ldots, 899$, according to the Log-Gain theory. Figure 1 shows the fluxes relative to the dynamics of Sirius initially generated, while the Figure 2 shows the inferred fluxes. These results show an almost complete accordance.

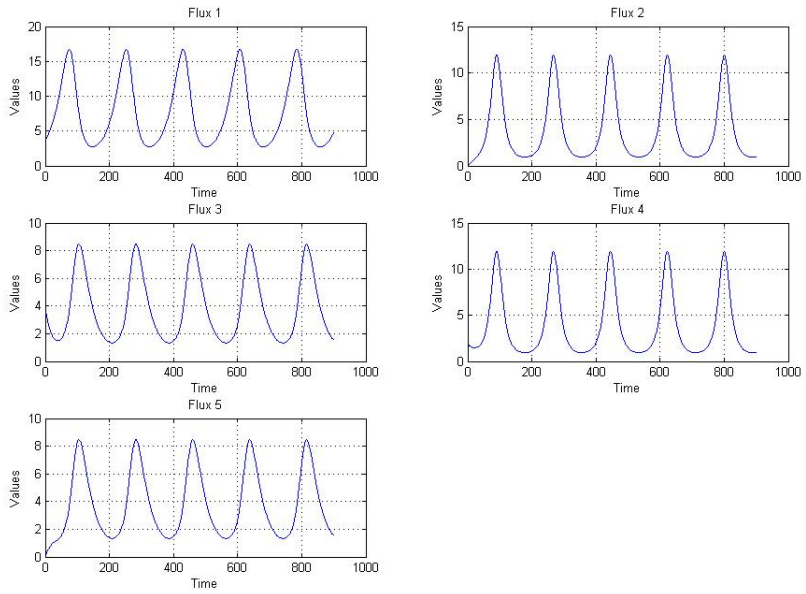


Fig. 1. The values of Sirius' fluxes calculated by using the flux regulation maps given in Table 2.

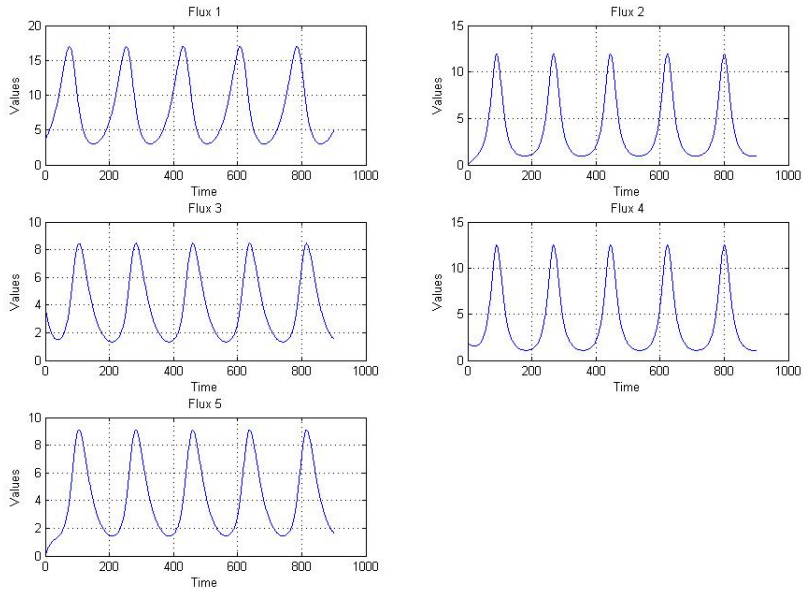


Fig. 2. The values of Sirius' fluxes calculated by applying the Log-gain theory and the initial vector of fluxes inferred by the proposed algorithm.

5.2 A biochemical case study

In this subsection the application of the algorithm to approximate the initial fluxes of the Belousov-Zhabotinsky reaction, also known as BZ reaction, is discussed. This reaction represents a famous example of a biochemical oscillatory phenomenon. Its importance is that it is the first evidence of a biochemical clock. Although the stoichiometry of the BZ reaction is quite complicated, several simplified mathematical models of this phenomenon have been proposed. In particular, Prigogine and Nicolis [15] proposed a simplified formulation of the dynamics of the BZ reaction, called *Brusselator*, whose oscillating behavior is represented by only two substances, x and y respectively, and it is governed by the following system of differential equations:

$$\begin{aligned} \frac{dx}{dt} &= k_1 - k_2 x + k_3 x^2 y - k_4 x \\ \frac{dy}{dt} &= k_2 x - k_3 x^2 y \end{aligned} \tag{14}$$

where $k_1 = 100, k_2 = 3, k_3 = 10^{-4}$ and $k_4 = 1$ represent constant rates. The numerical solution of the system system (14) from initial conditions $x = 1$ and

$y = 10$ shows the oscillatory dynamics displayed in Figure 3. We use this dynamics as experimental data on which applying our algorithm. By reading the set of

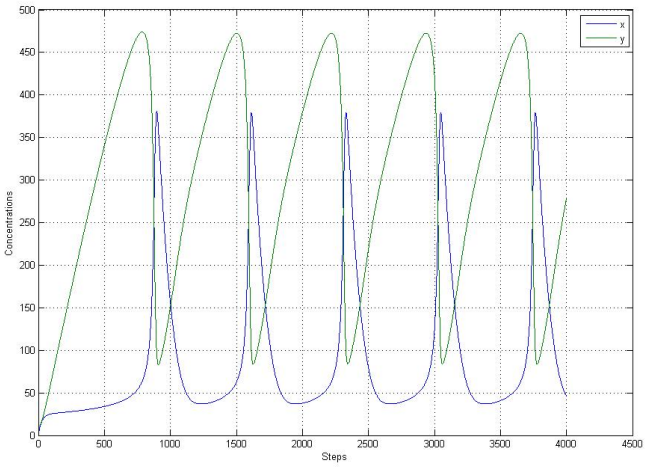


Fig. 3. Numerical solution of the system of differential equations (14).

differential equations (14) the stoichiometry of the Brusselator can be interpreted by using the set of rewriting rules reported in Table 3. In fact, species x has two positive and two negative contributions, while one positive and one negative contributions characterize y. Thus, the equations can be translate in the suitable stoichiometry by following the strategy described in [5].

Rules
$r_1 : \lambda \rightarrow x$
$r_2 : xxy \rightarrow xxx$
$r_3 : x \rightarrow y$
$r_4 : x \rightarrow \lambda$

Table 3. A set of rewriting rules that describes the Brusselator' stoichiometry.

In the case of BZ we adopt a different strategy of validation of our algorithm. In fact, there is a complete correspondence between the dynamics computed by the differential model and that one computed by the equational metabolic algorithm using the fluxes inferred (Figure 4) by solving an OLGA module starting from the initial fluxes inferred by means of our algorithm.

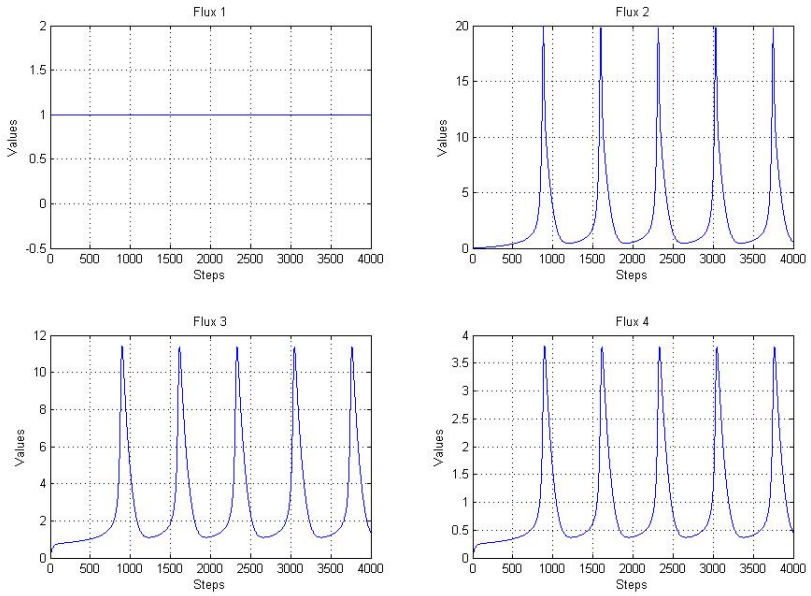


Fig. 4. The BZ reaction's fluxes calculated by using the Log-gain theory and the initial vector of fluxes inferred by our algorithm.

6 Conclusions

In this paper we have devised an algorithm for inferring the initial reaction fluxes of a metabolic network.

The proposed algorithm has been validated on test cases of a synthetic metabolic oscillator and Brusselator reaction. The near future investigations will be planed with the aim *i)* to show the applicability of our method to complex biological cases *ii)* and to improve this algorithm possibly with other relevant computational features.

References

1. P.B. Belousov: Sb. Ref. Radiats. Med. Medgiz. page 145, 1959.
2. L. von Bertalanffy: *General Systems Theory: Foundations, Developments, Applications*. George Braziller Inc, New York, NY, 1967.
3. G. Ciobanu, M.J. Pérez-Jiménez, G. Păun: *Applications of Membrane Computing (Natural Computing Series)*. Springer-Verlag, Berlin, 2006.
4. F. Fontana, L. Bianco, V. Manca: P systems and the modeling of biochemical oscillations. In R. Freud, G. Păun, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, WMC 2005,* LNCS 3850, Springer, 2005, 199–208.

5. F. Fontana, V. Manca: Discrete solution to differential equations by metabolic P systems. *Theoretical Computer Science*, 372 (2007), 165–182.
6. J.S. Huxley: *Problems of Relative Growth*. 2nd, Dover, New York, 1972.
7. D.S. Jones, B.D. Sleeman: *Differential Equations and Mathematical Biology*. Chapman & Hall/CRC, February 2003.
8. V. Manca: Log-Gain Principles for Metabolic P Systems. *Natural Computing*. To appear.
9. V. Manca: The Metabolic Algorithm for P systems: Principles and Applications. *Theoretical Computer Science*, 404 (2008), 142–157.
10. V. Manca: Fundamentals of metabolic P systems. In G. Păun, G. Rozenberg, and A. Salomaa, editors, *Handbook of Membrane Computing*, chapter 16. Oxford University Press, 2009. To appear.
11. V. Manca: Metabolic P dynamics. In G. Păun, G. Rozenberg, and A. Salomaa, editors, *Handbook of Membrane Computing*, chapter 17. Oxford University Press, 2009. To appear.
12. V. Manca, L. Bianco: Biological networks in metabolic P systems. *BioSystems*, 91, 3 (2008), 489–498.
13. V. Manca, L. Bianco, F. Fontana: Evolution and oscillation in P systems: Applications to biological phenomena. In G. Mauri, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, and A. Salomaa, editors, *Membrane Computing, WMC5*, LNCS 3365, Springer, 2005, 63–84.
14. V. Manca, R. Pagliarini, S. Zorzan: A photosynthetic process modelled by a metabolic P system. *Natural Computing*, 2008. DOI 10.1007/s11047-008-9104-x.
15. G. Nicolis, I. Prigogine: *Exploring Complexity. An Introduction*. Freeman ans Company, San Francisco, CA, 1989.
16. G. Păun: Computing with membranes. An introduction. *Bulletin of the EATCS*, 67 (February 1999), 139–152.
17. G. Păun: *Membrane Computing: An Introduction*. Springer, Berlin, 2002.
18. G. Păun, G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287, 1 (2002), 73–100.
19. K.S. Scott: *Chemical Chaos*. Cambridge University Press, Cambridge, UK, 1991.
20. E.O. Voit: *Computational Analysis of Biochemical Systems*. Cambridge University Press, 2000.
21. P. Waage, C.M. Guldberg: Forhandlinger, *Videnskabs-selskabet i Christiana*, 35 (1864).
22. A.M. Zhabotinsky: Proc. Acc. Sci, USRR. 157 (1964), 392.

New Normal Forms for Spiking Neural P Systems

Linqiang Pan[1,2], Gheorghe Păun[2,3]

[1] Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn, lqpan@us.es
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
george.paun@imar.ro, gpaun@us.es

Summary. We consider a natural restriction in the architecture of a spiking neural P system, namely, to have neurons of a small number of types (i.e., using a small number of sets of rules), and we prove that three types of neurons are sufficient in order to generate each recursively enumerable set of numbers as the distance between the first two spikes emitted by the system or as the number of spikes in a specified neuron, in the halting configuration. The case we investigate is that of spiking neural P systems with standard rules, with delays, but without using forgetting rules; similar normal forms remain to be found for other types of systems.

1 Introduction

The spiking neural P systems (in short, SN P systems) were introduced in [3], and then investigated in a large number of papers. We refer to the respective chapter of [4] for general information in this area, and to the membrane computing website from [5] for details.

In this note, the SN P systems are considered as generators of sets of numbers, with the numbers obtained as the distance in time between the first two spikes emitted by the output neuron of the system, or with the generated number given as the number of spikes present in a given neuron in the end of the computation. Systems with standard rules are used (i.e., with only one spike produced by each rule), with the rules used sequentially in each neuron, but the whole system working in the maximally parallel way (i.e., each neuron which can use a rule has to do it).

Several normal forms were imposed to SN P systems – see, e.g., [2]. A natural restriction suggested by biology but also natural by itself is to restrict the number

of types of neurons used in a system, where by "type" we understand the set of rules present in a neuron. An SN P system whose neurons are of at most k types is said to be in the kR-normal form.

We prove that each recursively enumerable set of natural numbers can be generated by an SN P system (without forgetting rules, but using delays) in the $3R$-normal form when the number is obtained as the distance between the first two spikes sent out by the system or when the number is given by the number of spikes from a specified neuron. Slightly bigger values are obtained when we also consider the number of spikes initially present in a neuron for defining the "type" of a neuron. We do not know whether or not these results can be improved, or how they extend to other classes of SN P systems. (Do the forgetting rules help? What about extended rules, about asynchronous SN P systems, systems with exhaustive use of rules, etc?) What about SN P systems used in the accepting mode? (The systems can then be deterministic, which usually brings some simplifications.)

2 Prerequisites

We assume the reader to be familiar with basic elements about SN P systems, e.g., from [4] and [5], and we introduce here only a few notations, as well as the notion of register machines, used later in the proofs of our results. We also assume familiarity with very basic elements of automata and language theory, as available in many monographs.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$. The family of Turing computable sets of natural numbers is denoted by NRE. For a regular expression E we denote by $L(E)$ the regular language identified by E.

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 instruction `HALT`), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M computes (generates) a number n in the following way: we start with all registers empty (i.e., 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. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE.

Without loss of generality, we may assume 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. In the proofs of our results we assume that the register machines which we simulate have these properties.

We can also use a register machine in the accepting mode: a number is stored in the first register (all other registers are empty); if the computation starting in this configuration eventually halts, then the number is accepted. Again, all sets of numbers in NRE can be obtained, even using deterministic register machines, i.e., with the ADD instructions of the form $l_i : (\texttt{ADD}(r), l_j, l_k)$ with $l_j = l_k$ (in this case, the instruction is written in the form $l_i : (\texttt{ADD}(r), l_j)$).

Convention: when evaluating or comparing the power of two number generating/accepting devices, number zero is ignored.

3 Spiking Neural P Systems

In order to have the paper self-contained, we recall here the definition of an SN P system and of the set of numbers generated or accepted by it.

An SN P system of degree $m \geq 1$ is a construct of the form

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out),$$

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 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, $c \geq 1$, and $d \geq 0$;
      (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. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neurons, respectively.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a; d$ can be applied. The application of this rule means removing c spikes (thus only $k - c$ remain in σ_i), the neuron is fired, and it produces a 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, etc. 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* (this corresponds to the refractory period from neurobiology), 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). 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 $t + d + 1$).

The rules of type (2) are *forgetting* rules and they 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.

If a rule $E/a^c \to a; d$ of type (1) has $E = a^c$, then we will write it in the following simplified form: $a^c \to a; d$.

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 $E_2/a^{c_2} \to a; d_2$, 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 in that case, only one of them is chosen non-deterministically. Note however that, by definition, if a firing rule is applicable, then no forgetting rule is applicable, and vice versa.

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. During a computation, the "state" of the system is described by both by the number of spikes present in each neuron, and by the open/closed condition of each neuron: if a neuron is closed, then we have to specify when it will become open again.

Using the rules as described above, one 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*, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0.

An SN P system can be used in various ways. In the generative mode, we start from the initial configuration and we define the result of a computation (i) either as the number of steps between the first two spikes sent out by the output neuron, or (ii) as the number of spikes present in neuron σ_{out} when the computation halts (note that in the first case we do not request that the computation halts after sending out two spikes). We denote by $N_2(\Pi)$ the set of numbers computed in the first way by an SN P system Π and by $N_{gen}(\Pi)$ the set of numbers generated

by Π in the second case. We can also use Π in the accepting mode: a number n is introduced in the system in the form of a number $f(n)$ of spikes placed in neuron σ_{in}, for a well-specified mapping f, and the number n is accepted if and only if the computation halts. (Alternatively, we can introduce the number to be recognized as the distance in time between two spikes entering neuron σ_{in} from the environment.) We denote by $N_{acc}(\Pi)$ the set of numbers accepted by Π.

In the generative case, the neuron (with label) *in* is ignored, in the accepting mode the neuron *out* is ignored (in most cases below, we identify the neuron σ_i with its label i, so we say "neuron i" understanding that we speak about "neuron σ_i"). We can also use an SN P system in the computing mode, introducing a number in neuron *in* and obtaining a result in neuron *out*, but we do not consider this case here.

A neuron σ_i (in the initial configuration of an SN P system) is characterized by n_i, the number of spikes present in it, and by R_i, its associated set of rules. An SN P system is said to be in the kR-normal form, for some $k \geq 1$, if there are at most k different sets $R_1, \ldots, R_k$ of rules used in the m neurons of the system. An SN P system is said to be in the knR-normal form, for some $k \geq 1$, if there are at most k different pairs $(n_1, R_1), \ldots, (n_k, R_k)$ describing the m neurons of the system.

We denote by $N_\alpha SNP(k\beta, forg, dley)$ the families of all sets $N_\alpha(\Pi)$ computed by SN P systems in the $k\beta$-normal form, for $\alpha \in \{2, gen, acc\}$, $\beta \in \{R, nR\}$, and $k \geq 1$; if no forgetting rules are used, then we remove the indication *forg* from the notation; if all rules have delay $d = 0$, then we remove the indication *dley* from the notation.

4 A 3*R*-Normal Form Result

We are going now to prove the main result mentioned in the Introduction: SN P systems with only three different sets of rules are universal when generating numbers encoded in the first two spikes of the spike train.

Theorem 1. $NRE = N_2SNP(3R, dley)$.

Proof. We show that $NRE \subseteq N_2SNP(3R, dley)$; the converse inclusion is straightforward (or we can invoke for it the Turing-Church thesis). Let us consider a register machine $M = (m, H, l_0, l_h, I)$ with the properties specified in Section 2. We construct an SN P system Π which simulates M in the way somewhat standard already when proving that a class of SN P systems is universal. Specifically, we construct modules ADD and SUB to simulate the instructions of M, as well as an output module FIN which provides the result (in the form of a suitable spike train). Each register r of M will have a neuron r in Π, and if the register contains the number n, then the associated neuron will contain $2n$ spikes.

The modules will be given in a graphical form, indicating their initial configuration, the synapses, and, for each neuron, the associated set of rules; all neurons

are initially empty, with the exception of the neuron associated with the initial label, l_0, of M, which contains one spike, and with exception of a few other neurons, as shown in the following figures.

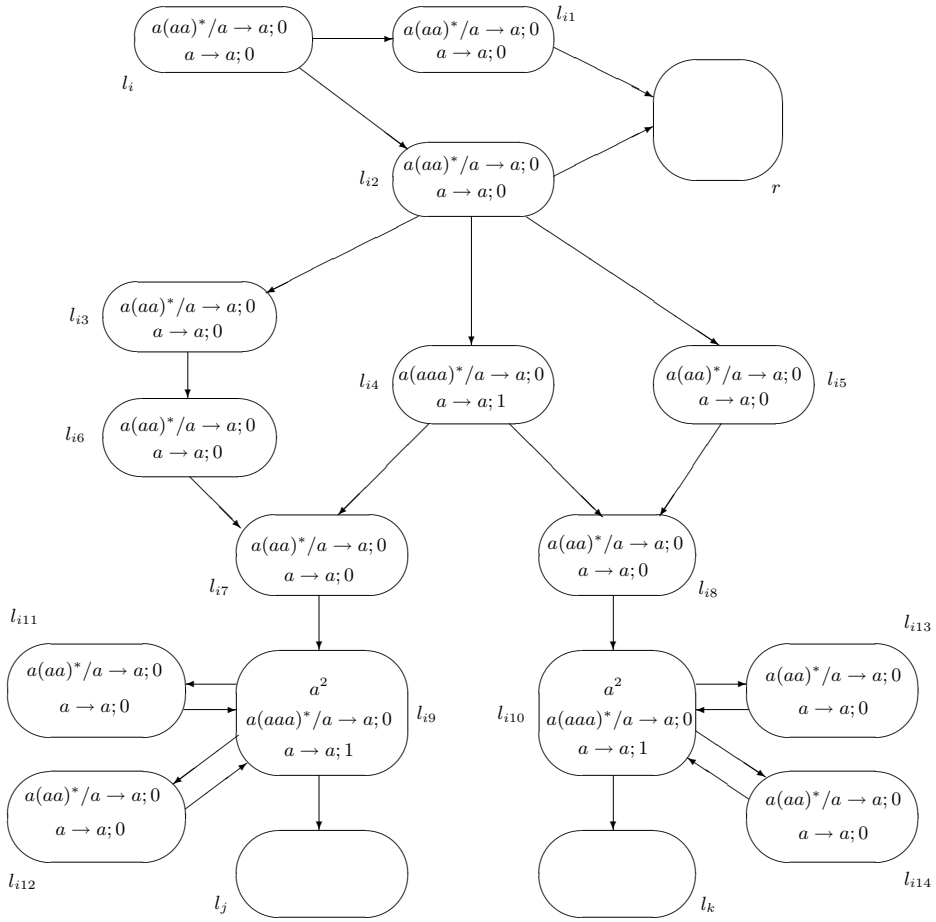


Fig. 1. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

We consider the following three sets of rules:

$$R_1 = \{a(aa)^*/a \to a; 0, \quad a \to a; 0\},$$
$$R_2 = \{a(aa)^*/a^3 \to a; 0, \quad a \to a; 1\},$$
$$R_3 = \{a(aaa)^*/a \to a; 0, \quad a \to a; 1\}.$$

The *ADD* module used to simulate an addition instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$ is indicated in Figure 1. No rule in R_1 can be applied in the presence of an even number of spikes. If a spike enters the neuron (with the label) l_i, then this neuron starts using its rules; initially, this is the case with neuron l_0. Neuron l_i spikes and one spike is sent to both neuron l_{i1} and neuron l_{i2}, which also spike in the next step. In this way, two spikes are sent to neuron r, and this represents the increment of register r by one. Neuron l_{i2} also sends a spike to neurons l_{i3}, l_{i4}, and l_{i5}. Neurons l_{i3} and l_{i5} spike immediately, while neuron l_{i4} can non-deterministically choose either rule to use as both of them are enabled by the existence of a single spike – this ensures the non-deterministic passage to one of the instructions l_j or l_k..

Assume that $\sigma_{l_{i4}}$ uses the rule $a(aa)^*/a \to a; 0$. This means that in the next step $\sigma_{l_{i8}}$ receives two spikes, hence no rule here can be used. Simultaneously, neurons l_{i6} and l_{i7} receive one spike each, and both of them spike. In this way, $\sigma_{l_{i9}}$ receives one spike and $\sigma_{l_{i7}}$ continues having one spike. Neuron l_{i9} contains now a number of spikes of the form $3n + 3$, for some $n \geq 0$ (initially we had two spikes here, hence $n = 0$) and no rule is enabled. In the next step, this neuron receives one further spike, and the first rule is fired (the number of spikes is now $3(n + 1) + 1$). All neurons l_j and l_{i11}, l_{i12} receive one spike. The last two neurons send back to $\sigma_{l_{i9}}$ one spike each, hence the number of spikes in this neuron will be again congruent with 2 modulo 3, as at the beginning. Thus, the neuron associated with the label l_j has been activated.

If neuron l_{i4} uses the rule $a \to a; 1$, then $\sigma_{l_{i7}}$ receives two spikes at the same (after one time unit) time and this branch remains idle, while neurons $l_{i8}, l_{i10}, l_{i13}, l_{i14}$ behave like neurons $l_{i7}, l_{i9}, l_{i11}, l_{i12}$, and eventually σ_{l_k} is activated and the number of spikes from $\sigma_{l_{i10}}$ returns to the form $3s + 2$, for some $s \geq 0$.

The simulation of the ADD instruction is correctly completed.

The SUB module used to simulate a subtraction instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is shown in Figure 2. Because the reader has the experience of examining the work of the ADD module, this time we do not write explicitly the rules, but the sets R_1, R_2, R_3 as defined above. Like in the case of the ADD module, the SUB module starts to work when a spike enters the neuron with the label l_i. The functioning of each neuron is similar to the previous case (the rules to be used are chosen in the same way and eventually the neurons remain with a number of spikes like that in the starting configuration).

The neuron l_i sends a spike to the neurons l_{i1}, and r. If register r is not empty, then the rule $a(aa)^*/a^3 \to a; 0$ of R_2 will be applied.

Assume that this is the case. This means that σ_r spikes immediately, hence $\sigma_{l_{i2}}$ receives two spikes (one from $\sigma_{l_{i1}}$) and is doing nothing, while neuron l_{i3} receives one spike and it fires. A spike is sent to each of the three neurons l_{i7}, l_{i8}, and l_{i9}. This last neuron will send a spike to $\sigma_{l_{i11}}$, which will spike, thus activating the neuron associated with label l_j. The two spikes sent by $\sigma_{l_{i7}}, \sigma_{l_{i8}}$ to $\sigma_{l_{i10}}$ wait here, as no rule is enabled for a number of spikes of the form $3n + 2$. In the next step, a spike comes from neuron l_{i11}, hence $\sigma_{l_{i10}}$ ends with a number of spikes which is a multiple of 3, hence no rule is activated.

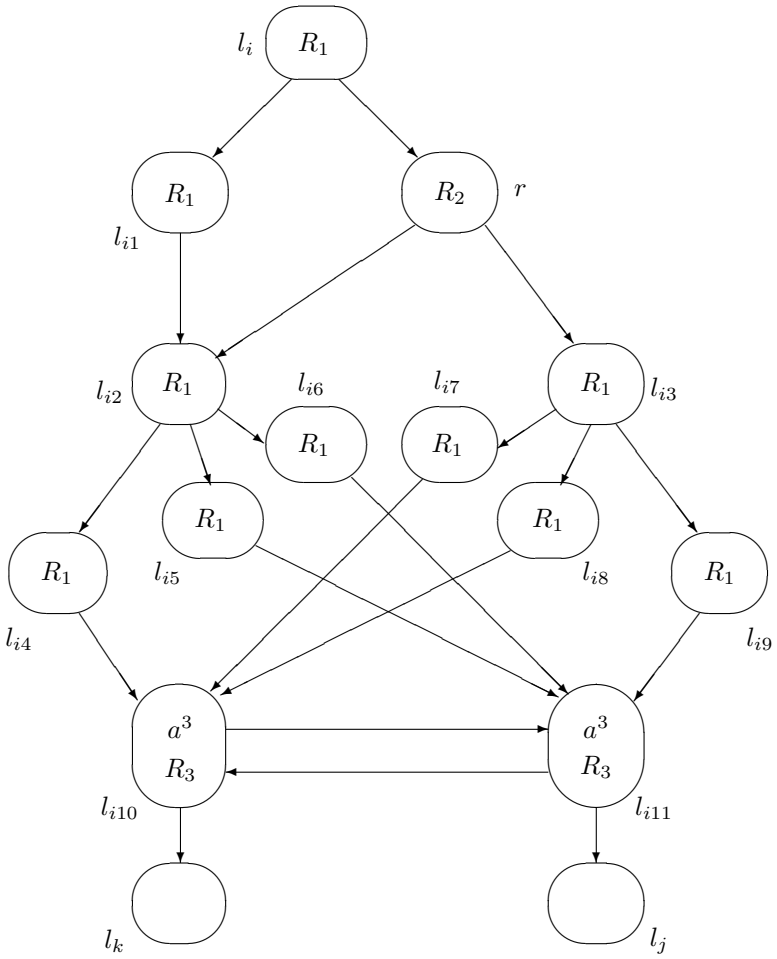


Fig. 2. Module SUB, simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

If the register r is empty, then in σ_r we have to use the rule $a \to a; 1$. The neuron l_{i2} receives a spike from $\sigma_{l_{i1}}$ and in the next step it fires, at the same time with the move of spikes produced at the previous step in σ_r and kept there because of the delay. In this moment, all neurons $l_{i2}, l_{i3}, l_{i4}, l_{i5}$, and l_{i6} contains one spike. Neurons l_{i4}, l_{i5}, l_{i6} send their spikes to $\sigma_{l_{i10}}$ and $\sigma_{l_{i11}}$, but they immediately receive one spike from $\sigma_{l_{i2}}$. This also happens with neurons l_{i7}, l_{i8}, l_{i9}, which receive one spike each from $\sigma_{l_{i3}}$. Neuron l_{i10} spikes and activated l_k, sending at the same time one spike to $\sigma_{l_{i11}}$, thus completing here the number of spikes to a multiples of three. Similarly, in the next step, $\sigma_{l_{i10}}$ (resp., $\sigma_{l_{i11}}$) receives three spikes each, from neurons l_{i4}, l_{i7}, l_{i8} (respectively, l_{i5}, l_{i6}, l_{i9}). The simulation of the SUB instruction

is correctly completed, with the neurons containing numbers of spikes of the same parity as in the beginning.

The modules have different neurons, precisely identified by the label of the respective instruction of M. Modules ADD do not interfere. However, a problem appears with modules SUB: when simulating an instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$, neuron σ_r send one spike to all neurons l_{s2}, l_{s3} from modules associated with instructions $l_s : (\texttt{SUB}(r), l_u, l_v)$ (that is, subtracting from the same register r). However, no undesired effect appears: the spikes arrive simultaneously in neurons l_{s2}, l_{s3}, hence they send one spike to each of the three neurons "below" them, which, in turn, send their spikes to neurons l_{s10}, l_{s11}; each of these neurons gets three spikes, hence no rule can be used here, the spikes are just accumulated (in a number which continues to be a multiple of 3).

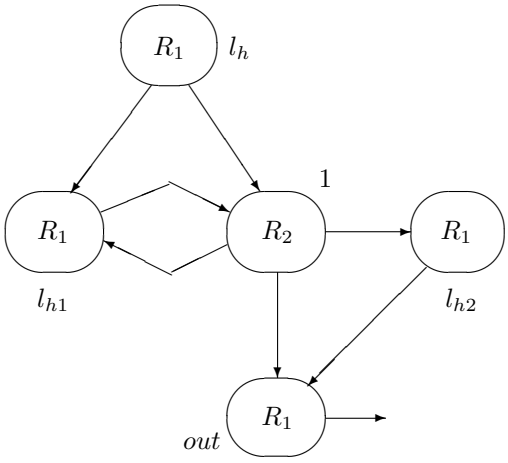


Fig. 3. The FIN module

The addition and subtraction modules simulate the computation of M. In order to produce the number generated by M as the distance between the first two spikes sent out by the system Π we use the module FIN from Figure 3. It is triggered when M reaches the l_h : HALT instruction. At this point a single spike is sent to neuron 1, and at the same time to $\sigma_{l_{h1}}$. Neuron σ_1 sends a spike to each neuron l_{h1}, l_{h2}, and out. The output neuron spikes (for the first time). Neurons σ_1 and $\sigma_{l_{h1}}$ continuously exchange spikes, hence at each step from now on neuron σ_1 contains an odd number of spikes and fires. Neuron out gets two spikes in each step, one from σ_1 and one from $\sigma_{l_{h2}}$, hence nothing happens. When the content of σ_1 is exhausted, the rule $a \to a; 1$ must be used here. The neuron is closed, the spike of $\sigma_{l_{h1}}$ is lost, σ_{out} receives only one spike, from $\sigma_{l_{h2}}$, and spikes for the second time. The work of the system continues forever, because of the interchange of spikes between $\sigma_{l_{h1}}$ and σ_1, but we are interested only in the distance between the first

two spikes emitted by σ_{out}, and this distance is equal to the number stored in register 1 in the end of the computation of M. Consequently, $N(M) = N_2(\Pi)$ and this concludes the proof. □

In the previous figures one can see that the set R_1 appears in neurons having zero or one spike (the case of σ_{l_0}) in the initial configuration, R_2 only with zero spikes, and the set R_3 appears in neurons with zero, two, or three spikes. This means that, if we also consider the number of spikes present in a neuron in the initial configuration when defining the type of a neuron, then the previous $3R$-normal form becomes a $6nR$-normal form.

Corollary 1. $NRE = N_2SNP(6nR, dley)$.

When considering the generated number encoded in the number of spikes present in the output neuron, then several simplifications of the previous constructions are possible. First, the module FIN is no longer necessary; moreover, when a spike is sent to neuron l_h, the computation will halt. Because no instruction is performed in the register machine after reaching the instruction l_h : `HALT`, we provide no outgoing synapse for neuron l_h, so it does matter which rules are present in this neuron, no change is implied on the result of the computation. Then, we only write to register 1, hence to neuron 1 we do not have to apply SUB operations; this means that we can only add spikes to this neuron, namely, one at a time, while any rule can be used inside because no outgoing synapse is present for this neuron. However, always we have the same number of types of neurons, because three types are necessary in modules ADD and SUB. (Similarly, we can use for neuron 1 a FIN module which halves the number of spikes and sends them to another neuron, which leads back to a construction as above.) The results are again as above (the details are left to the reader):

Corollary 2. $NRE = N_{gen}SNP(3R, dley) = N_{gen}SNP(6nR, dley)$.

5 Final Remarks

The accepting case brings further simplifications: the ADD instructions are deterministic (hence only one type of neurons is necessary – see Figure 4, where R_1 is as above), and the FIN module is no longer necessary (we consider the input given as the number of spikes initially present in neuron σ_{in}, without taking into account this number when defining the types of neurons).

However, if we also take into consideration the number of spikes present in the neurons, then we get the following types: $(0, R_1), (1, R_1), (0, R_2), (3, R_3)$, that is, we have the next result:

Corollary 3. $NRE = N_{acc}SNP(3R, dley) = N_{acc}SNP(4nR, dley)$.

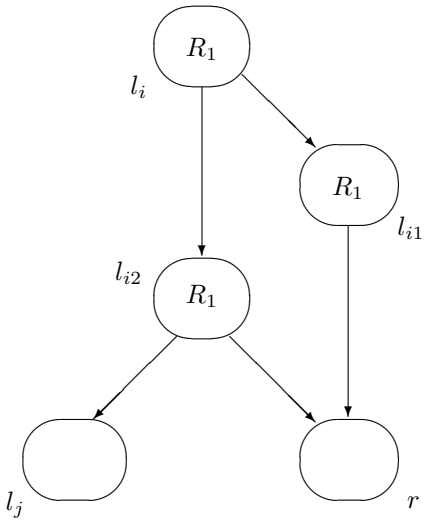


Fig. 4. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j)$

We do not have a construction for module SUB using only two types of neurons.

As mentioned in the Introduction, there are several open problems and research topics suggested by the previous results. We conclude by mentioning a few basic ones. Is the result in Theorem 1 optimal, or a $2R$-normal form or even a $1R$-normal form result is valid? Extend this study to other classes of SN P systems, with other types of rules or with other modes of using the rules.

Important remark: The proof of Theorem 1 implicitly shows that the universality of SN P systems can be obtained without using forgetting rules. The result was first stated in [2], but the construction of the SUB module as given in that paper has a bug: the interaction between neurons from different SUB modules acting on the same register is not examined and the undesired interactions avoided. This is done in the construction from Figure 2, as explicitly mentioned above. Note that our construction uses the delay feature; in [1] it is proved that both the forgetting rules and the delay feature can be avoided without losing the universality, but also there it seems that the interaction of neurons in different SUB modules is not carefully checked. Whether or not the idea in our module SUB can be used to obtain such a stronger normal form remains to be seen.

Acknowledgements

The work of L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60803113), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science

Foundation of Hubei Province (2008CDB113 and 2008CDB180). The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valía, de la Junta de Andalucía, grant P08 – TIC 04200.

References

1. M. García-Arnau, D. Pérez, A. Rodríguez-Patón, P. Sosík: Spiking neural P systems: Stronger normal forms. *Proc. Fifth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo et al., eds.), Fenix Editora, Sevilla, 2007, 157–178.
2. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Paton, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.
3. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308
4. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2010 (in press).
5. The P Systems Website: `http://ppage.psystems.eu`.

Spiking Neural P Systems with Anti-Spikes

Linqiang Pan[1,2], Gheorghe Păun[2,3]

[1] Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn, lqpan@us.es
[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
george.paun@imar.ro, gpaun@us.es

Summary. Besides usual spikes employed in spiking neural P systems, we consider "anti-spikes", which participate in spiking and forgetting rules, but also annihilate spikes when meeting in the same neuron. This simple extension of spiking neural P systems is shown to considerably simplify the universality proofs in this area: all rules become of the form $b^c \to b'$ or $b^c \to \lambda$, where b, b' are spikes or anti-spikes. Therefore, the regular expressions which control the spiking are the simplest possible, identifying only a singleton. A possible variation is not to produce anti-spikes in neurons, but to consider some "inhibitory synapses", which transform the spikes which pass along them into anti-spikes. Also in this case, universality is rather easy to obtain, with rules of the above simple forms.

1 Introduction

The spiking neural P systems (in short, SN P systems) were introduced in [4], and then investigated in a large number of papers. We refer to the respective chapter of [7] for general information in this area, and to the membrane computing website from [9] for details.

In this note, we consider a variation of SN P systems which was suggested several times, i.e., involving inhibitory impulses/spikes or inhibitory synapses and investigated in a few papers under various interpretations/formalizations – see, e.g., [1], [2], [5], [8]. The definition we take here for such spikes – we call them *anti-spikes* (somewhat thinking to anti-matter) – considers having, besides usual "positive" spikes denoted by a, objects denoted by $\bar{a}$, which participate in spiking or forgetting rules as usual spikes, but also in implicit rules of the form $a\bar{a} \to \lambda$: if an anti-spike meets a spike in a given neuron, then they annihilate each other,

and this happens instantaneously (the disappearance of one a and one $\bar{a}$ takes no time, it is like applying the rule $a\bar{a} \to \lambda$ without consuming any time for that).

This simple extension of SN P systems is proved to entail a surprising simplification of both the proofs and the form of rules necessary for simulating Turing machines (actually, the proofs here are based on simulating register machines) by means of SN P systems: all rules have a singleton regular expression, which, moreover, indicates precisely the number of spikes or anti-spikes to consume by the rule. (Precisely, we have rules of the forms $b^c \to b'$ or $b^c \to \lambda$, where b, b' are spikes or anti-spikes; such rules, having the regular expression E such that $L(E) = b^c$ are called *pure*; formal definitions will be given immediately.) This can be considered as a (surprising) normal form for this case; please compare with the normal forms from [3], especially with the simplifications of regular expressions obtained there.

Anti-spikes are produced from usual spikes by means of usual spiking rules; in turn, rules consuming anti-spikes can produce spikes or anti-spikes (actually, as we will see below, the latter case can be avoided). A possible variant is to produce always only spikes and to consider synapses which "change the nature" of spikes. Also in this case, universality is easily proved, using only pure rules.

2 Prerequisites

We assume the reader to be familiar with basic elements about SN P systems, e.g., from [7] and [9], and we introduce here only a few notations, as well as the notion of register machines, used later in the proofs of our results. We also assume familiarity with very basic elements of automata and language theory, as available in many monographs.

For an alphabet V, V^* denotes the set of all finite strings of symbols from V, the empty string is denoted by λ, and the set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we write simply a^* and a^+ instead of $\{a\}^*, \{a\}^+$.

A regular expression over an alphabet V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over V. Non-necessary parentheses can be omitted when writing a regular expression, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

The family of Turing computable sets of natural numbers is denoted by 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 instruction `HALT`), and I is the set of instructions; each label from H labels only one instruction from I, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\texttt{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j, l_k),
- $l_i : (\texttt{SUB}(r), l_j, l_k)$ (if register r is non-empty, then subtract 1 from it and go to the instruction with label l_j, otherwise go to the instruction with label l_k),
- $l_h : \texttt{HALT}$ (the halt instruction).

A register machine M computes (generates) a number n in the following way: we start with all registers empty (i.e., 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. The set of all numbers computed by M is denoted by $N(M)$. It is known that register machines compute all sets of numbers which are Turing computable, hence they characterize NRE.

Without loss of generality, we may assume 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.

We can also use a register machine in the accepting mode: a number is stored in the first register (all other registers are empty); if the computation starting in this configuration eventually halts, then the number is accepted. Again, all sets of numbers in NRE can be obtained, even using deterministic register machines, i.e., with the ADD instructions of the form $l_i : (\texttt{ADD}(r), l_j, l_k)$ with $l_j = l_k$ (in this case, the instruction is written in the form $l_i : (\texttt{ADD}(r), l_j)$).

Again, without loss of generality, we may assume that in the halting configuration all registers are empty.

Convention: when evaluating or comparing the power of two number generating/accepting devices, number zero is ignored.

3 Spiking Neural P Systems with Anti-Spikes

We recall first the definition of an SN P system in the classic form (without delays, because this feature is not used in our paper) and of the set of numbers generated or accepted by it.

An SN P system of degree $m \geq 1$ is a construct

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out),$$

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 in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^c \to a$, where E is a regular expression over a and $c \geq 1$;
      (2) $a^s \to \lambda$, for some $s \geq 1$;
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. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and *output* neurons, respectively.

The rules of type (1) are *firing* (we also say *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains k spikes, and $a^k \in L(E), k \geq c$, then the rule $E/a^c \to a$ can be applied. The application of this rule means removing c spikes (thus only $k - c$ remain in σ_i), the neuron is fired, and it produces a spike which is sent immediately to all neurons σ_j such that $(i, j) \in syn$.

The rules of type (2) are *forgetting* rules and they 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 we have not imposed here the restriction that for each rule $E/a^c \to a$ of type (1) and $a^s \to \lambda$ of type (2) from R_i to have $a^s \notin L(E)$.

If a rule $E/a^c \to a$ of type (1) has $E = a^c$, then we will write it in the simplified form $a^c \to a$ and we say that it is *pure*.

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, $E_1/a^{c_1} \to a$ and $E_2/a^{c_2} \to a$, 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 in that case only one of them is chosen non-deterministically. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

The configuration of the system is described by the number of spikes present in each neuron. The initial configuration is $n_1, n_2, \ldots, n_m$. Using the rules as described above, one 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 no rule can be used. With any computation (halting or not) we associate a *spike train*, the sequence of zeros and ones describing the behavior of the output neuron: if the output neuron spikes, then we write 1, otherwise we write 0.

When using an SN P system in the generative mode, we start from the initial configuration and we define the result of a computation as the number of steps between the first two spikes sent out by the output neuron. We denote by $N_2(\Pi)$ the set of numbers computed by Π in this way. In the accepting mode, a number n is introduced in the system in the form of a number $f(n)$ of spikes placed in neuron σ_{in}, for a well-specified mapping f, and the number n is accepted if and only if the computation halts. We denote by $N_{acc}(\Pi)$ the set of numbers accepted by Π. It is also possible to introduce the number n by means of a spike train entering neuron σ_{in}, as the distance between the first two spikes coming to σ_{in}.

In the generative case, the neuron (with label) in is ignored, in the accepting mode the neuron *out* is ignored (sometimes below, we identify the neuron σ_i with

its label i, so we say "neuron i" understanding that we speak about "neuron σ_i"). We can also use an SN P system in the computing mode, introducing a number in neuron in and obtaining a result in (by means of) neuron out, but we do not consider this case here.

We denote by $N_\alpha SNP(rule_k)$ the families of all sets $N_\alpha(\Pi)$, $\alpha \in \{2, acc\}$, computed by SN P systems with at most $k \geq 1$ rules (spiking or forgetting) in each neuron.

Let us now pass to the extension mentioned in the Introduction. A further object, $\bar{a}$, is added to the alphabet O, and the spiking and forgetting rules are of the forms

$$E/b^c \to b',\ b^c \to \lambda,$$

where E is a regular expression over a or over $\bar{a}$, while $b, b' \in \{a, \bar{a}\}$, and $c \geq 1$. As above, if $L(E) = b^c$, then we write the first rule as $b^c \to b'$ and we say that it is pure.

Note that we have four categories of rules, identified by $(b, b') \in \{(a, a), (a, \bar{a}), (\bar{a}, a), (\bar{a}, \bar{a})\}$.

The rules are used as in a usual SN P system, with the additional fact that a and $\bar{a}$ "cannot stay together", they instantaneously annihilate each other: if in a neuron there are either objects a or objects $\bar{a}$, and further objects of either type (maybe both) arrive from other neurons, such that we end with a^r and $\bar{a}^s$ inside, then immediately a rule of the form $a\bar{a} \to \lambda$ is applied in a maximal manner, so that either a^{r-s} or $\bar{a}^{s-r}$ remain, provided that $r \geq s$ or $s \geq r$, respectively.

We stress the fact that the mutual annihilation of spikes and anti-spikes takes no time, so that the neuron always contains either only spikes or anti-spikes. That is why, for instance, the regular expressions of the spiking rules are defined either on a or on $\bar{a}$, but not on both symbols. Of course, we can also imagine that the annihilation takes one time unit, when the explicit rule $a\bar{a} \to \lambda$ is used, but we do not consider this case here (if the rule $a\bar{a} \to \lambda$ has priority over other rules, then no essential change occurs in the proofs below).

The computations and the result of computations are defined in the same way as for usual SN P systems – but we consider the restriction that the output neuron produces only spikes, not also anti-spikes (again, this is a restriction which is only natural/elegant, but not essential). As above, we denote by $N_\alpha S_a NP(rule_k, forg)$ the families of all sets $N_\alpha(\Pi)$, $\alpha \in \{2, acc\}$, computed by SN P systems with at most $k \geq 1$ rules (spiking or forgetting) in each neuron, using also anti-spikes. When only pure rules are used, we write $N_\alpha S_a NP(prule_k)$.

4 Universality Results

We start by considering the generative case, for which we have the next result (universality is known for usual SN P systems, without anti-spikes, but now both the proof is simpler and the used rules are all pure):

Theorem 1. $NRE = N_2 S_a NP(prule_2)$.

Proof. We only have to prove the inclusion $NRE \subseteq N_2S_aNP(prule_2, forg)$.

Let us consider a register machine $M = (m, H, l_0, l_h, I)$ as introduced in Section 2. We construct an SN P system Π (with $O = \{a, \bar{a}\}$) which simulates M in the way already standard in the literature when proving that a class of SN P systems is universal. Specifically, we construct modules ADD and SUB to simulate the instructions of M, as well as an output module FIN which provides the result (in the form of a suitable spike train). Each register r of M will have a neuron σ_r in Π, and if the register contains the number n, then the associated neuron will contain n spikes, except for the neuron σ_1 associated with the first register (the neurons associated with registers will either contain occurrences of a, hence $\bar{a}$ disappears immediately, or only $\bar{a}$ is present, and it is consumed in the next step by a rule $\bar{a} \to a$). Two spikes are initially placed in the neuron σ_1 associated with the first register, so if the first register contains the number n, then neuron σ_1 will contain $n + 2$ spikes. These two spikes are used for outputting the computation result. Note that the number of spikes in the neuron σ_1 will not be smaller than two before the simulation reaches the instruction l_h and the output module FIN is activated, because we assume that the output register is never decremented during the computation. One neuron σ_{l_i} is associated with each label $l_i \in H$, and some auxiliary neurons $\sigma_{l_i^{(j)}}, j = 1, 2, 3, \ldots$, will be also considered, thus precisely identified by label l_i (remember that each $l_i \in H$ is associated with a unique instruction of M).

The modules will be given in a graphical form, indicating the synapses and, for each neuron, the associated set of rules. In the initial configuration, all neurons are empty, except for the neurons associated with label l_0 of M and the first register, which contain one spike and two spikes, respectively. In general, when a spike a is sent to a neuron σ_{l_i}, with $l_i \in H$, then that neuron becomes active and the module associated with the respective instruction of M starts to work, simulating the instruction.

The functioning of the module from Figure 1, simulating an instruction $l_i : (\texttt{ADD}(r), l_j, l_k)$, is obvious; the non-deterministic choice between instructions l_j and l_k is done by non-deterministically choosing the rule to apply in neuron $\sigma_{l_i^{(3)}}$.

The simulation of an instruction $l_i : (\texttt{SUB}(r), l_j, l_k)$ is also simple – see the module from Figure 2. The neuron σ_{l_i} sends a spike to neurons $\sigma_{l_i^{(1)}}$ and $\sigma_{l_i^{(2)}}$. In the next step, neuron $\sigma_{l_i^{(2)}}$ sends an anti-spike to neuron σ_r, corresponding to register r; at the same time, $\sigma_{l_i^{(1)}}$ sends a spike to each neuron $\sigma_{l_i^{(3)}}, \sigma_{l_i^{(4)}}$. If register r is non-empty, that is, neuron σ_r contains at least one a, then $\bar{a}$ removes one occurrence of a, which corresponds to subtracting one from register r, and no rule is applied in σ_r. This means $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive only two spikes, from $\sigma_{l_i^{(3)}}$ and $\sigma_{l_i^{(4)}}$, hence σ_{l_j} is activated and σ_{l_k} not. If register r is empty, then the rule $\bar{a} \to a$ is used in σ_r, hence $\sigma_{l_i^{(5)}}$ and $\sigma_{l_i^{(6)}}$ receive three spikes, and this leads to the activation of σ_{l_k}, which is the correct continuation also in this case.

Note that if there are several sub instructions l_t which act on register r, then σ_r will send one spike to neurons $\sigma_{l_t^{(5)}}$ and $\sigma_{l_t^{(6)}}$ while simulating the instruction

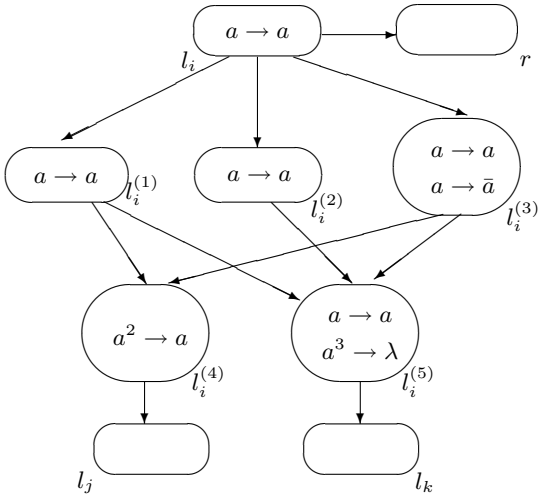


Fig. 1. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

$l_i : (\texttt{SUB}(r), l_j, l_k)$, but this spike is immediately removed by the rule $a \to \lambda$ present in all neurons $\sigma_{l_t^{(5)}}, \sigma_{l_t^{(6)}}$.

The module FIN, which produces a spike train such that the distance between the first two spikes equals the number stored in register 1 of M, is indicated in Figure 3. At some step t, the neuron σ_{l_h} is activated, which means that the register machine M reaches the halt instruction and the system Π starts to output the result. Suppose the number stored in register 1 of M is n. At step $t+2$, neurons σ_{h_1}, σ_{h_3} and σ_{h_4} contain a spike. Neurons σ_{h_1} and σ_{h_4} exchange spikes among them, and thus σ_{h_4} sends a spike to neuron σ_{h_5} continuously until neuron σ_1 spikes and neurons σ_{h_1}, σ_{h_4}, σ_{h_5} are "flooded". At step $t+4$, neuron σ_{out} receives a spike, and in the next step σ_{out} sends a spike to the environment; at the same time, σ_1 receives an anti-spike that decreases by one the number of spikes from σ_1. At step $t+n+4$, the neuron σ_1 contains one spikes, and in the next step neuron σ_1 sends a spike to neuron σ_{out}. At step $t+n+6$, neuron σ_{out} spikes again. The distance between the first two spikes emitted by σ_{out} equals n, which is exactly the number stored in register 1 of M. The spike produced by neuron σ_1 "floods" neurons σ_{h_1}, σ_{h_4}, and σ_{h_5}, thus blocking the work of these neurons. After the system sends the second spike out, the whole system halts.

From the previous explanations we get the equality $N(M) = N_2(\Pi)$ and this concludes the proof. □

Note that in the previous construction there is no rule of the form $\bar{a}^c \to \bar{a}$; is it possible to also avoid other types of rules? For instance, the rule $\bar{a} \to a$ only appears in the neurons associated with registers in module SUB. Is it possible to remove the $\bar{a} \to a$ by replacing it with the rules $a^c \to a$ and $a \to \bar{a}$?

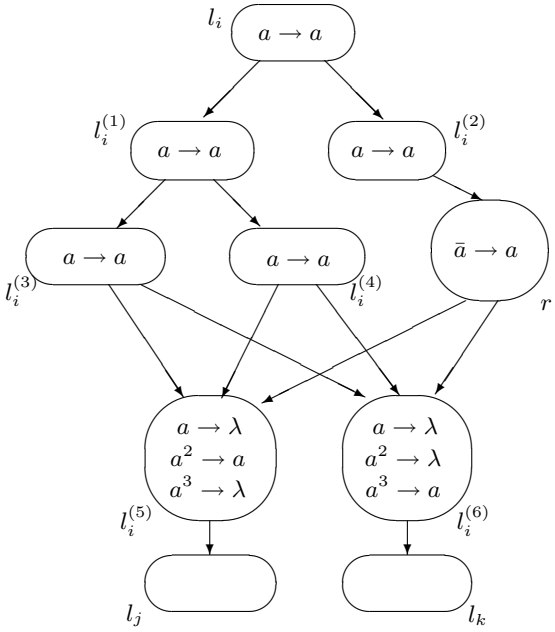


Fig. 2. Module SUB, simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

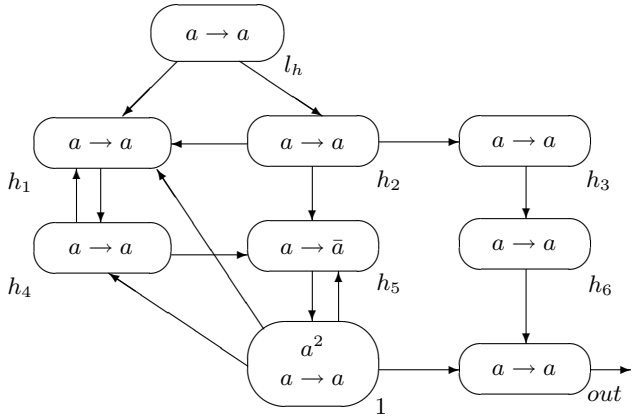


Fig. 3. The FIN module

If the SN P systems are used in the accepting mode, then a further simplification is entailed by the fact that the ADD instructions are deterministic. Such an instruction $l_i : (\texttt{ADD}(r), l_j)$ can be directly simulated by a simple module as in Figure 4.

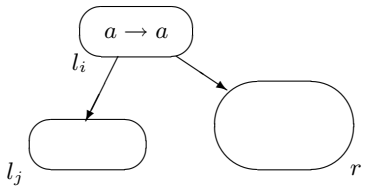


Fig. 4. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j)$

Together with SUB modules, this suffices in the case when the number to accept is introduced as the number of spikes initially present in neuron σ_1. If this number is introduced in the system as the distance between the first two spikes which enters the input neuron, then a input module is necessary, as used, for instance, in [3]. Note that the module INPUT from [3] uses only pure rules (involving only spikes, not also anti-spikes), hence we get a theorem like Theorem 1 also for the accepting case, for both ways of providing the input number.

It is worth mentioning that in the previous constructions we do not have spiking rules which can be used at the same time with forgetting rules.

5 Using Inhibitory Synapses

Let us now consider the case when no rule can produce an anti-spike, but there are synapses which transform spikes into anti-spikes. The previous modules ADD, SUB, FIN can be modified in such a way to obtain a characterization of NRE also in this case. We directly provide these modules, without any explanation about their functioning, in Figures 5, 6, and 7; the synapses which change a into $\bar{a}$ are marked with a dot.

Note that this time the non-determinism in the ADD instruction is simulated by allowing the non-deterministic choice among the spiking rule $\bar{a} \to a$ and the forgetting rule $\bar{a} \to \lambda$ of neuron $\sigma_{l_i^{(1)}}$, which is not allowed in the classic definition of SN P systems. Removing this feature, without introducing rules which are not pure or other ingredients, such as the delay, remains as an open problem.

Denoting by $N_\alpha S_a NP_s(prule_k)$ the respective families of sets of numbers (the subscript s in P_s indicates the use of inhibitory synapses, in the sense specified above), we conclude having the next result:

Theorem 2. $NRE = N_2 S_a NP_s(prule_2)$.

6 Final Remarks

There are several open problems and research topics suggested by the previous results. Some of them were already mentioned, but further questions can be formulated. For instance, can the proofs be improved so that less types of rules are

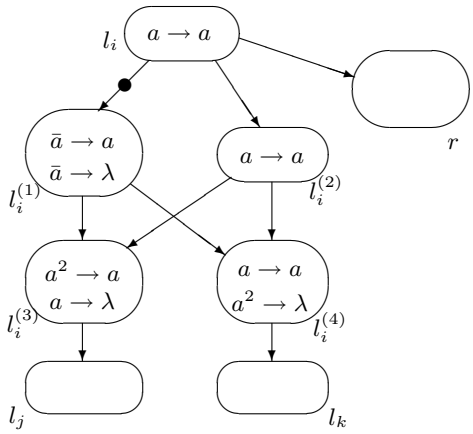


Fig. 5. Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

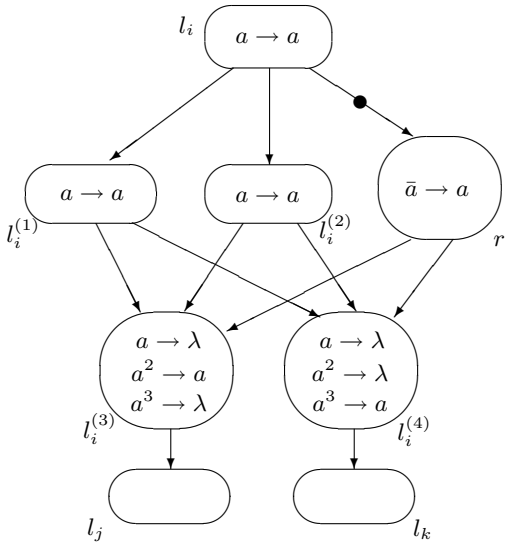


Fig. 6. Module SUB, simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

necessary? We have avoided using rules $\bar{a}^c \to \bar{a}$, but not the other three types, corresponding to the pairs $(a, a), (a, \bar{a}), (\bar{a}, a)$. Then, following the idea from [6], can we decrease the number of *types* of neurons, in the sense of having a small number of sets of rules which are used in each neuron (three such sets are found in [6] to be sufficient for universality in the case of usual SN P systems; do the anti-spikes helps also in this respect?).

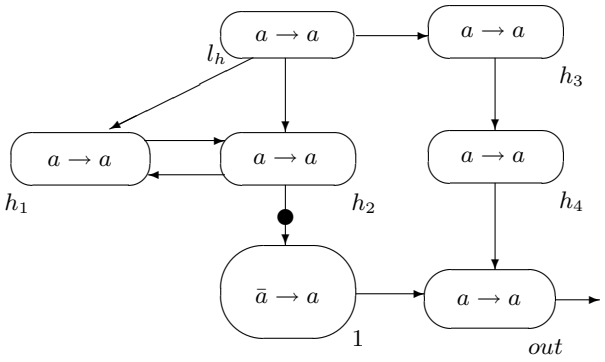


Fig. 7. Module FIN

Acknowledgements

The work of L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60803113), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). The work of Gh. Păun was supported by Proyecto de Excelencia con Investigador de Reconocida Valia, de la Junta de Andalucia, grant P08 – TIC 04200. Useful remarks by Artiom Alhazov are gratefully acknowledged.

References

1. A. Binder, R. Freund, M. Oswald, L. Vock: Extended spiking neural P systems with excitatory and inhibitory astrocytes. Submitted, 2007.
2. R. Freund, M. Oswald: Spiking neural P systems with inhibitory axons. *AROB Conf.*, Japan, 2007.
3. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.
4. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
5. J.M. Mingo: Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. In the present volume.
6. L. Pan, Gh. Păun: New normal forms for spiking neural P systems. In the present volume.
7. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing*. Oxford University Press, 2009.

8. J. Wang, L. Pan: Excitatory and inhibitory spiking neural P systems. Submitted, 2007.
9. The P Systems Website: http://ppage.psystems.eu.

Spiking Neural P Systems with Neuron Division and Budding

Linqiang Pan[1,2], Gheorghe Păun[2,3], Mario J. Pérez-Jiménez[2]

[1] Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
lqpan@mail.hust.edu.cn, lqpan@us.es

[2] Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

[3] Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
george.paun@imar.ro, gpaun@us.es

Summary. In order to enhance the efficiency of spiking neural P systems, we introduce the features of neuron division and neuron budding, which are processes inspired by neural stem cell division. As expected (as it is the case for P systems with active membranes), in this way we get the possibility to solve computationally hard problems in polynomial time. We illustrate this possibility with SAT problem.

1 Introduction

Spiking neural P systems (in short, SN P systems) were introduced in [6] in the framework of membrane computing [13] as a new class of computing devices which are inspired by the neurophysiological behavior of neurons sending electrical impulses (spikes) along axons to other neurons. Since then, many computational properties of SN P systems have been studied; for example, it has been proved that they are Turing-complete when considered as number computing devices [6], when used as language generators [3, 1] and also when computing functions [12].

Investigations related to the possibility to solve computationally hard problems by using SN P systems were first proposed in [2]. The idea was to encode the instances of decision problems in a number of spikes which are placed in an arbitrarily large pre-computed system at the beginning of the computation. It was shown that the resulting SN P systems are able to solve the **NP**-complete problem SAT (the satisfiability of propositional formulas expressed in conjunctive normal form) in a constant time. Slightly different solutions to SAT and 3-SAT by using SN

P systems with pre-computed resources were considered in [7]; here the encoding of an instance of the given problem is introduced into the pre-computed resources in a polynomial number of steps, while the truth values are assigned to the Boolean variables of the formula and the satisfiability of the clauses is checked. The answer associated with the instance of the problem is thus computed in a polynomial time. Finally, very simple semi-uniform and uniform solutions to the numerical **NP**-complete problem `Subset Sum` – by using SN P systems with exponential size pre-computed resources – have been presented in [8]. All the systems constructed above work in a deterministic way.

A different idea of constructing SN P systems for solving **NP**-complete problems was given in [10, 11], where the `Subset Sum` and `SAT` problems were considered. In these papers, the solutions are obtained in a semi-uniform or uniform way by using non-deterministic devices but without pre-computed resources. However, several ingredients are also added to SN P systems such as extended rules, the possibility to have a choice between spiking rules and forgetting rules, etc. An alternative to the constructions of [10, 11] was given in [9], where only standard SN P systems without delaying rules and having a uniform construction are used. However, it should be noted that the systems described in [9] either have an exponential size, or their computations last an exponential number of steps. Indeed, it has been proved in [11] that a deterministic SN P system of a polynomial size cannot solve an **NP**-complete problem in a polynomial time unless **P=NP**. Hence, under the assumption that $\mathbf{P} \neq \mathbf{NP}$, efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance the efficiency of the system.

In this paper, neuron division and budding are introduced into the framework of SN P systems in order to enhance the efficiency of these systems. We exemplify this possibility with a uniform solution to `SAT` problem.

The biological motivation of introducing neuron division and budding into SN P systems comes from the recent discoveries in neurobiology related to neural stem cells – see, e.g., [4]. Neural stem cells persist throughout life within central nervous system in the adult mammalian brain, and this ensures a life-long contribution of new neurons to self-renewing nervous system with about 30000 new neurons being produced every day. Even in vitro, neural stem cells can be grown and extensively expanded for months. New neurons are produced by symmetric or asymmetric division. Two main neuron cell types are found: neuroblasts and astrocytes. The latter form a meshwork and are organized into channels. These observations are incorporated in SN P systems by considering neuron division and budding, and by providing a "synapse dictionary" according to which new synapses are generated, respectively.

The paper is organized as follows. In Section 2 we recall some mathematical preliminaries that will be used in the following. In Section 3 the formal definition of SN P systems with neuron division rules and neuron budding rules is given. In Section 4 we present a uniform family of SN P systems with neuron division and budding rules such that the systems can solve `SAT` problem in a polynomial time.

Section 5 concludes the paper and suggests some possible open problems for future work.

2 Prerequisites

We assume the reader to be familiar with basic elements about membrane computing, e.g., from [13] and [14], and formal language theory, as available in many monographs. We mention here only a few notions and notations which are used through the paper.

For an alphabet V, V^* denotes the set of all finite strings over V, with the empty string denoted by λ. The set of all nonempty strings over V is denoted by V^+. When $V = \{a\}$ is a singleton, then we simply write a^* and a^+ instead of $\{a\}^*$, $\{a\}^+$.

A regular expression over an alphabet V is defined as follows: (i) λ and each $a \in V$ is a regular expression, (ii) if E_1, E_2 are regular expressions over V, then $(E_1)(E_2)$, $(E_1) \cup (E_2)$, and $(E_1)^+$ are regular expressions over V, and (iii) nothing else is a regular expression over V. With each regular expression E we associate a language $L(E)$, defined in the following way: (i) $L(\lambda) = \{\lambda\}$ and $L(a) = \{a\}$, for all $a \in V$, (ii) $L((E_1) \cup (E_2)) = L(E_1) \cup L(E_2)$, $L((E_1)(E_2)) = L(E_1)L(E_2)$, and $L((E_1)^+) = (L(E_1))^+$, for all regular expressions E_1, E_2 over V. Non-necessary parentheses can be omitted when writing a regular expression, and also $(E)^+ \cup \{\lambda\}$ can be written as E^*.

3 SN P Systems with Neuron Division and Budding

As stated in the Introduction, SN P systems have been introduced in [6], in the framework of membrane computing. They can be considered as an evolution of P systems, corresponding to a shift from *cell-like* to *neural-like* architectures.

In SN P systems the cells (also called *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*. Every cell may also contain a number of *firing* and *forgetting* rules. Firing rules 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 applicability of each rule is determined by checking the contents of the neuron against a regular set associated with the rule. In each time unit, if a neuron can use one of its rules, then one of such rules must be used. If two or more rules could be applied, then only one of them is non-deterministically chosen. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other. Note that, as usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, and hence the functioning of the system is synchronized. When a cell sends out spikes it becomes "closed" (inactive) for a specified period

of time, a fact which 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, then there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

The structure of SN P systems (that is, the synapse graph) introduced in [6] is static. For both biological and mathematical motivations discussed in the Introduction, neuron division and budding are introduced into SN P systems. In this way, an exponential workspace can be generated in polynomial (even linear) time and computationally hard problems can be efficiently solved by means of a space-time tradeoff.

Formally, a *spiking neural P system with neuron division and budding* of (initial) degree $m \geq 1$ is a construct of the form

$$\Pi = (O, H, syn, n_1, \ldots, n_m, R, in, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. H is a finite set of *labels* for neurons;
3. $syn \subseteq H \times H$ is a *synapse dictionary*, with $(i, i) \notin syn$ for $i \in H$;
4. $n_i \geq 0$ is the *initial number of spikes* contained in neuron i, $i \in \{1, 2, \ldots, m\}$;
5. R is a finite set of *developmental rules*, of the following forms:
   (1) *extended firing* (also called *spiking*) rule $[E/a^c \to a^p; d]_i$, where $i \in H$, E is a regular expression over a, and $c \geq 1$, $p \geq 0$, $d \geq 0$, with the restriction $c \geq p$;
   (2) *neuron division rule* $[E]_i \to [\,]_j \parallel [\,]_k$, where E is a regular expression and $i, j, k \in H$;
   (3) *neuron budding rule* $[E]_i \to [\,]_i / [\,]_j$, where E is a regular expression and $i, j \in H$;
6. $in, out \in H$ indicate the *input* and the *output* neurons of Π.

Note that we have presented here an SN P system in a way slightly different from the usual definition present in the literature, where the neurons present initially in the system are explicitly listed as $\sigma_i = (n_i, R_i)$, where $1 \leq i \leq m$ and R_i are the rules associated with neuron with label i. In what follows we will refer to neuron with label $i \in H$ also denoting it with σ_i.

It is worth to mention that by applying division rules different neurons can appear with the same label. In this context, $(i, j) \in syn$ means the following: there exist synapses from each neuron with label i to each neuron with label j.

If an extended firing rule $[E/a^c \to a^p; d]_i$ has $E = a^c$, then we will write it in the simplified form $[a^c \to a^p; d]_i$; similarly, if a rule $[E/a^c \to a^p; d]_i$ has $d = 0$, then we can simply write it as $[E/a^c \to a^p]_i$; hence, if a rule $[E/a^c \to a^p; d]_i$ has

$E = a^c$ and $d = 0$, then we can write $[a^c \to a^p]_i$. A rule $[E/a^c \to a^p]_i$ with $p = 0$ is written in the form $[E/a^c \to \lambda]_i$ and is called *extended forgetting* rule. Rules of the types $[E/a^c \to a; d]_i$ and $[a^c \to \lambda]_i$ are said to be *standard*.

If a neuron σ_i contains k spikes and $a^k \in L(E), k \geq c$, then the rule $[E/a^c \to a^p; d]_i$ is enabled and it can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in neuron σ_i); 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, etc. 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 (this corresponds to the refractory period from neurobiology), 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). In the step $t + d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t + d + 1$, when the neuron can again apply rules). Once emitted from neuron σ_i, the p spikes reach immediately all neurons σ_j such that there is a synapse going from σ_i to σ_j and which are open, that is, the p spikes are replicated and each target neuron receives p spikes; as stated above, spikes sent to a closed neuron are "lost", that is, they are removed from the system. In the case of the output neuron, p 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. If the rule is a forgetting one of the form $[E/a^c \to \lambda]_i$, then, when it is applied, $c \geq 1$ spikes are removed. When a neuron is closed, none of its rules can be used until it becomes open again.

If (1) a neuron σ_i contains s spikes and $a^s \in L(E)$, and (2) there is no neuron σ_g such that the synapse (g, i) or (i, g) exists in the system, for some $g \in \{j, k\}$, then the division rule $[E]_i \to [\,]_j \parallel [\,]_k$ is enabled and it can be applied. This means that consuming all these s spikes the neuron σ_i is divided into two neurons, σ_j and σ_k. The new neurons contain no spike in the moment when they are created. They can have different labels, but they inherit the synapses that the father neuron already has (if there is a synapse from neuron σ_g to the neuron σ_i, then in the process of division one synapse from neuron σ_g to new neuron σ_j and another one from σ_g to σ_k are established; similarly, if there is a synapse from the neuron σ_i to neuron σ_h, then one synapse from σ_j to σ_h and another one from σ_k to σ_h are established). Note that the restriction provided by condition (2) to the use of the rule ensures that no synapse (j, j) or (k, k) appears. Except for the inheritance of synapses, the new neurons produced by division can have new synapses as provided by the synapse dictionary. Note that during the computation, it is possible that a synapse between neurons involved in the division rule and neurons existing in the system will appear that is not in the synapse dictionary *syn*, because of the inheritance of synapses. Therefore, the synapse dictionary *syn* has two functions: one is to deduce the initial topological structure of the SN P system (a directed graph), for example, if there are neurons $\sigma_1, \ldots, \sigma_k$ at the beginning of computation, then the initial topological structure of the system is $syn \cap (\{1, 2, \ldots, k\} \times \{1, 2, \ldots, k\})$; another function is to guide the synapse establishment associated with the new

neurons generated by neuron division or neuron budding. That is why we call *syn* a synapses dictionary.

If (1) a neuron σ_i contains s spikes, and $a^s \in L(E)$, and (2) there is no neuron σ_j such that the synapse (i, j) exists in the system,then the budding rule $[E]_i \rightarrow [\ ]_i/[\ ]_j$ is enabled and it can be applied. This means that consuming all the s spikes a new neuron is created, σ_j. Both neurons are empty after applying the rule. The neuron σ_i inherits the synapses going to it before using the rule. The neuron σ_j created by budding by neuron σ_i inherits the synapses going out of σ_i before budding, that is, if there is a synapse from neuron σ_i to some neuron σ_h, then a synapse from neuron σ_j to neuron σ_h is established (condition (2) ensures the fact that no synapse (j, j) appears). There is also a synapse (i, j) between neurons σ_i and σ_j. Except for the above synapses associated with neurons σ_i and σ_j, other synapses associated with neuron σ_j can be established according to the synapses dictionary *syn* as in the case of neuron division rule.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R *must* be used. If several rules are enabled in neuron σ_i, irrespective of their types (spiking, dividing, or budding) then only one of them is chosen non-deterministically. When a spiking rule is used, the state of neuron σ_i (open or closed) depends on the delay d. When a neuron division rule or neuron budding rule is applied, at this step the associated neuron is closed, it cannot receive spikes. In the next step, the neurons obtained by division or budding will be open and can receive spikes. Thus, the rules are used in the sequential manner in each neuron, but neurons function in parallel with each other.

It is worth noting here that the two neurons produced by a division rule can have labels different from each other and from the divided neuron, and that they are placed "in parallel", while in the budding case the old neuron (consumes all its spikes and) produces one new neuron which is placed "serially", after the neuron which budded.

The *configuration* of the system is described by the topology structure of the system, the number of spikes associated with each neuron, and the *state* of each neuron (open or closed). Using the rules as described above, one 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.

Traditionally, the input of an SN P system used in the accepting mode is provided in the form of a spike train, a sequence of steps when one spike or no spike enters the input neuron. In what follows we need several spikes at a time to come into the system via the input neuron, that is we consider "generalized spike trains", written in the form $a^{i_1} \cdot a^{i_2} \cdot \ldots \cdot a^{i_r}$, where $r \geq 1, i_j \geq 0$ for each $1 \leq j \leq r$. The meaning is that i_j spikes are introduced in neuron σ_{in} in step j (all these i_j spikes are provided at the same time). Note that we can have $i_j = 0$, which means that no spike is introduced in the input neuron. The period which separate the "packages" a^{i_j} of spikes is necessary in order to make clear that we do not have

here a concatenation of the strings describing these "packages", but a sequence of blocks (more formally, a sequence of multisets over the singleton alphabet O).

Spiking neural P systems can be used to solve decision problems, both in a *semi–uniform* and in a *uniform* way. When solving a problem $\mathcal{Q}$ in the *semi–uniform* setting, for each specified instance $\mathcal{I}$ of $\mathcal{Q}$ an SN P system $\Pi_{\mathcal{Q},\mathcal{I}}$ is (i) built by a Turing machine in a polynomial time (with respect to the size of $\mathcal{I}$), (ii) its structure and initial configuration depend upon $\mathcal{I}$, and (iii) it halts (or emits a specified number of spikes in a given interval of time) if and only if $\mathcal{I}$ is a positive instance of $\mathcal{Q}$. On the other hand, a *uniform* solution of $\mathcal{Q}$ consists of a family $\{\Pi_{\mathcal{Q}}(n)\}_{n\in\mathbb{N}}$ of SN P systems that are built by a Turing machine in a polynomial time (with respect to the size n). When having an instance $\mathcal{I} \in \mathcal{Q}$ of size n, we introduce a polynomial (in n) number of spikes in a designated input neuron of $\Pi_{\mathcal{Q}}(n)$ and the computation halts (or, alternatively, a specified number of spikes is emitted in a given interval of time) if and only if $\mathcal{I}$ is a positive instance. The preference for uniform solutions over semi–uniform ones is given by the fact that they are more strictly related to the structure of the problem, rather than to specific instances. Indeed, in the semi–uniform setting we do not even need any input neuron, as the instance of the problem can be embedded into the initial configuration of the system from the very beginning.

4 A Uniform Solution to SAT Problem

Let us consider the **NP**-complete decision problem SAT [5]. The instances of SAT depend upon two parameters: the number n of variables, and the number m of clauses. We recall that a *clause* is a disjunction of literals, occurrences of x_i or $\neg x_i$, built on a given set $X = \{x_1, x_2, \ldots, x_n\}$ of Boolean variables. Without loss of generality, we can avoid the clauses in which the same literal is repeated or both the literals x_i and $\neg x_i$, for any $1 \leq i \leq n$, occur. In this way, a clause can be seen as a set of at most n literals. An *assignment* of the variables $x_1, x_2, \ldots, x_n$ is a mapping $T : X \to \{0, 1\}$ 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 (*true*) as a result.

We can now formally state the SAT problem as follows.

Problem 1. NAME: SAT.

- INSTANCE: a set $C = \{C_1, C_2, \ldots, C_m\}$ of 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?

Equivalently, we can say that an instance of SAT is a propositional formula $\gamma_{n,m} = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, expressed in the conjunctive normal form as a conjunction

of m clauses, where each clause is a disjunction of literals built using the Boolean variables $x_1, x_2, \ldots, x_n$. With a little abuse of notation, from now on we will denote by $SAT(n,m)$ the set of instances of SAT which have n variables and m clauses.

Let us consider the polynomial time computable function $\langle n,m\rangle = ((m+n)(m+n+1)/2)+m$ (the pair function), which is a primitive recursive and bijective function from $\mathbb{N}^2$ to $\mathbb{N}$. Let us build a uniform family $\{\Pi_{SAT}(\langle n,m\rangle)\}_{n,m\in\mathbb{N}}$ of SN P systems such that for all $n,m\in\mathbb{N}$ the system $\Pi_{SAT}(\langle n,m\rangle)$ solves all the instances of $SAT(n,m)$ in a number of steps which is linear in both n and m. All the systems $\Pi_{SAT}(\langle n,m\rangle)$ will work in a deterministic way.

Because the construction is uniform, we need a way to encode any given instance $\gamma_{n,m}$ of $SAT(n,m)$. As stated above, each clause C_i of $\gamma_{n,m}$ can be seen as a disjunction of at most n literals, and thus for each $j\in\{1,2,\ldots,n\}$ either x_j occurs in C_i, or $\neg x_j$ occurs, or none of them occurs. In order to distinguish these three situations we define the *spike variables* $\alpha_{i,j}$, for $1\leq i\leq m$ and $1\leq j\leq n$, as variables whose values are amounts of spikes; we assign to them the following values:

$$\alpha_{i,j}=\begin{cases} a, & \text{if } x_j \text{ occurs in } C_i;\\ a^2, & \text{if } \neg x_j \text{ occurs in } C_i;\\ a^0, & \text{otherwise.}\end{cases}$$

In this way, clause C_i will be represented by the sequence $\alpha_{i,1}\cdot\alpha_{i,2}\cdot\ldots\cdot\alpha_{i,n}$ of spike variables; in order to represent the entire formula $\gamma_{n,m}$ we just concatenate the representations of the single clauses, thus obtaining the generalized spike train $\alpha_{1,1}\cdot\alpha_{1,2}\cdot\ldots\cdot\alpha_{1,n}.\alpha_{2,1}\cdot\alpha_{2,2}\cdot\ldots\cdot\alpha_{2,n}\cdot\ldots\cdot\alpha_{m,1}\cdot\alpha_{m,1}\cdot\ldots\cdot\alpha_{m,n}$. As an example, the representation of $\gamma_{3,2}=(x_1\vee\neg x_2)(x_1\vee x_3)$ is the sequence $a\cdot a^2\cdot a^0\cdot a\cdot a^0\cdot a$. In order to let the systems have enough time to generate necessary workspace before computing the instances of $SAT(n,m)$, a spiking train $(a^0\cdot)^{2n}$ is added in front of the formula encoding spike train $\alpha_{1,1}\cdot\alpha_{1,2}\cdot\ldots\cdot\alpha_{1,n}.\alpha_{2,1}\cdot\alpha_{2,2}\cdot\ldots\cdot\alpha_{2,n}\cdot\ldots\cdot\alpha_{m,1}\cdot\alpha_{m,1}\cdot\ldots\cdot\alpha_{m,n}$. In general, for any given instance $\gamma_{n,m}$ of $SAT(n,m)$, the encoding sequence is $cod(\gamma_{n,m})=(a^0\cdot)^{2n}\alpha_{1,1}\cdot\alpha_{1,2}\cdot\ldots\cdot\alpha_{1,n}\cdot\alpha_{2,1}\cdot\alpha_{2,2}\cdot\ldots\cdot\alpha_{2,n}\cdot\ldots\cdot\alpha_{m,1}\cdot\alpha_{m,1}\cdot\ldots\cdot\alpha_{m,n}$.

For each $n,m\in\mathbb{N}$, we construct

$$\Pi(\langle n,m\rangle)=(O,H,syn,n_1,\ldots,n_q,R,in,out),$$

with the following components:

The initial degree of the system is $q=4n+7$;

$O=\{a\}$;

$$\begin{aligned}H=\ &\{in,out,cl\}\cup\{d_i\mid i=0,1,\ldots,n\}\\ &\cup\{Cx_i\mid i=1,2,\ldots,n\}\cup\{Cx_i0\mid i=1,2,\ldots,n\}\\ &\cup\{Cx_i1\mid i=1,2,\ldots,n\}\cup\{t_i\mid i=1,2,\ldots,n\}\\ &\cup\{f_i\mid i=1,2,\ldots,n\}\cup\{0,1,2,3\};\end{aligned}$$

$syn = \{(d_i, d_{i+1}) \mid i = 0, 1, \ldots, n-1\} \cup \{(d_n, d_1)\}$
$\cup \{(in, Cx_i) \mid i = 1, 2, \ldots, n\} \cup \{(d_i, Cx_i) \mid i = 1, 2, \ldots, n\}$
$\cup \{(Cx_i, Cx_i 0) \mid i = 1, 2, \ldots, n\} \cup \{(Cx_i, Cx_i 1) \mid i = 1, 2, \ldots, n\}$
$\cup \{(i+1, i) \mid i = 0, 1, 2\} \cup \{(1, 2), (0, out)\}$
$\cup \{(Cx_i 1, t_i) \mid i = 1, 2, \ldots, n\} \cup \{(Cx_i 0, f_i) \mid i = 1, 2, \ldots, n\}$;
$n_{d_0} = n_0 = n_2 = n_3 = 1$, $n_{d_1} = 6$, and there is no spike in the other neurons;

R is the following set of rules:

(1) **spiking rules:**
$[a \to a]_{in}$,
$[a^2 \to a^2]_{in}$,
$[a \to a; 2n + nm]_{d_0}$,
$[a^4 \to a^4]_i$, $i = d_1, \ldots, d_n$,
$[a^5 \to \lambda]_{d_1}$,
$[a^6 \to a^4; 2n + 1]_{d_1}$,
$[a \to \lambda]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a^2 \to \lambda]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a^4 \to \lambda]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a^5 \to a^5; n - i]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a^6 \to a^6; n - i]_{Cx_i}$, $i = 1, 2, \ldots, n$,
$[a^5 \to a^4]_{Cx_i 1}$, $i = 1, 2, \ldots, n$,
$[a^6 \to \lambda]_{Cx_i 1}$, $i = 1, 2, \ldots, n$,
$[a^5 \to \lambda]_{Cx_i 0}$, $i = 1, 2, \ldots, n$,
$[a^6 \to a^4]_{Cx_i 0}$, $i = 1, 2, \ldots, n$,
$[(a^4)^+ \to a]_{t_i}$, $i = 1, 2, \ldots, n$,
$[(a^4)^+ \to a]_{f_i}$, $i = 1, 2, \ldots, n$,
$[a^{4k-1} \to \lambda]_{t_i}$, $k = 1, 2, \ldots, n$, $i = 1, 2, \ldots, n$,
$[a^{4k-1} \to \lambda]_{f_i}$, $k = 1, 2, \ldots, n$, $i = 1, 2, \ldots, n$,
$[a^m \to a^2]_{cl}$,
$[(a^2)^+ / a \to a]_{out}$,
$[a \to a]_i$, $i = 1, 2$,
$[a^2 \to \lambda]_2$,
$[a \to a; 2n - 1]_3$;

(2) **neuron division rules:**
$[a]_0 \to [\,]_{t_1} \parallel [\,]_{f_1}$,
$[a]_{t_i} \to [\,]_{t_{i+1}} \parallel [\,]_{f_{i+1}}$, $i = 1, 2, \ldots, n-1$,
$[a]_{f_i} \to [\,]_{t_{i+1}} \parallel [\,]_{f_{i+1}}$, $i = 1, 2, \ldots, n-1$;

(3) **neuron budding rules:**
$[a]_{t_n} \to [\,]_{t_n} / [\,]_{cl}$,
$[a]_{f_n} \to [\,]_{f_n} / [\,]_{cl}$.

The solution of the SAT problem is obtained by means of a brute force algorithm in the framework of SN P systems with neuron division and budding. Our strategy consists in the following phases:

- *Generation Stage*: The neuron division and budding are applied to generate an exponential number of neurons such that each possible assignment of variables $x_1, x_2, \ldots, x_n$ is represented by a neuron (with associated connections with other neurons by synapses).
- *Input Stage*: The system reads the encoding of the given instance of SAT.
- *Satisfiability Checking Stage*: The system checks whether or not there exists an assignment of variables $x_1, x_2, \ldots, x_n$ that satisfies all the clauses in the propositional formula C.
- *Output Stage*: According to the result of the previous stage, the system sends a spike to the environment if the answer is positive; otherwise, the system does not send any spike to the environment.

Let us have an overview of the computation. The initial structure of the system is shown in Figure 1 (in the figures which follow we only present the spiking and the forgetting rules, but not also the division and budding rules). The first three layers of the system constitutes the input module. The neuron σ_0 and its offsprings will be used to generate an exponential workspace by neuron division and budding rules. The auxiliary neurons σ_1, σ_2, and σ_3 supply necessary spikes to the neuron σ_0 and its offsprings for neuron division and budding rules. The neuron σ_{out} is used to output the result.

Generation Stage: By the way of the encoding of instances, it is easy to see that the spike variables $\alpha_{i,j}$ will be introduced into neuron σ_{in} from step $2n+1$ (it takes $2n$ steps to read $(a^0\cdot)^{2n}$ of $cod(\gamma_{n,m})$). In the first $2n$ steps, the system generates an exponential workspace; after that, the system checks the satisfiability, and outputs the result.

The neuron σ_0 contains one spike, the rule $[a]_0 \to [\,]_{t_1} \parallel [\,]_{f_1}$ is applied, and two neurons σ_{t_1} and σ_{f_1} are generated. They have the associated synapses $(1, f_1)$, $(1, t_1)$, (t_1, out), (f_1, out), $(Cx_1 1, t_1)$ and $(Cx_1 0, f_1)$, where the first 4 synapses are obtained by the heritage of the synapses $(0, out)$ and $(1, 0)$, respectively, and the last 2 synapses are established by the synapse dictionary. The auxiliary neuron σ_2 sends one spike to neuron σ_1, and at step 2 neuron σ_1 sends this spike to neurons σ_{t_1} and σ_{f_1} for the next division. At step 1, the neuron σ_3 contains one spike, and the rule $[a \to a; 2n-1]_3$ is applied. It will send one spike to neuron σ_2 at step $2n$ because of the delay $2n-1$. (As we will see, at step $2n$, neuron σ_1 also sends one spike to neuron σ_2, so neuron σ_2 will have 2 spikes, and the rule $[a^2 \to \lambda]_2$ will be applied. In this way, after step $2n$, the auxiliary neurons stop the work of supplying spikes for division and budding.) The structure of the system after step 1 is shown in Figure 2.

At step 2, neuron σ_1 sends one spike to neurons σ_2, σ_{t_1}, and σ_{f_1}. In the next step, neuron σ_2 sends one spike back to neuron σ_1; in this way, the auxiliary neurons σ_1, σ_2, and σ_3 supply spikes for division and budding every two steps in

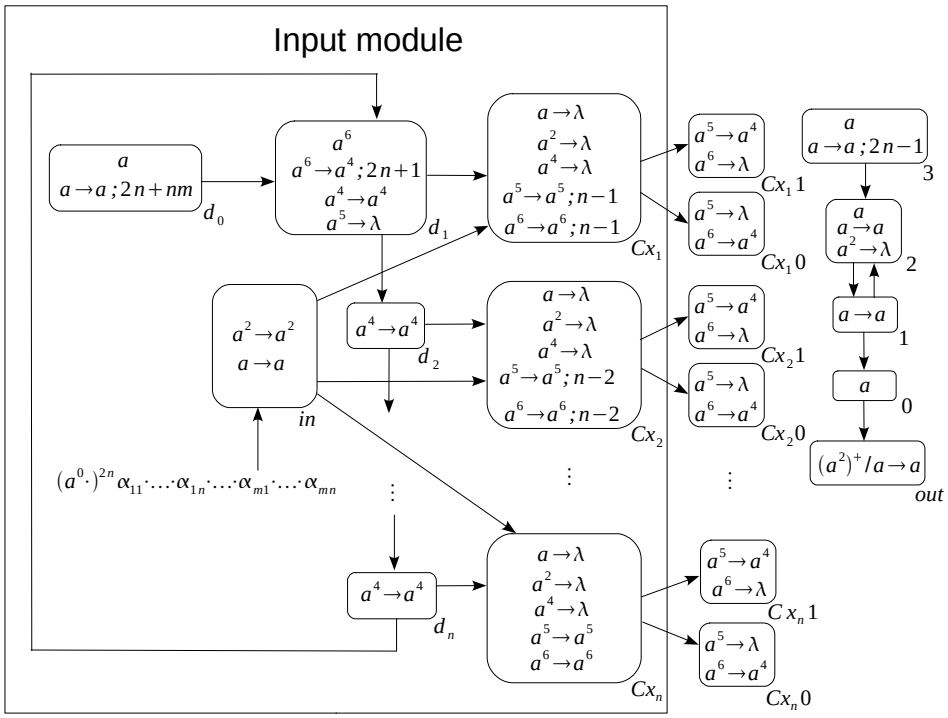


Fig. 1. The initial structure of system $\Pi(\langle n, m\rangle)$

the first $2n$ steps. At step 3, only division rules can be applied in neurons σ_{t_1} and σ_{f_1}, these two neurons are divided, and the associated synapses are obtained by heritage or synapse dictionary. The four neurons with labels t_2 or f_2 correspond to assignments where (1) $x_1 = 1$ and $x_2 = 1$, (2) $x_1 = 1$ and $x_2 = 0$, (3) $x_1 = 0$ and $x_2 = 1$, (4) $x_1 = 0$ and $x_2 = 0$, respectively. The neuron $Cx_1 1$ (encoding that x_1 appears in a clause) has synapses from it to neurons whose corresponding assignments have $x_1 = 1$. That is, assignments with $x_1 = 1$ satisfy clauses where x_1 appears. The structure of the system after step 3 is shown in Figure 3. The neuron division is iterated until 2^n neurons with labels t_n or f_n appear at step $2n - 1$. The corresponding structure after step $2n - 1$ is shown in Figure 4.

At step $2n$, each neuron with label t_n or f_n obtains one spike from neuron σ_1, then in next step the budding rules $[a]_{t_n} \to [\,]_{t_n} / [\,]_{cl}$ and $[a]_{f_n} \to [\,]_{f_n} / [\,]_{cl}$ are applied. Each created neuron σ_{cl} has synapses (t_n, cl) or (f_n, cl) and (cl, out) by heritage. At step $2n$, neuron σ_1 also sends one spike to neuron σ_2, at the time, neuron σ_3 sends one spike to neuron σ_2. So neuron σ_2 has two spikes, and the rule $[a^2 \to \lambda]_2$ is applied at step $2n+1$. After that, the auxiliary neurons cannot supply spikes any more, and the system passes to read the encoding of given instance. The structure of the system after step $2n + 1$ is shown in Figure 5.

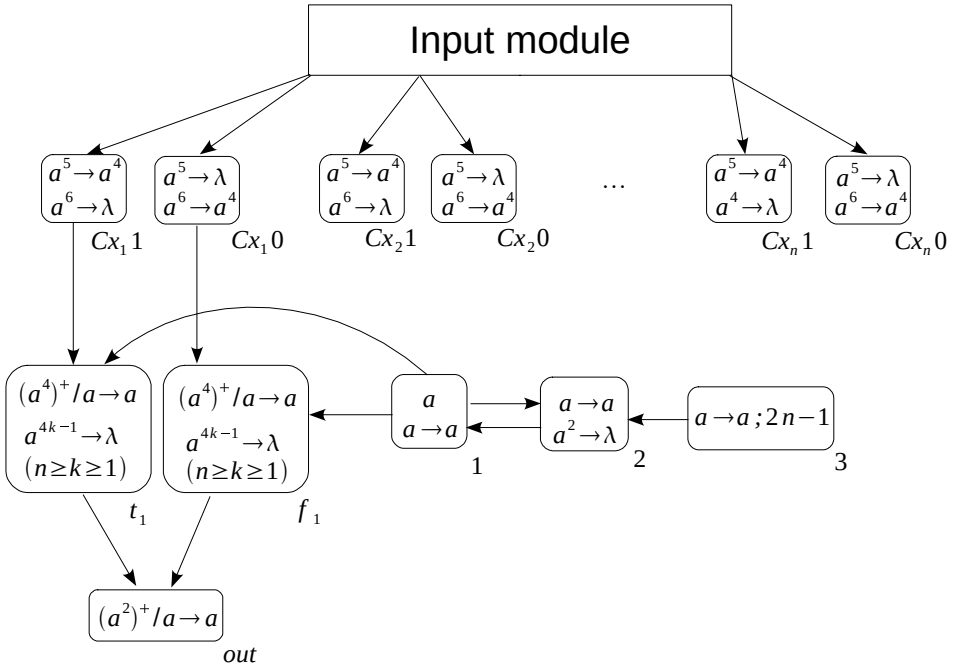


Fig. 2. The structure of system $\Pi(\langle n, m \rangle)$ after step 1

Input Stage: The input module consists of $2n + 2$ neurons, which are in the layers $1 - 3$ as illustrated in Figure 1; σ_{in} is the unique input neuron. The spikes of the encoding sequence $code(\gamma_{n,m})$ are introduced into σ_{in} one "package" by one "package", starting from step 1. It takes $2n$ steps to introduce $(a^0 \cdot)^{2n}$ into neuron σ_{in}. At step $2n + 1$, the value of the first spike variable α_{11}, which is the virtual symbol that represents the occurrence of the first variable in the first clause, enters into neuron σ_{in}. In the next step, the value of the spike variable α_{11} is replicated and sent to neurons σ_{Cx_i}, for all $i \in \{1, 2, \ldots, n\}$; in the meanwhile, neuron σ_{d_1} send four auxiliary spikes to neurons σ_{Cx_1} and σ_{d_2} (the rule $[a^6 \to a^4; 2n+1]_{d_1}$ is applied at step 1). Hence, neuron σ_{Cx_1} will contain 4, 5 or 6 spikes: if x_1 occurs in C_1, then neuron σ_{Cx_1} collects 5 spikes; if $\neg x_1$ occurs in C_1, then it collects 6 spikes; if neither x_1 nor $\neg x_1$ occur in C_1, then it collects 4 spikes. Moreover, if neuron σ_{Cx_1} has received 5 or 6 spikes, then it will be closed for $n - 1$ steps, according to the delay associated with the rules in it; on the other hand, if 4 spikes are received, then they are deleted and the neuron remains open. At step $2n + 3$, the value of the second spike variable α_{12} from neuron σ_{in} is distributed to neurons σ_{Cx_i}, $2 \leq i \leq n$, where the spikes corresponding to α_{11} are deleted by the rules $[a \to \lambda]_{Cx_i}$ and $[a^2 \to \lambda]_{Cx_i}$, $2 \leq i \leq n$. At the same time, the four auxiliary spikes are duplicated and one copy of them enters into neurons σ_{Cx_2} and σ_{d_3}, respectively. The neuron σ_{Cx_2} will be closed for $n - 2$ steps only if it contains

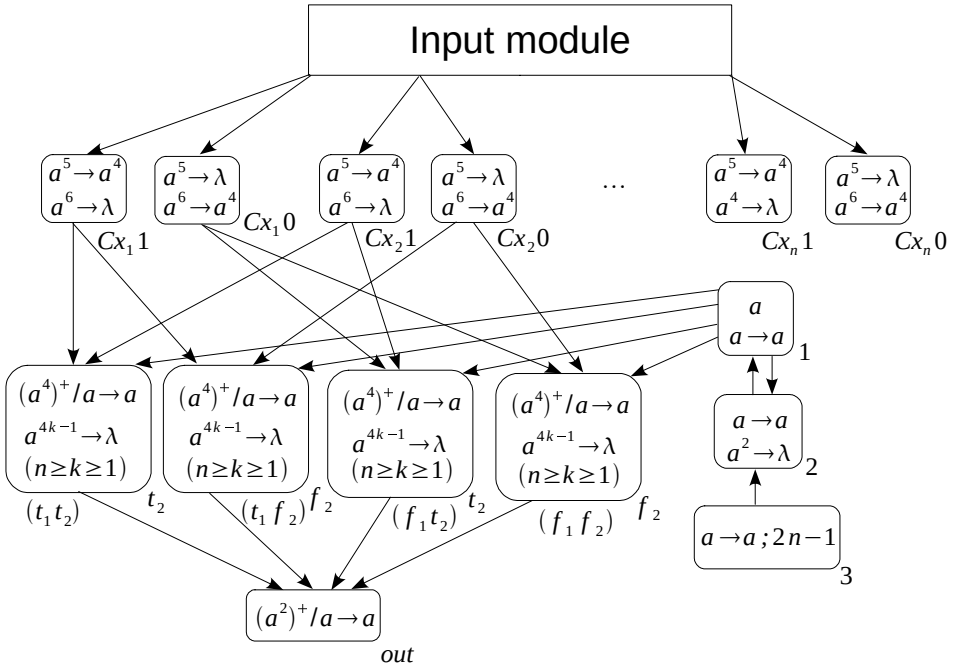


Fig. 3. The structure of system $\Pi(\langle n, m\rangle)$ after step 3

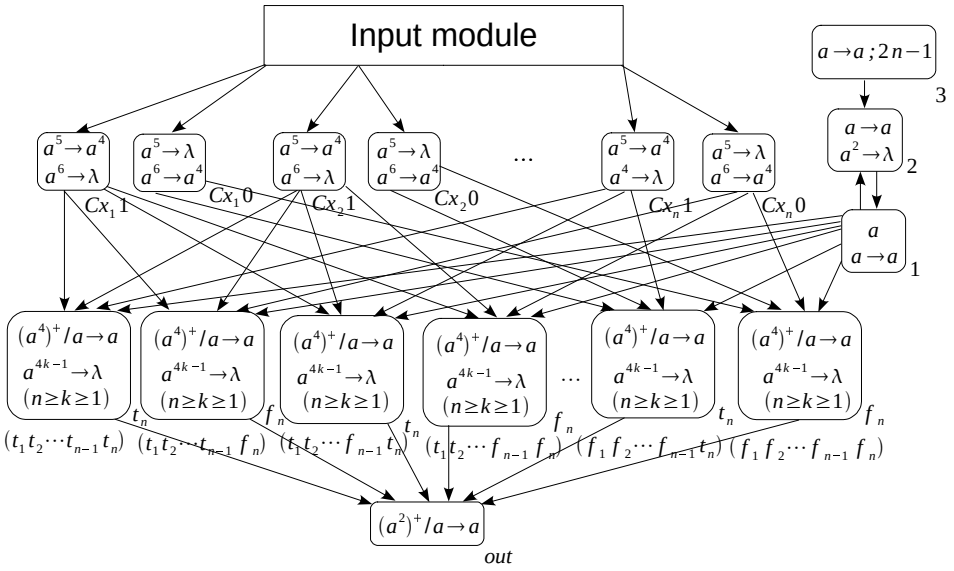


Fig. 4. The structure of system $\Pi(\langle n, m\rangle)$ after step $2n-1$

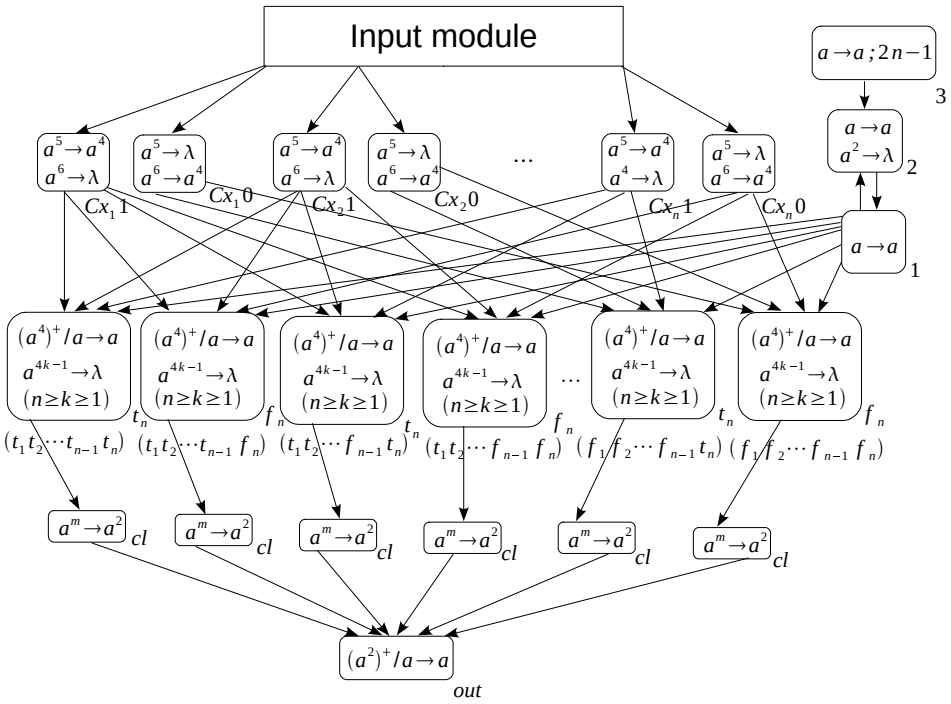


Fig. 5. The structure of system $\Pi(\langle n, m \rangle)$ after step $2n + 1$

5 or 6 spikes, which means that this neuron will not receive any spike during this period. In neurons σ_{Cx_i}, $3 \leq i \leq n$, the spikes represented by α_{12} are forgotten in the next step.

In this way, the values of the spike variables are introduced and delayed in the corresponding neurons until the value of the spike variable α_{1n} of the first clause and the four auxiliary spikes enter together into neuron σ_{Cx_n} at step $3n+1$. At that moment, the representation of the first clause of $\gamma_{n,m}$ has been entirely introduced in the system, and the second clause starts to enter into the input module. In general, it takes $mn + 1$ steps to introduce the whole sequence $code(\gamma_{n,m})$ in the system, and the input process is completed at step $2n + nm + 1$.

At step $2n + nm + 1$, the neuron σ_{d_n} sends four spike to neuron σ_{d_1}. At the same time, the auxiliary neuron σ_{d_0} also sends a spike to the neuron σ_{d_1} (the rule $[a \to a; 2n + nm]_{d_0}$ is used at the first step of the computation). So neuron σ_{d_1} contains 5 spikes, and in the next step these 5 spikes are forgotten by the rule $[a^5 \to \lambda]_{d_1}$. This ensures that the system eventually halts.

Satisfiability Checking Stage: Once all the values of spike variables α_{1i} $(1 \leq i \leq n)$, representing the first clause, have appeared in their corresponding neurons σ_{Cx_i} in layer 3, together with a copy of the four auxiliary spikes, all the spikes contained in σ_{Cx_i} are duplicated and sent simultaneously to the pair of neurons

σ_{Cx_i1} and σ_{Cx_i0}, for $i \in \{1, 2, \ldots, n\}$, at step $3n + 2$. In this way, each neuron σ_{Cx_i1} and σ_{Cx_i0} receives 5 or 6 spikes when x_i or $\neg x_i$ occurs in C_1, respectively, whereas it receives no spikes when neither x_i nor $\neg x_i$ occurs in C_1. In general, if neuron σ_{Cx_i1} receives 5 spikes, then the literal x_i occurs in the current clause (say C_j), and thus the clause is satisfied by all those assignments in which x_i is true. Neuron σ_{Cx_i0} will also receive 5 spikes, but they will be deleted during the next computation step. On the other hand, if neuron σ_{Cx_i1} receives 6 spikes, then the literal $\neg x_i$ occurs in C_j, and the clause is satisfied by those assignments in which x_i is false. Since neuron σ_{Cx_i1} is designed to process the case in which x_i occurs in C_j, it will delete its 6 spikes. However, neuron σ_{Cx_i0} will also have received 6 spikes, and this time it will send four spikes to those neurons which are bijectively associated with the assignments for which x_i is false (refer to the generation stage for the corresponding synapses). In the next step, those neurons with label t_n or f_n that received at least four spikes send one spike to the corresponding neurons σ_{cl} (the remaining spikes will be forgotten; note that the number of spikes received in neurons with label t_n or f_n is not more than $4n$, because, without loss of generality, we assume that the same literal is not repeated and at most one of literals x_i or $\neg x_i$, for any $1 \leq i \leq n$, can occur in a clause; that is, a clause is a disjunction of at most n literals), with the meaning that the clause is satisfied by the assignments in which x_i is false. This occurs in step $3n+4$. Thus, the check for the satisfiability of the first clause has been performed; in a similar way, the check for the remaining clauses can proceed. All the clauses can thus be checked to see whether there exist assignments that satisfy all of them.

If there exist some assignments that satisfy all the clauses of $\gamma_{n,m}$, then the corresponding neurons with label cl succeed to accumulate m spikes. Thus, the rule $[a^m \to a^2]_{cl}$ can be applied in these neurons. The satisfiability checking process is completed at step $2n + mn + 4$.

Output Stage: From the above explanation, it is not difficult to see that the output neuron receives spikes if and only if $\gamma_{n,m}$ is *true*. Furthermore, the output neuron sends exactly one spike to the environment at step $2n+mn+6$ if and only if $\gamma_{n,m}$ is *true*.

From the previous explanations, one can see that the system correctly answers the question whether or not $\gamma_{n,m}$ is satisfiable. The duration of the computation is polynomial in term of n and m: if the answer is positive, then the system sends one spike to the environment at step $2n + mn + 6$; if the answer is negative, then the system halts in $2n+mn+6$ steps, but does not send any spike to the environment.

Finally, we show that the family $\Pi = \{\Pi(\langle n, m\rangle) \mid n, m \in \mathbb{N}\}$ is polynomially uniform by deterministic Turing machines. We first note that the sets of rules associated with the system $\Pi(\langle n, m\rangle)$ are recursive. Hence, it is enough to note that the amount of necessary resources for defining each system is linear with respect to n, and this is indeed the case, since those resources are the following:

1. Size of the alphabet: $1 \in O(1)$.
2. Initial number of neurons: $4n + 7 \in O(n)$.
3. Initial number of spikes: $9 \in O(1)$.

4. Number of labels for neurons: $6n + 8 \in O(n)$.
5. Size of synapse dictionary: $7n + 6 \in O(n)$.
6. Number of rules: $2n^2 + 14n + 12 \in O(n^2)$.

5 Conclusions and Remarks

With computer science and biological motivation, neuron division and neuron budding are introduced into the framework of spiking neural P systems. We have proven that spiking neural P systems with neuron division and neuron budding can solve **NP**-complete problems in polynomial time. We exemplify this possibility with `SAT` problem.

Both neuron division rules and neuron budding rules can generate exponential workspace in linear time. In this sense, we used a double "power" to solve `SAT` problem in the systems constructed in this paper. It remains open to design efficient spiking neural P systems with either neuron division rules or neuron budding rules, but not both kinds of rules, for solving **NP**-complete problem.

Actually, we have here a larger set of problems. Besides the budding rules of the form considered above, we can also define "serial division rules", of the form $[E]_i \rightarrow [\ ]_j/[\ ]_k$, where E is a regular expression and $i, j, k \in H$. The meaning is obvious: neuron σ_i produces two neurons, σ_j and σ_k, with possibly new labels j, k, linked by synapses as in the case of budding rules. Note that a budding rule is a particular case, with $i = j$. Thus, we can consider three types of rules: parallel division rules, budding rules (these two types were considered above), and serial division rules. Are rules of a single type sufficient in order to devise SN P systems which solve computationally hard problems in polynomial time?

For cell-like P systems, besides membrane division, there is another operation which was proved to provide ways to generate an exponential workspace in polynomial time, useful for trading space for time in solving **NP**-complete problem, namely membrane creation (a membrane is created from objects present in other membranes). Can this idea be also extended to SN P systems? Note that in the case of SN P systems we do not have neurons inside neurons, neither spikes outside neurons, hence this issue does not look easy to handle; maybe further ingredients should be added, such as glia cells, astrocytes, etc.

Acknowledgements

Valuable comments from Jun Wang are greatly appreciated. The work of L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60803113), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of

Hubei Province (2008CDB113 and 2008CDB180). The work of the last two authors was supported by Project of Excellence with *Investigador de Reconocida Valia*, from Junta de Andalucia, grant P08 – TIC 04200.

References

1. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae*, 75 (2007), 141–162.
2. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. *Proceedings of the* 8th *International Conference on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
3. 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*, 7, 2 (2008), 147–166.
4. R. Galli, A. Gritti, L. Bonfanti, A.L. Vescovi: Neural stem cells: an overview. *Circulation Research*, 92 (2003), 598–608.
5. M.R. Garey, D.S. Johnson: *Computers and Intractability. A Guide to the Theory of* **NP***-completeness*. W.H. Freeman and Company, San Francisco, 1979.
6. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2–3 (2006), 279–308.
7. T.-O. Ishdorj, A. Leporati: Uniform solutions to `SAT` and `3-SAT` by spiking neural P systems with pre-computed resources. *Natural Computing*, 7, 4 (2008), 519–534.
8. A. Leporati, M.A. Gutiérrez-Naranjo: Solving `Subset Sum` by spiking neural P systems with pre-computed resources. *Fundamenta Informaticae*, 87, 1 (2008), 61–77.
9. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez: Uniform solutions to `SAT` and `Subset Sum` by spiking neural P systems. *Natural Computing*, online version (DOI: 10.1007/s11047-008-9091-y).
10. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical **NP**-complete problem with spiking neural P systems. *Lecture Notes in Computer Sci.*, 4860 (2007), 336–352.
11. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: On the computational power of spiking neural P systems. *International Journal of Unconventional Computing*, 2007, in press.
12. A. Păun, Gh. Păun: Small universal spiking neural P systems. *BioSystems*, 90, 1 (2007), 48–60.
13. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.
14. The P systems Website: `http://ppage.psystems.eu`.

Efficiency of Tissue P Systems with Cell Separation

Linqiang Pan[1,2], Mario J. Pérez-Jiménez[3]

[1] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, People's Republic of China
lqpan@mail.hust.edu.cn

[2] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, 41012 Sevilla, Spain
marper@us.es

Summary. The most investigated variants of P systems in the last years are cell-like models, especially in terms of efficiency. Recently, different new models of tissue-like (symport/antiport) P systems have received important attention. This paper presents a new class of tissue P systems with cell separation, where cell separation can generate new workspace. Its efficiency is investigated, specifically, (a) only tractable problem can be efficiently solved by using cell separation and communication rules with length at most 1, and (b) an efficient (uniform) solution to SAT problem by using cell separation and communication rules with length at most 6 is presented. Further research topics and open problems are discussed, too.

1 Introduction

Membrane computing is inspired by the structure and the function of living cells, as well as from the organization of cells in tissues, organs, and other higher order structures. The devices of this model, called *P systems*, provide distributed parallel and non-deterministic computing models.

Roughly speaking, the main components of such a model are a cell-like *membrane structures*, in the *compartments* of which one places *multisets* of *symbol-objects* which evolve in a synchronous maximally parallel manner according to given *evolution rules*, also associated with the membranes (for introduction see [19] and for further bibliography see [29]).

Membrane computing is a young branch of natural computing initiated by Gh. Păun in the end of 1998 [17]. It has received important attention from the scientific community since then, with contributions by computer scientists, biologists,

formal linguists and complexity theoreticians, enriching each others with results, open problems and promising new research lines. In fact, membrane computing was selected by the Institute for Scientific Information, USA, as a fast *Emerging Research Front* in computer science, and [20] was mentioned in [28] as a highly cited paper in October 2003.

In the last years, many different models of P systems have been proposed. The most studied variants are characterized by a *cell-like* membrane structure, where the communication takes place between a membrane and the surrounding one. In this model we have a set of nested membranes, in such a way that the graph of neighborhood relation is a tree.

One of the topics in the field is the study of the computational power and efficiency of P systems. In particular, different models of these cell-like P systems have been successfully used in order to design solutions to **NP**-complete problems in polynomial time (see [6] 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. Inspired by living cell, several ways for obtaining exponential workspace in polynomial time were proposed: membrane division (*mitosis*) [18], membrane creation (*autopoiesis*) [8] membrane separation (*membrane fission*) [16]. These three ways have given rise to the corresponding P systems model: *P systems with active membranes*, *P systems with membrane creation*, and *P systems with membranes separation*. These three models are universal from a computational point of view, but technically, they are pretty different. In fact, nowadays there does not exist any theoretical result which proves that these models can simulate each other in polynomial time.

Under the hypothesis $\mathbf{P}\neq\mathbf{NP}$, Zandron et al. [27] established the limitations of P systems that do not use membrane division concerning the efficient solution of **NP**-complete problems. This result was generalized by Pérez-Jiménez et al. [24] obtaining a characterization of the $\mathbf{P}\neq\mathbf{NP}$ conjecture by the polynomial time unsolvability of an **NP**-complete problem by means of language accepting P systems (without using rules that allow to increase the size of the structure of membranes).

Here, we shall focus on another type of P systems, the so-called (because of their membrane structure) *tissue P systems*. Instead of considering a hierarchical arrangement, membranes are placed in the nodes of a virtual graph. This variant has two biological inspirations (see [15]): 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 to P systems in [20]. Symport rules move objects across a membrane together in one direction, whereas antiport rules move objects across a membrane in opposite directions.

From the seminal definitions of tissue P systems [14, 15], several research lines have been developed and other variants have arisen (see, for example, [1, 2, 3, 9, 10, 26]). One of the most interesting variants of tissue P systems was presented in [22]. In that paper, the definition of tissue P systems is combined with the one of P

systems with active membranes, yielding *tissue P systems with cell division*, and a polynomial-time uniform solution to the **NP**-complete problem SAT is shown. In this kind of tissue P systems [22], there exists replication, that is, the two new cells generated by a division rule have exactly the same objects except for at most a pair of different objects. However, in the biological phenomenon of separation, the contents of the two new cells evolved from a cell can be significantly different, and membrane separation inspired by this biological phenomenon in the framework of cell-like P systems was proved to be an efficient way to obtain exponential workspace in polynomial time [16]. In this paper, a new class of tissue P systems based on cell separation, called *tissue P systems with cell separation*, is presented, and a linear time uniform solution to the **NP**-complete problem SAT is shown.

The paper is organized as follows: first, we recall some preliminaries, and then, the definition of tissue P systems with cell separation is given. Next, recognizer tissue P systems are briefly described. In Section 5, we prove that only tractable problem can be efficiently solved by using cell separation and communication rules with length at most 1. In Section 6, an efficient (uniform) solution to SAT problem by using cell separation and communication rules with length at most 6 is shown, including a short overview of the computation and of the necessary resources. The formal verification of the solution is also given. Finally, some discussion is presented.

2 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 $supp(m) = \{x \in A \mid f(x) > 0\}$ 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, and $supp(m) = \{a_1, \ldots, a_k\}$, then it will be denoted as $m = \{\{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}\}$. That is, superscripts indicate the multiplicity of each element, and if $f(x) = 0$ for any $x \in A$, then this element is omitted.

In what follows we assume the reader is already familiar with the basic notions and the terminology of P systems. For details, see [19].

3 Tissue P Systems with Cell Separation

In the first definition of the model of tissue P systems [14, 15], the membrane structure did not change along the computation. We will give a new model of *tissue P systems with cell separation* based on the cell-like model of P systems with membranes separation [16]. The biological inspiration is clear: alive tissues are not *static* network of cells, since new cells are generated by membrane fission in a natural way.

The main features of this model, from the computational point of view, are that cells are not polarized (the contrary holds in the cell-like model of P systems with active membranes, see [19]); the cells obtained by separation have the same labels as the original cell and if a cell is separated, its interaction with other cells or with the environment is blocked during the separation process. In some sense, this means that while a cell is separating it closes its communication channels.

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

$$\Pi = (\Gamma, O_1, O_2, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_0),$$

where:

1. Γ is a finite *alphabet* whose elements are called *objects*, $\Gamma = O_1 \cup O_2$, $O_1, O_2 \neq \emptyset$, $O_1 \cap O_2 = \emptyset$;
2. $w_1, \ldots, w_q$ are strings over Γ, representing the multisets of objects placed in the q cells of the system at the beginning of the computation;
3. $\mathcal{E} \subseteq \Gamma$ is a finite alphabet representing the set of objects in the environment in arbitrary copies each;
4. $\mathcal{R}$ is a finite set of rules of the following forms:
   (a) $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$;
   Communication rules; $1, 2, \cdots, q$ identify the cells of the system, 0 is the environment; when applying a rule $(i, u/v, j)$, the objects of the multiset represented by u are sent from region i to region j and simultaneously the objects of the multiset v are sent from region j to region i, ($|u| + |v|$ is called the length of the communication rule $(i, u/v, j)$);
   (b) $[a]_i \rightarrow [O_1]_i[O_2]_i$, where $i \in \{1, 2, \ldots, q\}$ and $a \in \Gamma$;
   Separation rules; in reaction with an object a, the cell is separated into two cells with the same label; at the same time, object a is consumed; the objects from O_1 are placed in the first cell, those from O_2 are placed in the second cell;
5. $i_0 \in \{0, 1, 2, \ldots, q\}$ is the output cell.

The rules of a system like the above one are used in the non-deterministic maximally parallel manner as customary in membrane computing. At each step, all cells which can evolve must evolve in a maximally parallel way (in each step we apply a multiset of rules which is maximal, no further rule can be added). This way of applying rules has only one restriction: when a cell is separated, the separation rule is the only one which is applied for that cell in that step; the objects

inside that cell do not evolve by means of communication rules. The new cells could participate in the interaction with other cells or the environment by means of communication rules in the next step – providing that they are not separated once again. Their labels precisely identify the rules which can be applied to them.

The configuration of tissue P system with cell separation is described by all multisets of objects associated with all the cells and environment $(w'_1, \cdots, w'_q; w'_0)$, where w'_0 is a multiset over $\Gamma - \mathcal{E}$ representing the objects present in the environment having a finite multiplicity. The tuple $(w_1, w_2, \cdots, w_q; \emptyset)$ is the initial configuration. The computation starts from the initial configuration and proceeds as defined above; only halting computations give a result, and the result is encoded by the objects present in cell i_0 in the halting configuration.

4 Recognizer Tissue P Systems with Cell Separation

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, the notions from classical *computational complexity theory* are adapted for membrane computing, and a special class of cell-like P systems is introduced in [25]: *recognizer P systems*. For tissue P systems, with the same idea as recognizer cell-like P systems, *recognizer tissue P systems* is introduced in [22].

A *recognizer tissue P system with cell separation* of degree $q \geq 1$ is a construct

$$\Pi = (\Gamma, O_1, O_2, \Sigma, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_{in}, i_0)$$

where:

- $(\Gamma, O_1, O_2, w_1, \ldots, w_q, \mathcal{E}, \mathcal{R}, i_o)$ is a tissue P system with cell separation of degree $q \geq 1$ (as defined in the previous section).
- The working alphabet Γ has two distinguished objects `yes` and `no`, at least one copy of them present in some initial multisets w_1, ..., w_q, but none of them are 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 object `yes` or object `no` (but not both) must have been released into the environment, and only at the last step of the computation.

The computations of the system Π with input $w \in \Sigma^*$ start from a configuration of the form $(w_1, w_2, \ldots, w_{i_{in}} w, \ldots, w_q; \emptyset)$, that is, after adding the multiset w to the contents of the input cell i_{in}. We say that $\mathcal{C}$ is an accepting computation

(respectively, rejecting computation) if object `yes` (respectively, `no`) appears in the environment associated to the corresponding halting configuration of $\mathcal{C}$.

We denote by $\mathbf{TSC}(k)$ the class of recognizer tissue P systems with cell separation and with communication rules of length at most k.

Definition 1. *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer tissue P systems with cell separation 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.*

We denote by $\mathbf{PMC}_{\mathbf{TSC}(k)}$ the set of all decision problems which can be solved by means of recognizer tissue P systems from $\mathbf{TSC}(k)$.

5 Limitation on the Efficiency of TSC(1)

In this section, we consider the efficiency of tissue P systems with cell separation and communication rules with length 1. Specifically, we shall prove that such systems can efficiently solve only tractable problems.

Let Π be a tissue P system with cell separation and let all communication rules be of length 1. In this case, each rule of the system can be activated by a single object. Hence, there exists, in some sense, a *dependency* between the object triggering the rule and the object or objects produced by its application. This dependency allows to adapt the ideas developed in [13] for cell-like P systems with active membranes to tissue P systems with cell separation and communication rules of length 1.

We can consider a general pattern $(a, i) \rightarrow (b_1, j) \dots (b_s, j)$ where $i, j \in \{0, 1, 2, \dots, q\}$, and $a, b_k \in \Gamma$, $k \in \{1, \cdots, s\}$. This pattern can be interpreted

as follows: from the object a in the cell (or in the environment) labeled with i we can *reach* the objects $b_1, \ldots, b_s$ in the cell (or in the environment) labeled with j. Communication rules correspond to the case $s = 1$ and $b_1 = a$.

Without loss of generality we can assume that all communication rules in the system obey the syntax $(i, a/\lambda, j)$, since every rule of the form $(j, \lambda/a, i)$ can be rewritten to follow the above syntax, with equivalent semantics.

We formalize these ideas in the following definition.

Definition 2. *Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in})$ be a tissue P system of degree $q \geq 1$ with cell separation. Let $H = \{0, 1, \ldots, q\}$. The dependency graph associated with Π is the directed graph $G_\Pi = (V_\Pi, E_\Pi)$ defined as follows:*
$V_\Pi = \{(a, i) \in \Gamma \times H : \exists j \in H \ ((i, a/\lambda, j) \in R \ \vee \ (j, a/\lambda, i) \in R)\}$,
$E_\Pi = \{((a, i), (b, j)) : (a = b \ \wedge \ (i, a/\lambda, j) \in R)\}$.

Note that when a separation rule is applied, objects do not evolve, and the labels of membranes do not change, so separation rules do not add any nodes and edges to the associated dependency graph.

Proposition 1. *Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in})$ be a tissue P system with cell separation, in which the length of all communication rules is 1. Let $H = \{0, 1, \ldots, q\}$. There exists a deterministic Turing machine that constructs the dependency graph G_Π associated with Π, in polynomial time (that is, in a time bounded by a polynomial function depending on the total number of communication rules).*

Proof. A deterministic algorithm that, given a P system Π with the set R_c of communication rules, constructs the corresponding dependency graph, is the following:

```
   Input: Π (with R_c as its set of communication rules)
V_Π ← ∅; E_Π ← ∅
for each rule r ∈ R_c of Π do
  if r = (i, a/λ, j) then
    V_Π ← V_Π ∪ {(a, i), (a, j)}; E_Π ← E_Π ∪ {((a, i), (a, j))}
```

The running time of this algorithm is bounded by $O(|R_c|)$.

Proposition 2. *Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in})$ be a tissue P system with cell separation, in which the length of all communication rules is 1. Let $H = \{0, 1, \ldots, q\}$. Let Δ_Π be defined as follows:*

$$\Delta_\Pi = \{(a, i) \in \Gamma \times H : \textit{there exists a path (within the dependency graph) from } (a, i) \textit{ to } (\mathtt{yes}, 0)\}.$$

Then, there exists a Turing machine that constructs the set Δ_Π in polynomial time (that is, in a time bounded by a polynomial function depending on the total number of communication rules).

Proof. We can construct the set Δ_Π from Π as follows:

- We construct the dependency graph G_Π associated with Π.
- Then we consider the following algorithm:

```
Input: G_Π = (V_Π, E_Π)
  Δ_Π ← ∅
  for each (a, i) ∈ V_Π do
    if reachability (G_Π, (a, i), (yes, 0)) = yes then
      Δ_Π ← Δ_Π ∪ {(a, i)}
```

The running time of this algorithm [3] is of order $O(|V_\Pi| \cdot |V_\Pi|^2)$, hence it is of order $O(|\Gamma|^3 \cdot |H|^3)$.

Notation: Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$ be a tissue P system with cell separation. Let m be a multiset over Σ. Then we denote $\mathcal{M}_j^* = \{(a, j) : a \in \mathcal{M}_j\}$, for $1 \leq j \leq q$, and $m^* = \{(a, i_{in}) : a \in m\}$.

Below we characterize accepting computations of a recognizer tissue P system with cell separation and communication rules of length 1 by distinguished paths in the associated dependency graph.

Lemma 1. *Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in})$ be a recognizer confluent tissue P system with cell separation in which the length of all communication rules is 1. The following assertions are equivalent:*

(1) There exists an accepting computation of Π.
(2) There exists $(a_0, i_0) \in \bigcup_{j=1}^{q} \mathcal{M}_j^*$ and a path in the dependency graph associated with Π, from (a_0, i_0) to $(\mathtt{yes}, 0)$.

Proof. (1) $\Rightarrow$ (2). First, we show that for each accepting computation $\mathcal{C}$ of Π there exists $(a_0, i_0) \in \bigcup_{j=1}^{q} \mathcal{M}_j^*$ and a path $\gamma_\mathcal{C}$ in the dependency graph associated with Π from (a_0, i_0) to $(\mathtt{yes}, 0)$. By induction on the length n of $\mathcal{C}$.

If $n = 1$, a single step is performed in $\mathcal{C}$ from C_0 to C_1. A rule of the form $(j, \mathtt{yes}/\lambda, 0)$, with $\mathtt{yes} \in \Gamma, j \neq 0$, has been applied in that step. Then, $(\mathtt{yes}, j) \in \mathcal{M}_j^*$, for some $j \in \{1, \ldots, q\}$. Hence, $((\mathtt{yes}, j), (\mathtt{yes}, 0))$ is a path in the dependency graph associated with Π.

Let us suppose that the result holds for n. Let $\mathcal{C} = (C_0, C_1, \ldots, C_n, C_{n+1})$ be an accepting computation of Π. Then $\mathcal{C}' = (C_1, \ldots, C_n, C_{n+1})$ is an accepting computation of the system $\Pi' = (\Gamma, \Sigma, \Omega, \mathcal{M}_1', \ldots, \mathcal{M}_q', \mathcal{R}, i_{in})$, where $\mathcal{M}_j'$ is the

[3] The Reachability Problem is the following: *given a (directed or undirected) graph, G, and two nodes a, b, determine whether or not the node b is reachable from a, that is, whether or not there exists a path in the graph from a to b.* It is easy to design an algorithm running in polynomial time solving this problem. For example, given a (directed or undirected) graph, G, and two nodes a, b, we consider a depth–first–search with source a, and we check if b is in the tree of the computation forest whose root is a. The total running time of this algorithm is $O(|V| + |E|)$, that is, in the worst case is quadratic in the number of nodes. Moreover, this algorithm needs to store a linear number of items (it can be proved that there exists another polynomial time algorithm which uses $O(\log^2(|V|))$ space).

content of cell j in configuration C_1, for $1 \leq j \leq q$. By induction hypothesis there exists an object b_0 in a cell i_0 from C_1, and a path in the dependency graph associated with Π' from (b_0, i_0) to $(\texttt{yes}, 0)$. If (b_0, i_0) is an element of configuration C_0 (that means that in the first step a separation rule has been applied to cell i_0), then the result holds. Otherwise, there is an element (a_0, j_0) in C_0 producing (b_0, i_0). So, there exists a path $\gamma_{\mathcal{C}}$ in the dependency graph associated with Π from (a_0, j_0) to $(\texttt{yes}, 0)$.

(2) $\Rightarrow$ (1). Let us prove that for each $(a_0, i_0) \in \bigcup_{j=1}^{q} \mathcal{M}_j^*$ and for each path in the dependency graph associated with Π from (a_0, i_0) to $(\texttt{yes}, 0)$, there exists an accepting computation of Π. By induction on the length n of the path.

If $n = 1$, we have a path $((a_0, i_0), (\texttt{yes}, 0))$. Then, $a_0 = \texttt{yes}$ and the computation $\mathcal{C} = (C_0, C_1)$ where the rule $(i_0, \texttt{yes}/\lambda, 0)$ belongs to a multiset of rules m_0 that produces configuration C_1 from C_0 is an accepting computation of Π.

Let us suppose that the result holds for n. Let

$$((a_0, i_0), (a_1, i_1), \ldots (a_n, i_n), (\texttt{yes}, 0))$$

be a path in the dependency graph of length $n + 1$. Let C_1 be the configuration of Π reached from C_0 by the application of a multiset of rules containing the rule that produces (a_1, i_1) from (a_0, i_0). Then $((a_1, i_1), \ldots (a_n, i_n), (\texttt{yes}, 0))$ is a path of length n in the dependency graph associated with the system

$$\Pi' = (\Gamma, \Sigma, \Omega, \mathcal{M}_1', \ldots, \mathcal{M}_q', \mathcal{R}, i_{in})$$

where $\mathcal{M}_j'$ is the content of cell j in configuration C_1, for $1 \leq j \leq q$. By induction hypothesis, there exists an accepting computation $\mathcal{C}' = (C_1, \ldots, C_t)$ of Π'. Hence, $\mathcal{C} = (C_0, C_1, \ldots, C_t)$ is an accepting computation of Π.

Next, given a family $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ of recognizer tissue P system with cell separation in which the length of all communication rules is 1, solving a decision problem, we will characterize the acceptance of an instance of the problem, w, using the set $\Delta_{\Pi(s(w))}$ associated with the system $\Pi(s(w))$, that processes the given instance w. More precisely, the instance is accepted by the system if and only if there is an object in the initial configuration of the system $\Pi(s(w))$ with input $cod(w)$ such that there exists a path in the associated dependency graph starting from that object and reaching the object $\texttt{yes}$ in the environment.

Proposition 3. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer tissue P system with cell separation in which the length of all communication rules is 1 solving X, according to Definition 1. Let (cod, s) be the polynomial encoding associated with that solution. Then, for each instance w of the problem X the following assertions are equivalent:*

(a) $\theta_X(w) = 1$ (that is, the answer to the problem is $\texttt{yes}$ for w).

(b) $\Delta_{\Pi(s(w))} \cap ((cod(w))^* \cup \bigcup_{j=1}^{p} \mathcal{M}_j^*) \neq \emptyset$, where $\mathcal{M}_1, \ldots, \mathcal{M}_p$ are the initial multisets of the system $\Pi(s(w))$.

Proof. Let $w \in I_X$. Then $\theta_X(w) = 1$ if and only if there exists an accepting computation of the system $\Pi(s(w))$ with input multiset $cod(w)$. By Lemma 1, this condition is equivalent to the following: in the initial configuration of $\Pi(s(w))$ with input multiset $cod(w)$ there exists at least one object $a \in \Gamma$ in a cell labelled with i such that in the dependency graph the node $(\texttt{yes}, 0)$ is reachable from (a, i).

Hence, $\theta_X(w) = 1$ if and only if $\Delta_{\Pi(s(w))} \cap \mathcal{M}_j^* \neq \emptyset$ for some $j \in \{1, \ldots, p\}$, or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$.

Theorem 1. $\mathbf{P} = \boldsymbol{PMC}_{TSC(1)}$

Proof. We have $\mathbf{P} \subseteq \mathbf{PMC}_{TSC(1)}$ because the class $\mathbf{PMC}_{TSC(1)}$ is closed under polynomial time reduction. In what follows, we show that $\mathbf{PMC}_{TSC(1)} \subseteq \mathbf{P}$. Let $X \in \mathbf{PMC}_{TSC(1)}$ and let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer tissue P systems with cell separation solving X, according to Definition 1. Let (cod, s) be the polynomial encoding associated with that solution.

We consider the following deterministic algorithm:

```
Input: An instance w of X
- Construct the system Π(s(w)) with input multiset cod(w).
- Construct the dependency graph G_{Π(s(w))} associated with Π(s(w)).
- Construct the set Δ_{Π(s(w))} as indicated in Proposition 2
  answer ← no; j ← 1
  while j ≤ p ∧ answer = no do
      if Δ_{Π(s(w))} ∩ M_j^* ≠ ∅ then
          answer ← yes
      j ← j + 1
  endwhile
  if Δ_{Π(s(w))} ∩ (cod(w))^* ≠ ∅ then
      answer ← yes
```

On one hand, the answer of this algorithm is `yes` if and only if there exists a pair (a, i) belonging to $\Delta_{\Pi(s(w))}$ such that the symbol a appears in the cell labelled with i in the initial configuration (with input the multiset $cod(w)$).

On the other hand, a pair (a, i) belongs to $\Delta_{\Pi(s(w))}$ if and only if there exists a path from (a, i) to $(\texttt{yes}, 0)$, that is, if and only if we can obtain an accepting computation of $\Pi(s(w))$ with input $cod(w)$. Hence, the algorithm above described solves the problem X.

The cost to determine whether or not $\Delta_{\Pi(s(w))} \cap \mathcal{M}_j^* \neq \emptyset$ (or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$) is of order $O(|\Gamma|^2 \cdot |H|^2)$.

Hence, the running time of this algorithm can be bounded by $f(|w|) + O(|R_c|) + O(q \cdot |\Gamma|^2 \cdot n^2)$, where f is the (total) cost of a polynomial encoding from X to $\mathbf{\Pi}$, R_c is the set of rules of $\Pi(s(w))$, and q is the number of (initial) cells of $\Pi(s(w))$. Furthermore, by Definition 1 we have that all involved parameters are polynomial in $|w|$. That is, the algorithm is polynomial in the size $|w|$ of the input.

6 Solving Computationally Hard Problems by Using TSC(6)

In this section, we consider the efficiency of tissue P systems with cell separation and communication rules of length at most 6. As expected, such systems can efficiently solve computationally hard problems. In what follows, we show how to efficiently solve SAT problem by such systems.

The SAT problem is the following: given a boolean formula in conjunctive normal form (CNF), to determine whether or not there exists an assignment to its variables on which it evaluates true. This is a well known **NP**-complete problem [5].

The solution proposed follows a brute force approach in the framework of recognizer P systems with cell separation. The solution consists of the following stages:

- *Generation Stage*: All truth assignments for the n variables are produced by using cell separation in an adequate way.
- *Checking Stage*: We determine whether there is a truth assignment that makes the boolean formula evaluate to true.
- *Output Stage*: The system sends to the environment the right answer according to the results of the previous stage.

Let us consider the polynomial time computable function $\langle n, m\rangle = ((m + n)(m + n + 1)/2) + m$ (the pair function), which is a primitive recursive and bijective function from $\mathbb{N}^2$ to $\mathbb{N}$. We shall construct a family $\mathbf{\Pi} = \{\Pi(t) \mid t \in \mathbb{N}\}$ such that each system $\Pi(t)$ will solve all instances of SAT problem with n variables and m clauses, where $t = \langle n, m\rangle$, provided that the appropriate input multisets are given.

For each $n, m \in \mathbb{N}$,

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

with the following components:

- $\Gamma(\langle n, m\rangle) = O_1 \cup O_2$,
  $O_1 = \{x_{i,j}, \bar{x}_{i,j}, c_{i,j}, z_{i,j} \mid 1 \le i \le n, 1 \le j \le m\} \cup$
  $\{A_i \mid 1 \le i \le n\} \cup \{a_{1,i}, b_{1,i}, g_i, h_i \mid 1 \le i \le n-1\} \cup$
  $\{d_{1,i}, e_i, l_i \mid 1 \le i \le n-2\} \cup \{a_{2,i}, b_{2,i}, d_{2,i} \mid 2 \le i \le n-1\} \cup$
  $\{a_{i,j,k}, b_{i,j,k}, d_{i,j,k} \mid 1 \le i \le n, 1 \le j \le m, 1 \le k \le n-1\} \cup$
  $\{B_i \mid 1 \le i \le 4n\} \cup \{C_i \mid 1 \le i \le 3n\} \cup$
  $\{D_i \mid 1 \le i \le 4n + 2m\} \cup \{E_i \mid 1 \le i \le 4n + 2m + 3\} \cup$
  $\{r_j \mid 1 \le j \le m\} \cup \{T_i, F_i, t_i, f_i \mid 1 \le i \le n\} \cup \{c, p, s, y, z, \texttt{yes}, \texttt{no}\}$,
  $O_2 = \{x'_{i,j}, \bar{x}'_{i,j}, z'_{i,j}\} \mid 1 \le i \le n, 1 \le j \le m\} \cup$
  $\{T'_i, F'_i \mid 1 \le i \le n\} \cup \{y', z'\}$.
- $\Sigma(\langle n, m\rangle) = \{c_{i,j}, x_{i,j}, \bar{x}_{i,j} \mid 1 \le i \le n, 1 \le j \le m\}$.
- $w_1 = \{\{a_{1,1}, a_{2,2}, g_1, B_1, C_1, D_1, E_1, p, \texttt{yes}, \texttt{no}\}\} \cup$
  $\{\{a_{i,j,1} \mid 1 \le i \le n, 1 \le j \le m\}\}$.

- $w_2 = A_1$.
- $\mathcal{R}(\langle n, m\rangle)$ is the set of rules:
  1. **Separation rule:**
     $r_1 \equiv [s]_2 \rightarrow [O_1]_2[O_2]_2$.
  2. **Communication rules:**
     $r_{2,i,j,k} \equiv (1, a_{i,j,k}/b_{i,j,k}, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq k \leq n-1$;
     $r_{3,i,j,k} \equiv (1, b_{i,j,k}/c_{i,j}^2 d_{i,j,k}^2, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq k \leq n-2$;
     $r_{4,i,j} \equiv (1, b_{i,j,n-1}/c_{i,j}^2, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
     $r_{5,i,j,k} \equiv (1, d_{i,j,k}/a_{i,j,k+1}, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$, $1 \leq k \leq n-2$;
     $r_{6,i} \equiv (1, a_{1,i}/b_{1,i}, 0)$ for $1 \leq i \leq n-1$;
     $r_{7,i} \equiv (1, b_{1,i}/c^2 d_{1,i}^2 e_i^2, 0)$ for $1 \leq i \leq n-2$;
     $r_8 \equiv (1, b_{1,n-1}/c^2, 0)$;
     $r_{9,i} \equiv (1, d_{1,i}/a_{1,i+1}, 0)$ for $1 \leq i \leq n-2$;
     $r_{10,i} \equiv (1, e_i/a_{2,i+1}, 0)$ for $1 \leq i \leq n-2$;
     $r_{11,i} \equiv (1, a_{2,i}/b_{2,i}, 0)$ for $2 \leq i \leq n-1$;
     $r_{12,i} \equiv (1, b_{2,i}/c^2 d_{2,i}^2, 0)$ for $2 \leq i \leq n-2$;
     $r_{13} \equiv (1, b_{2,n-1}/c^2, 0)$;
     $r_{14,i} \equiv (1, d_{2,i}/a_{2,i+1}, 0)$ for $2 \leq i \leq n-2$;
     $r_{15,i} \equiv (1, g_i/h_i, 0)$ for $1 \leq i \leq n-1$;
     $r_{16,i} \equiv (1, h_i/l_i^2 A_{i+1}^2, 0)$ for $1 \leq i \leq n-2$;
     $r_{17} \equiv (1, h_{n-1}/A_n^2, 0)$;
     $r_{18,i} \equiv (1, l_i/g_{i+1}, 0)$ for $1 \leq i \leq n-2$;
     $r_{19,i,j} \equiv (2, c_{i,j}x_{i,j}/z_{i,j}z'_{i,j}x_{i,j}x'_{i,j}, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
     $r_{20,i,j} \equiv (2, c_{i,j}\bar{x}_{i,j}/z_{i,j}z'_{i,j}\bar{x}_{i,j}\bar{x}'_{i,j}, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
     $r_{21,i,j} \equiv (2, c_{i,j}x'_{i,j}/z_{i,j}z'_{i,j}x_{i,j}x'_{i,j}, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
     $r_{22,i,j} \equiv (2, c_{i,j}\bar{x}'_{i,j}/z_{i,j}z'_{i,j}\bar{x}_{i,j}\bar{x}'_{i,j}, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
     $r_{23,i} \equiv (2, A_i/T_i F'_i zz'yy's, 0)$ for $1 \leq i \leq n-1$;
     $r_{24} \equiv (2, A_n/T_n F'_n yy's, 0)$;
     $r_{25,i} \equiv (2, cT_i/zz'T_iT'_i, 0)$ for $1 \leq i \leq n-1$;
     $r_{26,i} \equiv (2, cT'_i/zz'T_iT'_i, 0)$ for $1 \leq i \leq n-1$;
     $r_{27,i} \equiv (2, cF_i/zz'F_iF'_i, 0)$ for $1 \leq i \leq n-1$;
     $r_{28,i} \equiv (2, cF'_i/zz'F_iF'_i, 0)$ for $1 \leq i \leq n-1$;
     $r_{29,i,j} \equiv (1, c_{i,j}/z_{i,j}, 2)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
     $r_{30,i,j} \equiv (1, c_{i,j}/z'_{i,j}, 2)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
     $r_{31} \equiv (1, c/z, 2)$;
     $r_{32} \equiv (1, c/z', 2)$;
     $r_{33,i} \equiv (1, A_i/y, 2)$ for $2 \leq i \leq n$;
     $r_{34,i} \equiv (1, A_i/y', 2)$ for $2 \leq i \leq n$;
     $r_{35,i} \equiv (1, B_i/B_{i+1}, 0)$ for $1 \leq i \leq 2n-1$;
     $r_{36,i} \equiv (1, B_i/B_{i+1}^2, 0)$ for $2n \leq i \leq 3n-1$;
     $r_{37,i} \equiv (1, C_i/C_{i+1}, 0)$ for $1 \leq i \leq 2n-1$;
     $r_{38,i} \equiv (1, C_i/C_{i+1}^2, 0)$ for $2n \leq i \leq 3n-1$;
     $r_{39,i} \equiv (1, D_i/D_{i+1}, 0)$ for $1 \leq i \leq 2n-1$;
     $r_{40,i} \equiv (1, D_i/D_{i+1}^2, 0)$ for $2n \leq i \leq 3n-1$;

$r_{41,i} \equiv (1, E_i/E_{i+1}, 0)$ for $1 \leq i \leq 4n + 2m + 2$;
$r_{42,i,j} \equiv (1, z_{i,j}/\lambda, 0)$;
$r_{43,i,j} \equiv (1, z'_{i,j}/\lambda, 0)$;
$r_{44} \equiv (1, y/\lambda, 0)$;
$r_{45} \equiv (1, y'/\lambda, 0)$;
$r_{46} \equiv (1, z/\lambda, 0)$;
$r_{47} \equiv (1, z'/\lambda, 0)$;
$r_{48} \equiv (1, B_{3n}C_{3n}D_{3n}/y, 2)$;
$r_{49} \equiv (1, B_{3n}C_{3n}D_{3n}/y', 2)$;
$r_{50,i} \equiv (2, C_{3n}T_i/C_{3n}t_i, 0)$ for $1 \leq i \leq n$;
$r_{51,i} \equiv (2, C_{3n}T'_i/C_{3n}t_i, 0)$ for $1 \leq i \leq n$;
$r_{52,i} \equiv (2, C_{3n}F_i/C_{3n}f_i, 0)$ for $1 \leq i \leq n$;
$r_{53,i} \equiv (2, C_{3n}F'_i/C_{3n}f_i, 0)$ for $1 \leq i \leq n$;
$r_{54,i} \equiv (2, B_i/B^2_{i+1}, 0)$ for $3n \leq i \leq 4n - 1$;
$r_{55,i} \equiv (2, D_i/D_{i+1}, 0)$ for $3n \leq i \leq 4n - 1$;
$r_{56,i,j} \equiv (2, B_{4n}t_i x_{i,j}/B_{4n}t_i r_j, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
$r_{57,i,j} \equiv (2, B_{4n}t_i \bar{x}_{i,j}/B_{4n}t_i, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
$r_{58,i,j} \equiv (2, B_{4n}t_i x'_{i,j}/B_{4n}t_i r_j, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
$r_{59,i,j} \equiv (2, B_{4n}t_i \bar{x}'_{i,j}/B_{4n}t_i, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
$r_{60,i,j} \equiv (2, B_{4n}f_i \bar{x}_{i,j}/B_{4n}f_i r_j, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
$r_{61,i,j} \equiv (2, B_{4n}f_i x_{i,j}/B_{4n}f_i, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
$r_{62,i,j} \equiv (2, B_{4n}f_i \bar{x}'_{i,j}/B_{4n}f_i r_j, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
$r_{63,i,j} \equiv (2, B_{4n}f_i x'_{i,j}/B_{4n}r_j, 0)$ for $1 \leq i \leq n$, $1 \leq j \leq m$;
$r_{64,i} \equiv (2, D_i/D_{i+1}, 0)$ for $4n \leq i \leq 4n + m - 1$;
$r_{65,i} \equiv (2, D_{4n+m+i}r_{i+1}/D_{4n+m+i+1}, 0)$ for $0 \leq i \leq m - 1$;
$r_{66} \equiv (2, D_{4n+2m}/\lambda, 1)$;
$r_{67} \equiv (1, D_{4n+2m}\ p\ \texttt{yes}/\lambda, 0)$;
$r_{68} \equiv (1, E_{4n+2m+3}\ p\ \texttt{no}/\lambda, 0)$.

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

6.1 An Overview of the Computation

A family of recognizer tissue P systems with cell separation is constructed above. For an instance of SAT problem $\varphi = M_1 \wedge \cdots \wedge M_m$, consisting of m clauses $M_i = y_{i,1} \vee \cdots \vee y_{i,l_i}$, $1 \leq i \leq m$, where $Var(\varphi) = \{x_1, \cdots, x_n\}$, $y_{i,k} \in \{x_j, \neg x_j \mid 1 \leq j \leq n\}$, $1 \leq i \leq m, 1 \leq k \leq l_i$, the *size* mapping on the set of instances is defined as $s(\varphi) = \langle n, m\rangle = ((m + n)(m + n + 1)/2) + m$, the encoding of the instance is the multiset $cod(\varphi) = \{\{c_{i,j}x_{i,j} \mid x_i \in \{y_{j,k} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq n, 1 \leq j \leq m\}\} \cup \{\{c_{i,j}\bar{x}_{i,j} \mid \neg x_i \in \{y_{j,k} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq n, 1 \leq j \leq m\}\}$.

Now, we informally describe how system $\Pi(s(\varphi))$ with input $cod(\varphi)$ works.

Let us start with the *generation stage*. This stage has several parallel processes, which we describe in several items.

- In cells with label 2, by rules $r_{19,i,j}$ – $r_{22,i,j}$, objects $c_{i,j}x_{i,j}$, $c_{i,j}\bar{x}_{i,j}$, $c_{i,j}x'_{i,j}$, $c_{i,j}\bar{x}'_{i,j}$ introduce objects $z_{i,j}z'_{i,j}x_{i,j}x'_{i,j}$, $z_{i,j}z'_{i,j}\bar{x}_{i,j}\bar{x}'_{i,j}$, $z_{i,j}z'_{i,j}x_{i,j}x'_{i,j}$, $z_{i,j}z'_{i,j}\bar{x}_{i,j}\bar{x}'_{i,j}$, respectively. In the next step, the objects with prime and the objects without prime are separated into the new daughter cells with label 2. The idea is that $c_{i,j}$ is used to duplicate $x_{i,j}$ and $\bar{x}_{i,j}$ (in the sense ignoring the prime), so that one copy of each of them will appear in each cell with label 2. The objects $z_{i,j}$ and $z'_{i,j}$ in cells with label 2 are exchanged with the objects $c_{i,j}$ in the cell with label 1 by the rules $r_{29,i,j}$ and $r_{30,i,j}$. In this way, the cycle of duplication-separation can be iterated.

- In parallel with the above duplication-separation process, the objects c are used to duplicate the objects T_i, T'_i, F_i, and F'_i by the rules $r_{25,i}$ – $r_{28,i}$; the rules r_{31} and r_{32} take care of introducing the object c from the cell with label 1 to cells with label 2.

- In the initial configuration of the system, the cell with label 2 contains an object A_1 (A_i encodes the i-th variable in the propositional formula). The objects T_1, F'_1, z, z', y, y' and s are brought in the cell with label 2, in exchange of A_1, by the rule $r_{23,i}$. The objects T_1 and F'_1 correspond to the values *true* and *false* which the variable x_1 may assume (in general, T_i (or T'_i) and F_i (or F'_i) are for the variable x_i), and in the next step they are separated into the new daughter cells with label 2 by separation rule, because $T_1 \in O_1$ and $F'_1 \in O_2$. The object s is used to activate the separation rule r_1, and is consumed during the application of separation rule. The objects y and y' are used to introduce A_2 from the cell with label 1, and the process of truth-assignment for variable x_2 can continue. In this way, in $3n-1$ steps, we get 2^n cells with label 2, and each one contains one of the 2^n possible truth-assignments for the n variables.

- In parallel with the operations in the cells with label 2, the objects $a_{i,j,k+1}$ from the cell with label 1 are traded for objects $b_{i,j,k+1}$ from the environment at the step $3k+1$ ($0 \leq k \leq n-3$) by the rule $r_{2,i,j,k}$. In the next step, each object $b_{i,j,k+1}$ is traded for two copies of objects $c_{i,j}$ and $d_{i,j,k+1}$ by the rule $r_{3,i,j,k}$. At step $3k+3$ ($0 \leq k \leq n-3$), the object $d_{i,j,k}$ is traded for object $a_{i,j,k+2}$ by the rule $r_{4,i,j,k}$. Especially, at step $3n-5$, $a_{i,j,n-1}$ is traded for $b_{i,j,n-1}$ by the $r_{2,i,j,k}$, at step $3n-4$, each copy of object $b_{i,j,n-1}$ is traded for two copies of $c_{i,j}$ by the $r_{4,i,j}$. After step $3n-4$, there is no object $a_{i,j,k}$ appears in the cell with label 1, and the group of rules $r_{2,i,j,k}$ – $r_{5,i,j,k}$ will not be used again. Note that the subscript k of the object $a_{i,j,k}$ grows by 1 in every 3 steps until reaching the value $n-1$, and the number of copies of $a_{i,j,k}$ is doubled in every 3 steps. At step $3k+3$ ($0 \leq k \leq n-2$), the cell with label 1 has 2^{k+1} copies of object $c_{i,j}$. At the same time, we have 2^{k+1} cells with label 2, and each cell with label 2 contains one copy of object $z_{i,j}$ or one copy of object $z'_{i,j}$. Due to the maximality of the parallelism of using the rules, each cell with label 2 gets exactly one copy of $c_{i,j}$ from the cell with label 1 by the rules $r_{29,i,j}$ and $r_{30,i,j}$. The object $c_{i,j}$ in cell with label 2 is used for duplication as described above.

- The objects $a_{1,i}$ and $a_{2,i}$ in the cell with label 1 has the similar role as object $a_{i,j,k}$ in cell 1, which introduce appropriate copies of object c for the duplication of objects T_i, T'_i, F_i, and F'_i by the rules $r_{6,i} - r_{14,i}$. Note that at step $3k+3$ ($0 \leq k \leq n-2$), there are $2^{k+1}(k+1)$ copies of object c, which ensure each cell with label 2 gets $k+1$ copies of object c by the maximality of the parallelism of using the rules.
- The object g_{i+1} in the cell with label 1 is traded for h_{i+1} from the environment at step $3i+1$ ($0 \leq i \leq n-3$) by the rule $r_{15,i}$. In the next step, the object h_{i+1} is traded for two copies of objects l_{i+1} and A_{i+2} by the rule $r_{13,i}$. At the step $3i+3$ ($0 \leq i \leq n-3$), the object l_{i+1} is traded for two copies of g_{i+2}, so that the process can be iterated, until the subscript i of g_i reaches $n-1$. Especially, at step $3n-5$, object g_{n-1} is traded for h_{n-1} by the rule $r_{15,i}$, at step $3n-4$, each object h_{n-1} is traded for two copies of A_n. After step $3n-4$, there is no object g_i appears in the cell with label 1, and the group of rules $r_{15,i} - r_{18,i}$ will not be used again. At the step $3i+3$ ($0 \leq i \leq n-2$), the cell with label 1 contains 2^{i+1} copies of A_{i+2}, and we have 2^{i+1} cells with label 2, each of them contains one copy of object y or one copy of object y'. Due to the maximality of the parallelism of using the rules, each cell with label 2 gets exactly one copy of A_{i+2} from the cell 1 by the rules $r_{33,i}$ and $r_{34,i}$. In this way, the truth-assignment for the valuable x_{i+1} can continue.
- The counters B_i, C_i, D_i, and E_i in the cell with label 1 grow their subscripts by the rules $r_{35,i} - r_{41,i}$. From step $2n$ to step $3n-1$, the number of copies of objects of the first three types is doubled, hence after $3n-1$ steps, the cell with label 1 contains 2^n copies of B_{3n}, C_{3n}, and D_{3n}. Objects B_i will check which clauses are satisfied by a given truth-assignment, objects C_i are used to multiply the number of copies of t_i, f_i as we will see immediately, objects D_i are used to check whether there is at least one truth-assignment which satisfies all clauses, and E_i will be used to bring the object `no` to the environment, if this will be the case, in the end of the computation.
- The objects $z_{i,j}$, $z'_{i,j}$, y, y', z, and z' in the cell with label 1 are removed by the rules $r_{42,i,j} - r_{47}$. (Actually, if the objects $z_{i,j}$, $z'_{i,j}$, y, y', z, and z' stay in the cell with label 1, they do not influence the work of the system. The rules $r_{38} - r_{43}$ are designed just in order to simplify the formal verification.)

In this way, after the $(3n-1)$–th step the generation stage finishes and the *checking stage* starts. At this moment, the cell with label 1 contains 2^n copies of objects B_{3n}, C_{3n}, and D_{3n}, and there are 2^n cells with label 2, each of them containing a copy of y and $n-1$ copies of z, or a copy of y' and $n-1$ copies of z'. The objects z and z' in cells with label 2 will not evolve anymore, because the cell with label 1 contains no object c from now on, and the rules r_{31} and r_{32} can not be applied.

At the step $3n$, objects y or y' are traded for objects B_{3n}, C_{3n}, and D_{3n} by rules r_{48} and r_{49}. (Note that the rules $r_{33,i}$ and $r_{34,i}$ can not be used, because there is no object A_i in the cell with label 1 at this moment and henceforth. And

the cells with label 2 cannot separate any more.) Due to the maximality of the parallelism of using the rules, each cell with label 2 gets exactly one copy of each of B_{3n}, C_{3n}, and D_{3n}.

In the presence of C_{3n}, the objects T_i and T'_i, F_i and F'_i introduce the objects t_i and f_i, respectively. We have only one copy of C_{3n} available, for each one of t_i and f_i we need one step. So this phase needs n steps that is, this phase ends at step $4n$.

In parallel with the previous operations, the counters B_i and D_i increase their subscripts, until reaching the value $4n$ by the rules $r_{54,i}$ and $r_{55,i}$. Each cell with label 2 contains one copy of D_{4n} and 2^n copies of B_{4n}. Simultaneously, E_i increase its subscript in the cell with label 1.

At step $4n+1$, with the presence of B_{4n}, we start to check the values assumed by clauses for the truth-assignments from each cell with label 2 by the rules $r_{56,i,j}$ – $r_{63,i,j}$. Each membrane with label 2 contains nm objects $x_{i,j}$ and $\bar{x}_{i,j}$ or nm objects $x'_{i,j}$ and $\bar{x}'_{i,j}$, because each clause contains at most n literals, and we have m clauses. Note that each membrane with label 2 contains 2^n copies of B_{4n} and n objects t_i and f_i. At each step, n objects $x_{i,j}$ and $\bar{x}_{i,j}$, or n objects $x'_{i,j}$ and $\bar{x}'_{i,j}$ are checked. So it takes m steps. In parallel, D_i increases the subscript, until reaching the value $4n+m$ (at step $4n+m$) by the rule $r_{64,i}$.

By the rule $r_{65,i}$, in each cell with label 2, we check whether or not all clauses are satisfied by the corresponding truth-assignment. For each clause which is satisfied, we increase by one the subscript of D_i, hence the subscript reaches the value $4n+2m$ if and only if all clauses are satisfied.

The output stage starts at the $(4n+2m+1)$-th step.

- *Affirmative answer*: If one of the truth-assignments from a cell with label 2 has satisfied all clauses, then in that cell there is an object D_{4n+2m} as described above, which is sent to the cell with label 1 by the rule r_{66}. In the next step, the object `yes` leaves the system by the rule r_{67}, signaling the fact that the formula is satisfiable. In cell 1, the counter E_i increases its subscript by the rule $r_{41,i}$, until reaching the value $4n+2m+3$, but after that it will remain unchanged – it can leave the cell with label 1 only in the presence of p, but this object p was already moved to the environment at step $4n+2m+2$. The computation halts at step $4n+2m+2$.
- *Negative answer*: If the counter E_i reaches the subscript $4n+2m+3$ and the object p is still in the cell with label 1, then the object `no` can be moved to the environment by the rule r_{68}, signaling that the formula is not satisfiable. The computation finishes at step $4n+2m+3$.

6.2 Formal Verification

In this subsection, we prove that the family built above solves SAT problem in a polynomial time, according to Definition 1. First of all, the Definition 1 requires that the defined family is *consistent*, in the sense that all systems of the family

must be recognizer tissue P systems with cell separation. By the construction (type of rules and working alphabet) it is clear that it is a family of tissue P systems with cell separation. In order to show that all members in $\mathbf{\Pi}$ are recognizer systems it suffices to check that all the computations halt (this will be deduced from the polynomial bound), and that either an object `yes` or an object `no` is sent out exactly in the last step of the computation (this will be deduced from the soundness and completeness).

Polynomial uniformity of the family

We will show that the family $\mathbf{\Pi} = \{\Pi(\langle n, m\rangle) \mid n, m \in \mathbb{N}\}$ defined above is polynomially uniform by Turing machines. To this aim it will be proved that $\Pi(\langle n, m\rangle)$ is built in polynomial time with respect to the size parameter n and m of instances of SAT problem.

It is easy to check that the rules of a system $\Pi(\langle n, m\rangle)$ of the family are defined recursively from the values n and m. And the necessary resources to build an element of the family are of a polynomial order, as shown below:

- Size of the alphabet: $3n^2m + 4nm + 30n + 5m - 5 \in O(n^2m)$.
- Initial number of cells: $2 \in O(1)$.
- Initial number of objects: $nm + 10 \in O(nm)$.
- Number of rules: $3n^2m + 15nm + 36n + 3m - 12 \in O(n^2m)$.
- Maximal length of a rule: $6 \in O(1)$.

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

Polynomial bound of the family

For an instance of SAT problem $\varphi = M_1 \wedge \cdots \wedge M_m$, consisting of m clauses $M_i = y_{i,1} \vee \cdots \vee y_{i,l_i},\ 1 \leq i \leq m$, where $Var(\varphi) = \{x_1, \cdots, x_n\}$, $y_{i,k} \in \{x_j, \neg x_j \mid 1 \leq j \leq n\},\ 1 \leq i \leq m, 1 \leq k \leq l_i$, we recall the *size* mapping function $s(\varphi)$ and the encoding function $cod(\varphi)$: $s(\varphi) = \langle n, m\rangle$, and $cod(\varphi) = \{\{c_{i,j}x_{i,j} \mid x_i \in \{y_{j,k} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq n, 1 \leq j \leq m\}\} \cup \{\{c_{i,j}\bar{x}_{i,j} \mid \neg x_i \in \{y_{j,k} \mid 1 \leq k \leq l_i\}, 1 \leq i \leq n, 1 \leq j \leq m\}\}$. The pair (cod, s) is computable in polynomial time, $cod(\varphi)$ is an input multiset of the system $\Pi(s(\varphi))$.

In order to prove that the system $\Pi(s(\varphi))$ with input $cod(\varphi)$ is polynomially bounded, it suffices to find the moment in which the computation halts, or at least, an upper bound for it.

Proposition 4. *The family* $\mathbf{\Pi} = \{\Pi(\langle n, m\rangle) \mid n, m \in \mathbb{N}\}$ *is polynomially bounded with respect to* (SAT, cod, s).

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

Let $\varphi = M_1 \wedge \cdots \wedge M_m$ be an instance of the problem SAT. We shall study what happens during the computation of the system $\Pi(s(\varphi))$ with input $cod(\varphi)$ in order to find the halting step, or at least, an upper bound for it.

First, the generation stage has exactly $3n-1$ steps, where at steps $3k+2$ $(0 \leq k \leq n-1)$ the cells with label 2 are separated. In this way, we get 2^n cells with label 2, each of them contains one of the 2^n possible truth-assignments for the n variables.

After one more step, the objects B_{3n}, C_{3n}, and D_{3n} arrive at cells with label 2, and the checking stage starts. The object C_{3n} works for n steps introducing objects t_i or f_i into cells with label 2, until all object T_i, T_i', F_i and F_i' are consumed, at the step $4n$. From step $4n+1$, the objects B_{4n} start to work checking which clauses are satisfied by the truth-assignment from each cell with label 2. This checking takes m steps. When the subscript of D_i grows to $4n+m$ at step $4n+m$, the system starts to check whether or not all clauses are satisfied by the corresponding truth-assignment. It takes m steps, and the checking stage ends at step $4n+2m$.

The last one is the answer stage. The longest case is obtained when the answer is negative. In this case there are two steps where only the counter E_i is working. At the step $4n+2m+3$ the object $E_{4n+2m+3}$ works together with object p bringing `no` from the cell with label 1 into the environment.

Therefore, there exists a linear bound (with respect to n and m) 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 (SAT, cod, s), we shall prove that for a given instance φ of the problem SAT, the system $\Pi(s(\varphi))$ with input $cod(\varphi)$ sends out an object `yes` if and only if the answer to the problem for the considered instance φ is affirmative and the object `no` is sent out otherwise. In both cases the answer will be sent to the environment in the last step of the computation.

For the sake of simplicity in the notation, we consider the following two functions $\psi(\sigma_j(x_i))$ and $\gamma(\sigma_j(x_i))$. Let $\mathcal{F}$ be the set of all assignments of the variables $x_1, x_2, \cdots, x_n$. We order the set $\mathcal{S}$ in lexicographical order, that is, $\mathcal{F} = \{\sigma_1, \sigma_2, \cdots, \sigma_{2^n}\}$, where $\sigma_j(x_i) \in \{0,1\}$ $(1 \leq j \leq 2^n,\ 1 \leq i \leq n)$ is an assignment of variables. For $1 \leq j \leq 2^n$, $1 \leq i \leq n$, we define ψ as follows: if j is odd, then

$$\psi(\sigma_j(x_i)) = \begin{cases} T_i, & \text{if } \sigma_j(x_i) = 1, \\ F_i, & \text{if } \sigma_j(x_i) = 0; \end{cases}$$

if j is even, then

$$\psi(\sigma_j(x_i)) = \begin{cases} T_i', & \text{if } \sigma_j(x_i) = 1, \\ F_i', & \text{if } \sigma_j(x_i) = 0. \end{cases}$$

For each assignment $\sigma_j(x_i) \in \{0,1\}$, and for $i = 1, \ldots, n$, we define γ as follows:

$$\gamma(\sigma_j(x_i)) = \begin{cases} t_i, & \text{if } \sigma_j(x_i) = 1; \\ f_i, & \text{if } \sigma_j(x_i) = 0. \end{cases}$$

In this way, each assignment of variables σ_j is associated a multiset $\{\{\psi(\sigma_j(x_1)), \psi(\sigma_j(x_2)), \cdots, \psi(\sigma_j(x_n))\}\}$ and a multiset $\{\{\gamma(\sigma_j(x_1)), \gamma(\sigma_j(x_2)), \cdots, \gamma(\sigma_j(x_n))\}\}$.

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.

Proposition 5. *Let $\mathcal{C}$ be an arbitrary computation of the system, then at step $3k+2$ for all k such that $0 \leq k \leq n-2$, the cell with label 1 gets 2^{k+1} copies of object $c_{i,j}$, $2^{k+1}(k+1)$ copies of object c, and 2^{k+1} copies of object A_{k+2} from the environment. And after step $3n-3$, the cell with label 1 cannot get objects $c_{i,j}$, c, A_i any more.*

Proof. It is not difficult to find that in the set of all rules there are 6 types of rules related to object $c_{i,j}$, that is, rules $r_{2,i,j,k}$ – $r_{5,i,j,k}$, $r_{29,i,j}$ and $r_{30,i,j}$. The rules $r_{29,i,j}$ and $r_{30,i,j}$ are used to move $c_{i,j}$ from the cell with label 1 to cells with label 2 in exchange of $z_{i,j}$ or $z'_{i,j}$, which happens at steps $3k+3$ for all k such that $0 \leq k \leq n-2$. Anyway, these two rules do not bring object $c_{i,j}$ into the cell with label 1. So we need only to check how these 4 types of rules $r_{2,i,j,k}$ – $r_{5,i,j,k}$ work.

First, by induction on k, we prove that at step $3k+2$ ($0 \leq k \leq n-3$), the cell with label 1 gets 2^{k+1} copies of object $c_{i,j}$ and the cell with label 1 has exactly 2^{k+1} copies of $d_{i,j,k+1}$.

In the multiset $\mathcal{C}_0(1)$, there is one copy of each object $a_{i,j,1}$ ($1 \leq i \leq n$, $1 \leq j \leq m$). By application of rules $r_{2,i,j,1}$ and $r_{3,i,j,1}$, the cell with label 1 gets two 2 copies of $c_{i,j}$ and 2 copies $d_{i,j,1}$ at step 2.

Now suppose the result is true for $k < n-4$. We have, by the inductive hypothesis, at step $3k+2$, the cell with label 1 gets 2^{k+1} copies of object $c_{i,j}$ and the cell with label 1 has exactly 2^{k+1} copies of $d_{i,j,k+1}$. At step $3k+3$, among these 4 types of rules $r_{2,i,j,k+1}$ – $r_{5,i,j,k}$, only rule $r_{5,i,j,k+1}$ can be applied, 2^{k+1} copies of $d_{i,j,k+1}$ are traded for 2^{k+1} copies of $a_{i,j,k+2}$. At step $3(k+1)+1$, 2^{k+1} copies of $a_{i,j,k+2}$ are traded for 2^{k+1} copies of $b_{i,j,k+2}$ by the $r_{2,i,j,k+2}$. At step $3(k+1)+2$, by the rule $r_{3,i,j,k+2}$, the cell with label 1 gets 2^{k+2} copies of object $c_{i,j}$ and the cell with label 1 has exactly 2^{k+2} copies of $d_{i,j,k+2}$.

Based on the above result, specifically, we have, at step $3(n-3)+2$, the cell with label 1 has exactly 2^{n-2} copies of $d_{i,j,n-2}$. In the next 3 steps, the rules $r_{5,i,j,n-2}$, $r_{2,i,j,n-1}$, and $r_{4,i,j,n-1}$ are applied in order. At step $3(n-2)+2$, the cell with label 1 gets 2^{n-1} copies of $c_{i,j}$. Note that on object $d_{i,j,k}$ is brought into the cell with label 1 at step $3(n-2)+2$, and the group of rules $r_{5,i,j,k}$, $r_{2,i,j,k}$, and $r_{4,i,j,k}$ cannot be used any more. Therefore the cell with label 1 will not get object $c_{i,j}$ any more, and the result holds.

For the cases of objects c and A_i, the results can be proved similarly. We omit them here.

Proposition 6. *Let $\mathcal{C}$ be an arbitrary computation of the system, then*

1. *For each assignment $\sigma_j(x_i)$, $i = 1, 2, \cdots, n$, there exists only one cell with label 2 in $\mathcal{C}_{3n-1}$ that contains multiset $\{\{\psi(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$.*

2. *There exist exactly* 2^n *cells with label* 2 *in configuration* $\mathcal{C}_k$ *(*$k \geq 3n-1$*). Particularly, in configuration* $\mathcal{C}_{3n-1}$*, each cell with label* 2 *contains a multiset* $\{\{y\}\} \cup \{\{z_{i,j}, x_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z_{i,j}, \bar{x}_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$ *or a multiset* $\{\{y'\}\} \cup \{\{z'_{i,j}, x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, \bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$.

Proof. We prove the result by induction.

In the configuration $\mathcal{C}_0$, there is only one cell with label 2, which has multiset $\{\{c_{ij}, x_{i,j}, \bar{x}_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq m\}\} \cup \{\{A_1\}\}$. The rules $r_{19,i,j} - r_{22,i,j}$ and $r_{23,1}$ can be applied. At step 1, the cell with label 2 has multiset $\{\{z_{i,j}, x_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z_{i,j}, \bar{x}_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, \bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\} \cup \{\{z, z', T_1, F'_1, y, y', s\}\}$. At step 2, with the appearance of object s, the separation rule r_1 is used to separate cell with label 2, object s is consumed, and the multiset $\{\{z_{i,j}, x_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z_{i,j}, \bar{x}_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\} \cup \{\{z, T_1, y\}\}$ are placed in one new cell with label 2, the multiset $\{\{z'_{i,j}, x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, \bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\} \cup \{\{z', F'_1, y'\}\}$ are placed in another new cell with label 2. We take the cell with label 2 where $\psi(\sigma_j(x_1))$ appears.

By Proposition 5, at step 2, the cell with label 1 has two copies of $c_{i,j}$, two copies of c, two copies of A_2. So, at step 3, the rules $r_{29,i,j}$, $r_{30,i,j}$, $r_{33,2}$ and $r_{34,2}$ can be applied. Due to the maximality of the parallelism of using rules, each cell with label 2 gets exactly one copy of $c_{i,j}$, one copy of c, and one copy of A_2 from the cell with label 1. Object $c_{i,j}$ and c are used for duplication, and A_2 is used to assign truth-values to the valuable x_2. In this way, the next cycle of duplication-separation can continue.

In general, after step $3k+2$ $(0 \leq k \leq n-1)$ (that is, the second step in the $(k+1)$-th cycle of duplication-separation), we take the cell with label 2 where $\{\{\psi(\sigma_j(x_1)), \cdots, \psi(\sigma_j(x_{k+1}))\}\}$ appears. In this way, at step $3n-1$, there exists exactly one cell with label 2 whose multiset is $\{\{\psi(\sigma_j(x_1)), \cdots, \psi(\sigma_j(x_n))\}\}$. (Note the difference of the rule r_{24} and the rules $r_{23,i}$. The rule r_{24} does not bring objects z and z' into cells with label 2 from the environment, and this rule is used at step $3n-2$. So the object z does not appear in the multiset of cell 2 that corresponds to the assignment σ_j.)

From the above proof, it is easy to see that the multiset $\{\{\psi(\sigma_j(x_1)), \cdots, \psi(\sigma_j(x_n))\}\}$ appears only in the corresponding cell with label 2.

In every cycle of duplication-separation, the number of cells with label 2 is doubled. In the $3n-1$ steps, there are n cycles. So there exist exactly 2^n cells with label 2 in configuration $\mathcal{C}_{3n-1}$; and from now on, cells with label 2 will not separate anymore.

In the last cycle of duplication-separation, at step $3n-2$, each of the 2^{n-1} cells with label 2 contains one copy of y and one copy of y' by the rule r_{24}; at step $3n-1$, the 2^{n-1} cells with label 2 are separated by the rule r_1, each of the 2^n new cells gets one copy of object y or one copy of object y'.

Proposition 7. *Let* $\mathcal{C}$ *be an arbitrary computation of the system, then* $\mathcal{C}_{3n-1}(1) = \{\{B_{3n}^{2^n}, C_{3n}^{2^n}, D_{3n}^{2^n}, E_{3n}, p, \texttt{yes}, \texttt{no}\}\}$.

Proof. In order to prove $\mathcal{C}_{3n-1}(1) = \{\{B_{3n}^{2^n}, C_{3n}^{2^n}, D_{3n}^{2^n}, E_{3n}, p, \texttt{yes}, \texttt{no}\}\}$, we will check how all the rules related to the cell 1 work in the first $3n-1$ steps.

- Checking the rules $r_{2,i,j,k} - r_{18,i}$.
  From the proofs of Propositions 5 and 6, we can find that after step $3n-3$, there are no objects $a_{i,j,k}, b_{i,j,k}, c_{i,j}, d_{i,j,k}, a_{1,i}, b_{1,i}, c, d_{1,i}, e_i, a_{2,i}, b_{2,i}, d_{2,i}, g_i, h_i, l_i, A_i$ in the cell with label 1, and the rules $r_{2,i,j,k} - r_{18,i}$ will not bring any more objects into the cell with label 1.

- Checking the rules $r_{35,i} - r_{40,i}$.
  In the first $2n-1$ steps of the computation, by the rules $r_{35,i}$, $r_{37,i}$, and $r_{39,i}$, the subscripts of B_1, C_1, and D_1 grow to $2n$. In the next n steps, by the rules $r_{36,i}$, $r_{38,i}$, and $r_{40,i}$, the subscripts of B_{2n}, C_{2n}, and D_{2n} grow to $3n$, and at every step, the numbers of objects of each type B_i, C_i, and D_i are doubled. So the cell with label 1 has 2^n copies of B_{3n}, 2^n copies of C_{3n}, and 2^n copies of D_{3n} at the step $3n-1$.

- Checking the rule $r_{41,i}$.
  By the rule $r_{41,i}$, the subscript of E_1 grow to $3n$ in the first $3n-1$ steps of the computation. So the cell 1 has the object E_{3n} at step $3n-1$.

- Checking the group of rules $r_{29,i,j} - r_{34,i,j}$ and the group of rules $r_{42,i,j} - r_{47}$.
  In the first $3n-3$ steps, the cell with label 1 has communication with cells with label 2 getting objects $z_{i,j}$, $z'_{i,j}$, z, z', y, y' from cells with label 2, by the rules $r_{29,i,j} - r_{34,i,j}$. In the next step after the objects $z_{i,j}$, $z'_{i,j}$, z, z', y, y' reach the cell 1, they are sent to the environment by the rules $r_{42,i,j} - r_{47}$.

- Checking the group of rules $r_{48} - r_{49}$.
  At the step $3n-1$, the subscript of objects B_i, C_i and D_i grow to $3n$. The rules $r_{48} - r_{49}$ can be applied at step $3n$. But, in the first $3n-1$ steps of the computation, they cannot be applied.

- Checking the rules $r_{66} - r_{68}$.
  In the first steps $3n-1$, there are no object D_{4n+2m} appearing in cells with label 2, and no object $E_{4n+2m+3}$ appearing in the cell with label 1. The rules $r_{66} - r_{68}$ cannot be applied in the first $3n-1$ steps of the computation, so the cell with label 1 has objects $\texttt{yes}$, $\texttt{no}$ and p at the step $3n-1$.

  Therefore, $\mathcal{C}_{3n-1}(1) = \{\{B_{3n}^{2^n}, C_{3n}^{2^n}, D_{3n}^{2^n}, E_{3n}, p, \texttt{yes}, \texttt{no}\}\}$.

Proposition 8. *Let $\mathcal{C}$ be an arbitrary computation of the system, then*

1. $\mathcal{C}_{3n}(1) = \{\{y^{2^{n-1}}, (y')^{2^{n-1}}, E_{3n+1}, p, \texttt{yes}, \texttt{no}\}\}$;
2. *For each assignment σ_j there exists only one cell with label 2 in $\mathcal{C}_{3n}$ that contains multiset $\{\{\psi(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$. In configuration $\mathcal{C}_{3n}$, each cell with label 2 contains a multiset $\{\{B_{3n}, C_{3n}, D_{3n}\}\} \cup \{\{z_{i,j}, x_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z_{i,j}, \bar{x}_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$ or multiset $\{\{B_{3n}, C_{3n}, D_{3n}\}\} \cup \{\{z'_{i,j}, x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, \bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$.*

Proof. The multiset $\mathcal{C}_{3n}(1)$ is obtained from $\mathcal{C}_{3n-1}(1)$ by the application of the rules $r_{41,i}$, r_{48} and r_{49}. The object E_{3n} in $\mathcal{C}_{3n-1}(1)$ increases its subscript by one by the rule $r_{41,i}$, so one copy of E_{3n+1} appears in $\mathcal{C}_{3n}(1)$. By Proposition 7, at step $3n-1$, the cell with label 1 has 2^n copies of B_{3n}, 2^n copies of C_{3n}, and 2^n copies of D_{3n}. By Proposition 6, at step $3n-1$, there exist exactly 2^n cells with label 2, each of them contains one copy of object y or one copy of object y', and the number of cells with label 2 containing object y (resp. y') is 2^{n-1} (resp. 2^{n-1}). At step $3n$, the rules r_{48} and r_{49} can be applied, 2^n copies of objects $B_{3n}C_{3n}D_{3n}$ in the cell with label 1 are traded for y or y' from the cells with label 2. Due to the maximality of the parallelism of using rules, each cell with label 2 gets one copy of objects $B_{3n}C_{3n}D_{3n}$, and the cell with label 1 gets 2^{n-1} copies of object y and 2^{n-1} copies of object y'. Therefore, $\mathcal{C}_{3n}(1) = \{\{y^{2^{n-1}}, (y')^{2^{n-1}}, E_{3n+1}, p, \texttt{yes}, \texttt{no}\}\}$; and for each assignment σ_j there exists only one cell with label 2 in $\mathcal{C}_{3n}$ that contains multiset $\{\{\psi(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$. By Proposition 6 and the above proof, it is not difficult to see that in configuration $\mathcal{C}_{3n}$, each cell with label 2 contains a multiset $\{\{B_{3n}, C_{3n}, D_{3n}\}\} \cup \{\{z_{i,j}, x_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z_{i,j}, \bar{x}_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$ or a multiset $\{\{B_{3n}, C_{3n}, D_{3n}\}\} \cup \{\{z'_{i,j}, x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, \bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$.

Proposition 9. $\mathcal{C}_{4n}(1) = \{\{E_{4n+1}, p, \texttt{yes}, \texttt{no}\}\}$ *holds.*

Proof. By the rules r_{44} and r_{45}, at step $3n+1$, the objects y and y' are moved to the environment. Henceforth the cell with label 1 will not get any more y or y', because the objects y or y' are traded into cells with label 2 from the environment against the object A_n, and no A_i will appear in cells with label 2 after step $3n-2$. By the rule $r_{41,i}$, the subscript of E_{3n+1} grows to E_{4n+1}. These are all the operations related to the cell with label 1 from step $3n+1$ to step $4n$. Therefore, $\mathcal{C}_{4n}(1) = \{\{E_{4n+1}, p, \texttt{yes}, \texttt{no}\}\}$ holds.

Proposition 10. *For each assignment σ_j, there exists only one cell with label 2 in $\mathcal{C}_{4n}$ that contains multiset $\{\{\gamma(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$. In configuration $\mathcal{C}_{4n}$, each cell with label 2 contains a multiset $\{\{B_{4n}^{2^n}, C_{3n}, D_{4n}\}\} \cup \{\{z_{i,j}, x_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z_{i,j}, \bar{x}_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$ or a multiset $\{\{B_{4n}^{2^n}, C_{3n}, D_{4n}\}\} \cup \{\{z'_{i,j}, x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, \bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$.*

Proof. Based on Proposition 8, we prove Proposition 10 holds.

From step $3n+1$ to step $4n$, the objects in multiset $\{\{z_{i,j}, x_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z_{i,j}, \bar{x}_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$ and multiset $\{\{z'_{i,j}, x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, \bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$ keep unchanged because no rules can be applied to them.

By the rules $r_{54,i}$ and $r_{55,i}$, the subscripts of objects B_{3n} and D_{3n} in $\mathcal{C}_{3n}$ increase, until reaching the value $4n$ at step $4n$. At each step from step $3n+1$ to step at step $4n$, the number of object B_i is doubled by the rule $r_{54,i}$, so in configuration $\mathcal{C}_{4n}$, there are 2^n copies of B_i.

For an assignment σ_j, we consider the only cell with label 2 in $\mathcal{C}_{3n}$ that contains multiset $\{\{\gamma(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$. From the definition of functions ψ and γ, we can see that T_i, T'_i and t_i correspond to the value *true* which the valuable x_i

assumes; F_i, F_i' and f_i correspond to the value $false$ which the valuable x_i assumes. Both multisets $\{\{\psi(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$ and $\{\{\gamma(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$ are associated with the same assignment σ_j. By the rules $r_{50,i}$ – $r_{53,i}$, with the object C_{3n}, the objects T_i or T_i' introduce the objects t_i, and the objects F_i or F_i' introduce the objects f_i. Because there is only one copy of C_{3n}, it takes n steps to introduce all the objects in multiset $\{\{\gamma(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$.

Therefore Proposition 10 holds.

Proposition 11. $\mathcal{C}_{4n+m}(1) = \{\{E_{4n+m+1}, p, \texttt{yes}, \texttt{no}\}\}$ *holds.*

Proof. By the rule $r_{41,i}$, the subscript of object E_{4n+1} in $\mathcal{C}_{4n}(1)$ grows to the value $4n + m + 1$ at step $4n + m$. The objects $p, \texttt{yes}, \texttt{no}$ keeps unchanged. Therefore, Proposition 11 holds.

Proposition 12. *For each assignment σ_j, there exists only one cell with label* 2 *in $\mathcal{C}_{4n+m}$ that contains multiset $(\cup_{i=1}^{n}\{\{r_{j_1} r_{j_2} \cdots r_{j_k} \mid \gamma(\sigma_j(x_i)) = t_i$, and $x_{i,j_l} \in cod(\varphi), l = 1, 2, \cdots, k, 1 \leq j_1 < j_2 < \cdots < j_k \leq n\}\}) \cup (\cup_{i=1}^{n}\{\{r_{j_1} r_{j_2} \cdots r_{j_k} \mid \gamma(\sigma_j(x_i)) = f_i$, and $\bar{x}_{i,j_l} \in cod(\varphi), l = 1, 2, \cdots, k, 1 \leq j_1 < j_2 < \cdots < j_k \leq n\}\})$. In configuration $\mathcal{C}_{4n+m}$, each cell with label* 2 *contains a multiset $\{\{B_{4n}^{2^n}, C_{3n}, D_{4n+m}\}\}$.*

Proof. By Proposition 10, for each assignment σ_j, there exists only one cell with label 2 in $\mathcal{C}_{4n}$ that contains multiset $\{\{\gamma(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$; and each cell with label 2 contains a multiset $\{\{B_{4n}, C_{3n}, D_{4n}\}\} \cup \{\{z_{i,j}, x_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z_{i,j}, \bar{x}_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$ or a multiset $\{\{B_{4n}, C_{3n}, D_{4n}\}\} \cup \{\{z'_{i,j}, x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{z'_{i,j}, \bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$. In the following, we consider this unique cell with label 2 that contains multiset $\{\{\gamma(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$.

The objects C_{3n} keep unchanged, and the subscript of D_{4n} reaches $4n + m$ at step $4n + m$ by the rule $r_{64,i}$.

With the presence of B_{4n} in $\mathcal{C}_{4n}$ (not appearing in $\mathcal{C}_i$ $(i < 4n)$), the rules $r_{56,i,j}$ – $r_{63,i,j}$ can be applied. We start to check which clauses are satisfied. If $\sigma_j((x_i)) = t_i$ and $x_{i,j} \in cod(\varphi)$, then rule $r_{56,i,j}$ or $r_{58,i,j}$ is applied, and an object r_j is introduced into the corresponding cell with label 2. If $\sigma_j((x_i)) = t_i$ and $\bar{x}_{i,j} \in cod(\varphi)$, then rule $r_{57,i,j}$ or $r_{59,i,j}$ is applied, and the object $\bar{x}_{i,j}$ or $\bar{x}'_{i,j}$ is removed from the corresponding cell with label 2. Similarly, if $\sigma_j((x_i)) = f_i$ and $\bar{x}_{i,j} \in cod(\varphi)$, then rule $r_{60,i,j}$ or $r_{62,i,j}$ is applied, and an object r_j is introduced into the corresponding cell with label 2. If $\sigma_j((x_i)) = f_i$ and $x_{i,j} \in cod(\varphi)$, then rule $r_{61,i,j}$ or $r_{63,i,j}$ is applied, and the object $x_{i,j}$ or $x'_{i,j}$ is removed from the corresponding cell with label 2. The sizes of both $cod(\varphi)$ and $\{\{x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{\bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$ are nm, and each cell with label 2 contains multiset $cod(\varphi)$ or $\{\{x'_{i,j} \mid x_{i,j} \in cod(\varphi)\}\} \cup \{\{\bar{x}'_{i,j} \mid \bar{x}_{i,j} \in cod(\varphi)\}\}$. We have 2^n copies of B_{4n}, n objects t_i and f_i from the multiset $\{\{\gamma(\sigma_j(x_i)) \mid i = 1, 2, \cdots, n\}\}$, so it takes m steps to check which clauses are satisfied. In total, all the introduced objects r_i form the multiset $(\cup_{i=1}^{n}\{\{r_{j_1} r_{j_2} \cdots r_{j_k} \mid \gamma(\sigma_j(x_i)) = t_i, \text{and } x_{i,j_l} \in cod(\varphi), l = 1, 2, \cdots, k, 1 \leq j_1 < j_2 < \cdots < j_k \leq n\}\}) \cup (\cup_{i=1}^{n}\{\{r_{j_1} r_{j_2} \cdots r_{j_k} \mid \gamma(\sigma_j(x_i)) = f_i, \text{and } \bar{x}_{i,j_l} \in cod(\varphi), l = 1, 2, \cdots, k, 1 \leq j_1 < j_2 < \cdots < j_k \leq n\}\})$.

Proposition 13. $\mathcal{C}_{4n+2m}(1) = \{\{E_{4n+2m+1}, p, \texttt{yes}, \texttt{no}\}\}$ *holds.*

Proof. By the rule $r_{41,i}$, the subscript of object E_{4n+m+1} in $\mathcal{C}_{4n+m}(1)$ grows to the value $4n + 2m + 1$ at step $4n + 2m$. The objects $p, \texttt{yes}, \texttt{no}$ keeps unchanged. Therefore, Proposition 13 holds.

Proposition 14. *Let $\mathcal{C}$ be an arbitrary computation of the system, then*

- *If σ_j is an assignment that does not satisfy the formula φ, then there exists only one cell with label 2 in $\mathcal{C}_{4n+2m}$ associated with σ_j, and whose associated multiset contains an object $D_{4n+m+\alpha}$, where $0 \leq \alpha < m$ such that the clauses $M_1, \cdots, M_\alpha$ are satisfied by the assignment f, but $M_{\alpha+1}$ is not satisfied by the assignment σ_j.*
- *If σ_j is an assignment that satisfies the formula φ, then there exists only one cell 2 in $\mathcal{C}_{4n+2m}$ associated with σ_j, and whose associated multiset contains one copy of object D_{4n+2m}.*

Proof. From the configuration $\mathcal{C}_{4n+m}$, we start to check whether or not all clauses are satisfied by the corresponding assignment. Such checking is simultaneous in all 2^n cells with label 2.

Let us consider an assignment σ_j. By Proposition 12, with the presence of object D_{4n+m}, the rule $r_{65,i}$ can be applied. The clauses are checked in the order from M_1 to M_m. For each clause which is satisfied (that is, the corresponding object r_i appears), we increase by one the subscript of D_i, hence the subscript of D_i reaches the value $4n + 2m$ if and only if all clauses are satisfied. If the clauses $M_1, \cdots, M_\alpha$ $(0 \leq \alpha < m)$ are satisfied, but $M_{\alpha+1}$ is not satisfied (that is, $r_1, \cdots, r_\alpha$ appear, but $r_{\alpha+1}$ does not appear), then the subscript of D_i can only reach the value $4n + m + \alpha$.

Therefore, Proposition 14 holds.

Proposition 15. *Let $\mathcal{C}$ be an arbitrary computation of the system, Let us suppose that there exists an assignment that satisfies the formula φ. Then*
(a) $\mathcal{C}_{4n+2m+1}(1) = \{\{E_{4n+2m+2}, D^{\beta}_{4n+2m}, p, \texttt{yes}, \texttt{no}\}\}$,
(b) $\mathcal{C}_{4n+2m+2}(1) = \{\{E_{4n+2m+3}, D^{\beta-1}_{4n+2m}, \texttt{no}\}\}$,
where β is the number of assignments that satisfy the formula φ. Furthermore, the object yes *appears in $\mathcal{C}_{4n+2m+2}(0)$.*

Proof. The configuration of item (a) is obtained by the application of rules $r_{41,i}$ and r_{66} to the previous configuration $\mathcal{C}_{4n+2m}$. By the rule $r_{41,i}$, the object $E_{4n+2m+1}$ in $\mathcal{C}_{4n+2m}(1)$ grows by one its subscript at step $4n+2m+1$. By Proposition 14, for each assignment that satisfies the formula φ, there exists exactly one associated cell with label 2 in $\mathcal{C}_{4n+2m}$ whose multiset contains an object D_{4n+2m}. The object D_{4n+2m} is moved to the cell with label 1 by the rule r_{66}. If there are β assignments that satisfy the formula φ, then the cell 1 gets β copies of object D_{4n+2m}.

The configuration of item (b) is obtained by the application of rules $r_{41,i}$ and r_{67} to the previous configuration $\mathcal{C}_{4n+2m+1}(1)$. By the rule $r_{41,i}$, the object $E_{4n+2m+2}$

in $\mathcal{C}_{4n+2m+1}(1)$ grows by one its subscript at step $4n + 2m + 2$. By the rule r_{67}, the object `yes` together with objects D_{4n+2m} and p leaves the system into the environment, signaling the formula φ is satisfiable. The one copy of object p is consumed by the rule r_{67}, so the rule r_{68} cannot be applied. The object `no` cannot exit into the environment.

Proposition 16. *Let $\mathcal{C}$ be an arbitrary computation of the system, Let us suppose that there does not exist any assignment that satisfies the formula φ. Then*
(a) $\mathcal{C}_{4n+2m+1}(1) = \{\{E_{4n+2m+2}, p, \texttt{yes}, \texttt{no}\}\}$,
(b) $\mathcal{C}_{4n+2m+2}(1) = \{\{E_{4n+2m+3}, p, \texttt{yes}, \texttt{no}\}\}$,
(c) $\mathcal{C}_{4n+2m+3}(1) = \{\{\texttt{yes}\}\}$.
Furthermore, the object `no` *appears in* $\mathcal{C}_{4n+2m+3}(0)$.

Proof. If there does not exist any assignment that satisfies the formula φ, by Proposition 14, all cells with label 2 do not contain object D_{4n+2m}. Of course, the cell with label 1 cannot get object D_{4n+2m}.

The configurations of items (a) an (b) are obtained by the application of rules $r_{41,i}$ to the previous configuration $\mathcal{C}_{4n+2m}$.

The configuration of item (c) is obtained by the application of rules r_{68} to the previous configuration.

6.3 Main Results

The system constructed to solution of SAT in Section 6 has communication rules with length at most 6. From the discussion in the previous sections and according to the definition of solvability given in Section 4, we have the following result:

Theorem 2. $SAT \in \mathbf{PMC}_{TSC(6)}$.

Corollary 1. $\mathbf{NP} \cup \mathbf{co} - \mathbf{NP} \subseteq \mathbf{PMC}_{TS(6)}$.

Proof. It suffices to make the following observations: the SAT problem is **NP**-complete, SAT$\in \mathbf{PMC}_{TS(6)}$ and this complexity class is closed under polynomial-time reduction and under complement.

7 Discussion

The efficiency of cell-like P systems for solving **NP**-complete problems has been widely studied. The space-time tradeoff method is used to efficiently solve **NP**-complete problems in the framework of cell-like P systems. Membrane division, membrane creation, and membrane separation are three efficient ways to obtain exponential workspace in polynomial time. Membrane division is introduced into tissue P systems, and a linear time solution for SAT problem by tissue P systems with cell division is given [22]. In this research, membrane separation is introduced into tissue P systems, and a polynomial time solution for SAT problem by tissue

P systems with cell separation and communication rules of lengths at most 6 is presented. We also prove that tissue P systems with cell separation and communication rules of length 1 can only solve tractable problems. Hence, in the framework of recognizer tissue P systems with cell separation, the lengths of the communication rules provide a borderline between efficiency and non-efficiency. Specifically, a frontier is there when we pass from length 1 to length 6. The role of the lengths of communication rules is worth further investigation. That is, what happens if we consider tissue P systems with communication rules of length k, $k \in \{2, 3, 4, 5\}$?

In the framework of tissue P systems, when cell division is used to generate exponential workspace in polynomial time, there is an advantage: all the other objects in the cell are duplicated except the object that activate the cell division operation. But both cell creation and cell separation have no such duplication function. In this sense, the solution for SAT problem presented in this paper gives some hint for answering the following open problem: how to efficiently solve **NP**-complete problems by tissue P systems with cell creation.

Although SAT problem is **NP**-complete (the other **NP** problems can be reduced to SAT problem in polynomial-time), we would like to stress that up to now there does not exist a methodology to compute the reduction process by P systems. The solution to SAT problem by tissue P systems with cell separation can be used as a scheme for designing solutions to other **NP**–complete problems such as the *vertex–cover* problem, the *clique* problem, the *Hamiltonian path* problem, etc.

Recently, a new kind of P system model, called spiking neural P systems, was introduced [7], which has neural-like architectures. It was proved that spiking neural P systems are Turing complete [7]. About the efficiency of spiking neural P systems to solve computationally hard problems, there is an interesting result: an SN P system of polynomial size cannot solve in a deterministic way in a polynomial time an **NP**-complete problem (unless **P=NP**) [11]. Hence, under the assumption that $\mathbf{P} \neq \mathbf{NP}$, efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance the efficiency. One of possible features is some ways to exponentially grow the workspace during the computation. Cell division, cell creation and cell separation are candidates to be introduced into spiking neural P systems for exponential workspace generation. Although the architectures of spiking neural P systems are similar to the architectures of tissue P systems, it is not a trivial work to introduce cell division, cell creation and cell separation into spiking neural P systems and give efficient solutions to **NP**-complete problems by new variants of spiking neural P systems.

In general, it is expected that the research of efficiency and complexity on P systems can provide insight on unconventional parallel computing models, and also help us in clarifying the relations between classic complexity classes.

Acknowledgements

The work of L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826, 60703047, and 60533010), Program for

New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180). The work of M.J. Pérez-Jiménez was supported by Project TIN2006-13452 of the Ministerio de Educación y Ciencia of Spain and Project of Excellence with *Investigador de Reconocida Valía*, from Junta de Andalucía, grant P08 – TIC 04200.

References

1. Alhazov, A., Freund, R. and Oswald, M. Tissue P Systems with Antiport Rules ans Small Numbers of Symbols and Cells. *Lecture Notes in Computer Science* **3572**, (2005), 100–111.
2. Bernardini, F. and Gheorghe, M. Cell Communication in Tissue P Systems and Cell Division in Population P Systems. *Soft Computing* **9**, 9, (2005), 640–649.
3. Freund, R., Păun, Gh. and Pérez-Jiménez, M.J. Tissue P Systems with channel states. *Theoretical Computer Science* **330**, (2005), 101–116.
4. Frisco, P. and Hoogeboom, H.J. Simulating counter automata by P systems with symport/antiport. *Lecture Notes in Computer Science* **2597**, (2003), 288–301.
5. Garey, M.R. and Johnson, D.S. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, (1979).
6. Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J. and Romero-Campero, F.J. A linear solution for QSAT with Membrane Creation. *Lecture Notes in Computer Science* **3850**, (2006), 241–252.
7. Ionescu, M., Păun, Gh. and Yokomori, T. Spiking neural P systems, *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
8. Ito, M., Martín Vide, C., and Păun, Gh. A characterization of Parikn sets of ET0L laguages in terms of P systems. In M. Ito, Gh. Păun, S. Yu (eds.) *Words, Semigroups and Transducers*, World Scientific, Singapore, 2001, 239-254.
9. Krishna, S.N., Lakshmanan K. and Rama, R. Tissue P Systems with Contextual and Rewriting Rules. *Lecture Notes in Computer Science* **2597**, (2003), 339–351.
10. Lakshmanan K. and Rama, R. On the Power of Tissue P Systems with Insertion and Deletion Rules. In A. Alhazov, C. Martín-Vide and Gh. Păun (eds.) *Preproceedings of the Workshop on Membrane Computing*, Tarragona, Report RGML 28/03, (2003), 304–318.
11. Leporati, A., Zandron, C., Ferretti, C., Mauri, G. On the computational power of spiking neural P systems. *International Journal of Unconventional Computing*, 2007, in press.
12. Maass, W., Bishop, C. eds.: *Pulsed Neural Networks*, MIT Press, Cambridge, (1999).
13. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez and F.J. Romero–Campero. On the power of dissolution in P systems with active membranes. *Lecture Notes in Computer Science* 3850 (2006), 224–240.
14. Martín Vide, C. Pazos, J. Păun, Gh. and Rodríguez Patón, A. A New Class of Symbolic Abstract Neural Nets: Tissue P Systems. *Lecture Notes in Computer Science* **2387**, (2002), 290–299.
15. Martín Vide, C. Pazos, J. Păun, Gh. and Rodríguez Patón, A. Tissue P systems. *Theoretical Computer Science*, **296**, (2003), 295–326.

16. Pan, L. and Ishdorj, T.-O. P systems with active membranes and separation rules. *Journal of Universal Computer Science*, **10**, 5, (2004), 630–649.
17. Păun, Gh. Computing with membranes. *Journal of Computer and System Sciences*, **61**, 1, (2000), 108–143.
18. Păun, Gh. Attacking **NP**-complete problems. In *Unconventional Models of Computation, UMC'2K* (I. Antoniou, C. Calude, M. J. Dinneen, eds.), Springer-Verlag, 2000, 94-115.
19. Păun, Gh. *Membrane Computing. An Introduction*. Springer–Verlag, Berlin, (2002).
20. Păun, A. and Păun, Gh. The power of communication: P systems with symport/antiport. *New Generation Computing*, **20**, 3, (2002), 295–305.
21. Păun, Gh. and Pérez-Jiménez, M.J. Recent computing models inspired from biology: DNA and membrane computing. *Theoria*, **18**, 46, (2003), 72–84.
22. Păun, Gh., Pérez-Jiménez, M.J. and Riscos-Núñez, A. Tissue P System with cell division. *In. J. of Computers, communications & control*, **3**, 3, (2008), 295–303.
23. Gh. Păun, Y. Sakakibara, T. Yokomori: P systems on graphs of restricted forms. *Publicationes Mathematicae Debrecen*, 60 (2002), 635–660.
24. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F. The P versus NP problem through cellular computing with membranes. *Lecture Notes in Computer Science* **2950**, (2004), 338–352.
25. Pérez-Jiménez, M.J., Romero-Jiménez, A. and Sancho-Caparrini, F. A polynomial complexity class in P systems using membrane division. *Journal of Automata, Languages and Combinatorics*, **11**, 4, (2006), 423-434.
26. Prakash, V.J. On the Power of Tissue P Systems Working in the Maximal-One Mode. In A. Alhazov, C. Martín-Vide and Gh. Păun (eds.). *Preproceedings of the Workshop on Membrane Computing*, Tarragona, Report RGML 28/03, (2003), 356–364.
27. Zandron, C., Ferretti, C. and Mauri, G. Solving NP-Complete Problems Using P Systems with Active Membranes. In I. Antoniou, C.S. Calude and M.J. Dinneen (eds.). *Unconventional Models of Computation, UMC'2K*, Springer-Verlag, (2000), 289–301.
28. ISI web page `http://esi-topics.com/erf/october2003.html`
29. P systems web page `http://ppage.psystems.eu/`

Some Open Problems Collected During 7th BWMC

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es

Summary. A few open problems and research topics collected during the 7th Brainstorming Week on Membrane Computing are briefly presented; further details can be found in the papers included in the volume.

1 Introduction

This is a short list of open problems and research topics which I have compiled during the Seventh Brainstorming Week on Membrane Computing (BWMC), Sevilla, 2-6 February 2009. Different from the previous years, when such lists (much more comprehensive) were circulated before the meeting, this time the problems were collected during and inspired by the presentations in Sevilla. Most of the problems directly refer to these presentations, hence for further details, in several cases for (preliminary) solutions, the reader should consult the present volume. This list is only a way to call attention to the respective ideas, not necessarily a presentation of the state-of-the-art of the research related to these problems.

Of course, the reader is assumed familiar with membrane computing, so that no preliminaries are provided. As usual, for basic references, the book [17], the handbook [21], and the membrane computing website [25] are suggested.

The problems are identified by letters, with no significance assigned to the ordering.

2 Open Problems

A. The following problem was considered in [5] and then in [3], as an extension to P systems of a problem well known for cellular automata (where it is called the

"firing squad synchronization problem", FSSP for short): Find a class of cell-like P systems where all membranes except the skin one may contain given objects, input some objects in the skin region, and aim to reach a configuration where each region contains a designated object F, all regions get this object at the same time, and, moreover, the system stops at that step. (A more technical presentation of the problem can be found in the cited papers.)

I recall from [3] some lines from the end of the Introduction: "The synchronization problem as defined above was studied in [5] for two classes of P systems: transitional P systems and P systems with priorities and polarizations. In the first case, a non-deterministic solution to FSSP was presented and for the second case a deterministic solution was found. These solutions need a time $3h$ and $4n + 2h$ respectively, where n is the number of membranes of a P system and h is the depth of the membrane tree.

In this article we significantly improve the previous results in the non-deterministic case. In the deterministic case, another type of P system was considered and this permitted to improve the parameters. The new algorithms synchronize the corresponding P systems in $2h + 3$ and $3h + 3$ steps, respectively."

Of course, these results depends on the type of P systems used. In particular, in both papers one makes use of a powerful communication command, of a broadcasting type: in! target indications can be associated with objects, with the meaning that the objects marked with in! are sent to ALL membranes placed inside the membrane where the rule containing the command in! was used. This is a very powerful feature, somewhat related to the considered issue itself, so that the natural question is whether or not one can get rid of in!. This will probably imply some cost, for instance, in terms of the time needed for synchronization, or – as A. Alhazov pointed out during BWMC – in the knowledge we assume to have (inside the system) about the starting systems (degree, depth, etc.).

A related topic is to consider synchronization issues for other classes of P systems: symport/antiport, active membranes, maybe also for spiking neural P systems (where the object F should be replaced with a specified number of spikes).

Actually, because of its interest (and presumably, of its difficulty, at least to define it), this can be formulated as a separate problem:

B. Define and investigate synchronization problems for spiking neural P systems (SN P systems).

C. Synchronization can suggest another problem, which has appeared from time to time in various contexts (for instance, when constructing SN P systems for simulating Boolean circuits, see [11]): reset a system to its initial configuration after completing a specified task. As in the case of synchronization, we need to consider various classes of P systems and, if the resetting is possible for them, we have to look for the time of doing this.

A possible way to address this problem is the following one: let us choose a class of P systems which halts the computation by sending a specified object to the environment (this is the case with the systems used in solving computationally hard

problems in a feasible time; those systems are in general with active membranes, membrane creation, etc.), AND for which the synchronization problem can be solved; when a system halts and produces an object, say, T, we use this object in order to start the synchronization phase; when the synchronization is completed, the object F introduced in each membrane is used for restoring the contents of that membrane as in the beginning of the computation. Of course, there are several difficulties here: is it possible to work with systems whose membrane structure evolves during the computation? (I guess, not); it is easy to introduce the objects present in the initial configuration starting from F, but how the "garbage" objects, those remaining inside in the end of a computation, can be removed? (this needs some kind of context sensitivity, for instance, using F as a promotor of erasing rules, or other similar ideas).

Needless to say that the resetting problem is of a particular interest for SN P systems, but this time I do not formulate this subproblem separately.

The reset problem can probably be addressed also from other points of view and in other contexts. An idea is to have somewhere/somewhow in the system a description of the system itself (a sort of "genome", in a "nucleus") and in the end of the computation to simply destroy the existing system, at the same time reconstructing it from its description, which amounts at resetting the initial system. Some related ideas, problems, and techniques are discussed, in a different set-up, in [6].

D. Let us continue with SN P systems, whose study is far from being exhausted, and which benefits of many suggestions coming from neurology, psychology, computer science. A problem suggested by Linqiang Pan and then preliminarily dealt with in [15] is the following: is it possible to work with SN P systems (in particular, having characterizations of Turing computable sets of numbers or recursively enumerable languages) which have neurons of a small number of types? By a "type" we can understand either only the set of rules present in a neuron (an SN P system whose neurons are of at most k such types is said to be in the kR-normal form), or we can consider both the set of rules and the number of spikes initially present in the neuron (and then we speak about SN P systems in the knR-normal form, if they have neurons of at most k such types).

In [15] it is proved that each recursively enumerable set of natural numbers can be generated by an SN P system (without forgetting rules, but using delays) in the $3R$-normal form when the number is obtained as the distance between spikes or when the number is given by the number of spikes from a specified neuron, but two types of neurons suffice in the accepting case. Slightly bigger values are obtained when we also consider the number of spikes initially present in a neuron for defining the "type" of a neuron: five and three, respectively.

A lot of open problems can now be formulated: First, we do not know whether or not these results can be improved. How these results extend to other classes of SN P systems? (Do the forgetting rules help? What about extended rules, about asynchronous SN P systems, systems with exhaustive use of rules, etc?) What about SN P systems which generate languages, or about universal SN P systems?

E. A problem about SN P systems which was formulated also earlier – see, e.g., problem **G** in [18] – concerns the possibility of dividing neurons, in general, of producing an exponential working space in a linear number of steps, in such a way to be able to solve computationally hard problems in a feasible time. Up to now, no idea how to divide neurons was discussed, although this operations was considered for cells in tissue-like P systems; see, e.g., [20]. What is really exciting is that a recent discovery of neuro-biology provides biological motivations for neuron division: there are so called neural stem cells which divide repeatedly, either without differentiating, hence producing further neural stem cells, or differentiating, so making possible the production of an exponential number of mature neurons; details can be found in [7].

In terms of SN P systems, we can easily capture the idea of having several types of neurons, for instance, using polarizations, as in P systems with active membranes. On the other hand, each neuron also has a label; if by division we obtain two new neurons with the same label, then polarizations are perhaps necessary; if we allow producing new neurons with labels different from the divided neuron, then the polarization can be included in the label (label changing can account also of polarizations).

However, the difficulties lie in other places. First, in SN P systems the contents of a neuron is described by the number of spikes it contains, and this number is usually controlled/sensed by means of a regular expression over alphabet $\{a\}$. This seems to be also the most natural idea for controlling the division, which leads to rules of the general form

$$[\ E]_i \to [\ E_1]_j [\ E_2]_k,$$

where E, E_1, E_2 are regular expressions over $\{a\}$, and i, j, k are labels of neurons (the labels of the produced neurons tell us which are the rules used in those neurons). By using such a rule, a neuron σ_i containing n spikes such that $a^n \in L(E)$ divides into two neurons, with labels j and k, containing n_1 and n_2 spikes, respectively, such that $n = n_1 + n_2$, and $a^{n_1} \in L(E_1),\ a^{n_2} \in L(E_2)$.

Now, the second important point/diffficulty appears: which are the synapses of the new neurons? There are two possibilities: to have the neurons places "in parallel" and to have them placed "serially/in cascade". In the first case, no connection exists between the two new neurons, but both of them inherit the links of the father neuron; in the second case, neuron σ_j inherits the synapses coming to σ_i, neuron σ_k inherits the synapses going out of σ_i, while a synapse (j, k) is also established. In order to distinguish the two interpretations, the rules are written as follows:

$$(a)\ \ [\ E]_i \to [\ E_1]_j || [\ E_2]_k,$$
$$(b)\ \ [\ E]_i \to [\ E_1]_j / [\ E_2]_k,$$

and the semantics described above is illustrated in Figure 1.

The use of these rules was already explored in [16], where SAT is solved in polynomial time by SN P systems having the possibility of dividing neurons. The

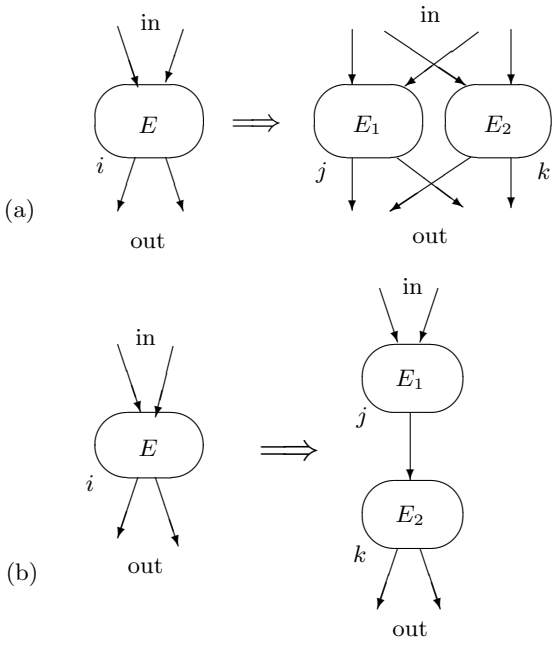


Fig. 1. Dividing neurons – two cases

solution is uniform. Is it possible to use only neuron division where the labels are not changed? (A more intensive use of the regular expressions controlling the division might be helpful, maybe also the use of polarizations, hence of suggestions coming from neural stem cells.) What about solving more difficult problems, for instance, `QSAT` (so that to cover **PSPACE**)?

In general, complexity investigations related to SN P systems need further efforts.

F. Both for stressing its interest and also because is related to the previous problem, I recall/reformulate here another problem from [18], problem **N**, about using pre-computed resources in solving hard problems by means of SN P systems. The motivation is clear, several papers have considered this idea, directly producing solutions to well-known problems, but still a more general approach is missing. The basic issue is to say which pre-computed (arbitrarily large) nets of neurons are allowed, how much information may be given "for free" (and how it can be evaluated/measured), how complexity classes should look like in this framework? A possible way to address some of these questions is to mix the idea of dividing neurons with that of pre-computed resources, and to build a net of neurons starting from a few given neurons, with a limited contents. This approach moves the pre-computation in the computation, with the important difference being that the time of the pre-computation (of "programming") is ignored as long as the pro-

duced system contains a limited quantity of information. (We return to the main problem: how to define and measure the information present in a system?) In the present volume there is a contribution related to this problem, namely [**?**], which address this issue starting from rules able to create neurons and synapses, hence to provide a dynamical SN P system.

G. Continuing with SN P systems, a problem which was vaguely formulate from time to time, but only orally, refers to a basic feature of the brain, the memory. How can this be captured in terms of SN P systems is an intriguing question. First, what means "memory"? In principle, the possibility to store some information for a certain time (remember that there is a short term and also a long term memory), and to use this information without losing it. For our systems, let us take the case of storing a number; we need a module of an SN P system where this number is "memorized" in such a way that in precise circumstances (e.g., at request, when a signal comes from outside the module), the number is "communicated" without "forgetting" it. In turn, the communication can be to only one destination or to several destinations. There are probably several ways to build such a module. The difficulty comes from the fact that if the number n is stored in the form of n spikes, "reading" these spikes would consume them, hence it is necessary to produce copies which in the end of the process reset the module. This is clearly possible in terms of SN P systems, what remains to do is to explicitly write the system. However, new questions appear related to the efficiency of the construction, in terms of time (after getting the request for the number n, how many steps are necessary in order to provide the information and to reset the module?), and also in terms of descriptional complexity (how many neurons and rules, how many spikes, how complex rules?). It is possible that a sort of "orthogonal" pair of ideas are useful: many spikes in a few neurons (n spikes in one neuron already is a way to store the number, what remains is to read and reset), or a few spikes in many neurons (a cycle of n neurons among which a single spike circulates, completing the cycle in n steps, is another "memory cell" which stores the number n; again, we need to read and reset, if possible, using only a few spikes). Another possible question is to build a reusable module, able to store several numbers: for a while (e.g., until a special signal) a number n_1 is stored, after that another number n_2, and so on.

H. The previous problem can be placed in a more general set-up, that of modeling other neurobiological issues in terms of SN P systems. A contribution in this respect is already included in the present volume, [13], where the sleep-awake passage is considered. Of course, the approach is somewhat metaphorical, as the distance between the physiological functioning of the brain and the formal structure and functioning of an SN P system is obvious, but still this illustrates the versatility and modeling power of SN P systems. Further biological details should be considered in order to have a model with some significance for the brain study (computer simulations will then be necessary, like in the case of other applications of P systems in modeling biological processes). However, also at this formal level there are several problems to consider. For instance, what happens if the sleeping

period is shortened, e.g., because a signal comes from the environment? Can this lead to a "damage" of the system? In general, what about taking the environment into account? For instance, we can consider a larger system, where some modules sleep while other modules not; during the awake period it is natural to assume that the modules interact, but not when one of them is sleeping, excepting the case of an "emergency", when a sleeping module can be awakened at the request of a neighboring module. Several similar scenarios can be imagined, maybe also coupling the sleep-awake issue with the memory issue.

I. Let us now pass to general P systems, where still there are many unsettled problems and, what is more important for the health of membrane computing, there still appear new research directions. One of them was proposed and preliminarily explored in [23]: space complexity for P systems. It is somewhat surprising that this classic complexity issue was not considered so far, while the time complexity is already intensively investigated. We do not enter here into any detail – anyway, the details are moving fast in this area – but we refer to the mentioned paper.

J. Remaining in the framework of complexity, it is worth noting a sound recent result, the first one of this type, dealing with the relation between uniform and semi-uniform solutions to (computationally hard) problems. In [24] one produces a first example where semi-uniform constructions cover a strictly larger family than the uniform one. There also are classes of P systems for which the two approaches, uniform and semi-uniform, characterize the same family of problems – see, e.g., [22]. However, as also in [24] is stated, it is highly possible – and definitely of interest to find it – that the separation between uniform and semi-uniform families is strict for further classes of P systems.

K. Similarly short formulations (actually, only pointing out the problem and the main reference, always from the present volume) will also have the next problems. First, the idea of merging two "classic" topics in membrane computing: P systems with active membranes and P systems with string objects. Up to now only P systems with active membranes and with symbol objects were considered, but the necessity to mix the two ideas arose in [2] in the framework of a specific application. It is also natural to investigate this kind of systems as a goal per se, mainly from the complexity (descriptional and computational – and here both time and space) points of view.

L. A related general problem is suggested by the above paper [2]: looking for other "practical" problems, of known polynomial complexity, but which are so frequent and important that as efficient as possible solutions are desirable. The mentioned paper considers such a problem: search and updating of dictionaries. Many other problems can probably be addressed in terms of membrane computing, with good (theoretical) results. And, as it happens also in the case of [2], it is also

possible that new classes of P systems will emerge in this context, as requested by the respective applications.

M. A problem several times mentioned, also addressed from various angles in several places concerns the general idea of "going backwards". Sometimes it is directly called computing backwards, as in [8], other times one speaks about dual P systems, as in [1], about reversible P systems, as in [12]. In the present volume, the reversibility is addressed again, in [4], but the question still deserves further research efforts.

N. I close this short list of research topics by mentioning the one proposed in [14] – the title is self-explanatory, so I do not enter into other details, but I mention that the issue seems rather promising, both theoretically (composition-decomposition, structural/descriptional complexity, etc.) and from the applications point of view.

Lists of open problems were presented in all previous brainstorming volumes – the reader can find them at [25]. There also exists an attempt to follow the research and to present the known solutions to some of these problems – see [19].

Acknowledgements

Work supported by Proyecto de Excelencia con investigador de recognoscida valia, de la Junta de Andalucia, grant P08 – TIC 04200.

References

1. O. Agrigoroaiei, G. Ciobanu: Dual P systems. *Membrane Computing - 9th International Workshop*, LNCS 5391, 2009, 95–107.
2. A. Alhazov, S. Cojocaru, L. Malahova, Y. Rogozhin: Dictionary search and update by P systems with string-objects and active membranes. In the present volume.
3. A. Alhazov, M. Margenstern, S. Verlan: Fast synchronization in P systems. *Membrane Computing, 9th Intern. Workshop. Edinburgh, UK, July 2008* (D.W. Corne et al eds.), LNCS 5391, Springer, Berlin, 2009, 118–128.
4. A. Alhazov, K. Morita: A short note on reversibility in P systems. In the present volume.
5. F. Bernardini, M. Gheorghe, M. Margenstern, S. Verlan: How to synchronize the activity of all components of a P system? *Prof. Intern. Workshop Automata for Cellular and Molecular Computing* (G. Vaszil, ed.), Budapest, Hungary, 2007, 11-22.
6. E. Csuhaj-Varju, A. Di Nola, Gh. Păun, M.J. Pérez-Jiménez, G. Vaszil: Editing configurations of P systems, *Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, RGNC Report 01/2005, 131–155, and *Fundamenta Informaticae*, 82, 1-2 (2008), 29–46.
7. R. Galli, A. Gritti, L. Bonfanti, A.L. Vescovi: Neural stem cells: an overview. *Circulation Research*, 92 (2003), 598–608.

8. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Computing backwards with P systems. In the present volume.
9. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. *Theoretical Computer Sci.*, 372, 2-3 (2007), 196–217.
10. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
11. M. Ionescu, D. Sburlan: Some applications of spiking neural P systems. *Proc. WMC8*, Thessaloniki, June 2007, 383–394, and *Computing and Informatics*, 27 (2008), 515–528.
12. A. Leporati, C. Zandron, G. Mauri: Reversible P systems to simulate Fredkin circuits. *Fundam. Inform.*, 74, 4 (2006), 529–548.
13. J.M. Mingo: Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. In the present volume.
14. R. Nicolescu, M.J. Dinneen, Y.-B. Kim: Structured modeling with hyperdag P systems: Part A. In the present volume.
15. L. Pan, Gh. Păun: New normal forms for spiking neural P systems. In the present volume.
16. L. Pan, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with neuron division and budding: In the present volume.
17. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
18. Gh. Păun: Twenty six research topics about spiking neural P systems. *Fiftth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, et al., eds.), Sevilla, January 29-February 2, 2007, Fenix Editora, Sevilla, 2007, 263–280.
19. Gh. Păun: Tracing some open problems in membrane computing, *Romanian J. Information Sci. and Technology*, 10, 4 (2007), 303–314.
20. Gh. Păun, M. Pérez-Jiménez, A. Riscos-Núñez: Tissue P systems with cell division. *Second Brainstorming Week on Membrane Computing* (A. Riscos-Núñez et al. eds.), Technical Report 01/04 of RGNC, Sevilla University, Spain, 2004, 380–386, and *Intern. J. Computers. Comm. Control*, 3, 3 (2008), 295–303.
21. Gh. Păun, G. Rozenberg, A. Salomaa, eds.: *Handbook of Membrane Computing.* Oxford University Press, 2009.
22. M.J. Pérez-Jiménez, A. Riscos-Núñez, Á. Romero-Jiménez, D. Woods: Complexity – Membrane division, membrane creation. In [21], in press.
23. A.E. Porreca, A. Leporati, G. Mauri, C. Zandron: Introducing a space complexity measure for P systems. In the present volume.
24. D. Woods, N. Murphy:
25. The P Systems Website: `http://ppage.psystems.eu`.

A Bibliography of Spiking Neural P Systems

Gheorghe Păun

Institute of Mathematics of the Romanian Academy
PO Box 1-764, 014700 Bucureşti, Romania
and
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
george.paun@imar.ro, gpaun@us.es

What follows is a bibliography of spiking neural P systems (SN P systems, for short), at the level of April 2009. This bibliography is included in the present volume having in mind the fact that this research area attracted much interest in the few years since it was initiated, so that it might be useful to the reader to have a comprehensive list of titles at hand, as complete as we were able to compile it. Of course, most of this information can also be found in the membrane computing website, at http://ppage.psystems.eu, with a comprehensive survey also provided by the chapter devoted to SN P systems in the *Handbook of Membrane Computing* (Gh. Păun, G. Rozenberg, A. Salomaa, eds.), Oxford University Press, 2009. The list which follows also includes the papers on SN P systems present in the present volume (with the indication "in the present volume"); the papers from the previous proceedings volumes of the Brainstorming Week on Membrane Computing are given with indications of the form "BWMC2007" with the obvious meaning, then specifying the pages. The three brainstorming volumes cited in this way are:

1. M.A. Gutiérrez-Naranjo, et al., eds.: *Fourth Brainstorming Week on Membrane Computing*, Sevilla, January 30-February 3, 2006, vol. I and II, Fenix Editora, Sevilla, 2006.
2. M.A. Gutiérrez-Naranjo, et al., eds.: *Fiftth Brainstorming Week on Membrane Computing*, Sevilla, January 29-February 2, 2007, Fenix Editora, Sevilla, 2007.
3. D. Diaz-Pernil, et al., eds.: *Sixth Brainstorming Week on Membrane Computing*, Sevilla, February 4-8, 2008, Fenix Editora, Sevilla, 2008.

For some of the papers it is indicated only the year, this meaning that the information we had when compiling the list was that the paper was still unpublished.

1. A. Alhazov, R. Freund, M. Oswald, M. Slavkovik: Extended variants of spiking neural P systems generating strings and vectors of non-negative integers. *WMC7, 2006*, 88–101, and *Membrane Computing, WMC2006, Leiden, Revised, Selected and Invited Papers*, LNCS 4361, Springer, 2006, 123–134.
2. A. Binder, R. Freund, M. Oswald: Extended spiking neural P systems with astrocytes - variants for modelling the brain. *Proc. 13th Intern. Symp. AL and Robotics*, AROB2008, Beppu, Japan, 520–524.
3. A. Binder, R. Freund, M. Oswald, L. Vock: Extended spiking neural P systems with excitatory and inhibitory astrocytes. Submitted, 2007.
4. M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems; decidability and undecidability. *Proc. DNA13*, LNCS 4848, Springer, 2007.
5. M. Cavaliere, E. Egecioglu, O.H. Ibarra, M. Ionescu, Gh. Păun, S. Woodworth: Asynchronous spiking neural P systems. *Theoretical Computer Sci.*, 2009.
6. M. Cavaliere, I. Mura: Experiments on the reliability of stochastic spiking neural P systems. *Natural Computing*, 7, 4 (2008), 453–470.
7. R. Ceterchi, A.I. Tomescu: Spiking neural P systems – a natural model for sorting networks. *BWMC2008*, 93–106.
8. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *BWMC2006*, vol. I, 169–194, and *Fundamenta Informaticae*, 75, 1-4 (2007), 141–162.
9. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. *BWMC2006*, vol. I, 195–206, and *Proc. 8th Intern. Conf. on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
10. 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*, 7, 2 (2008), 147–166.
11. H. Chen, M. Ionescu, A. Păun, Gh. Păun, B. Popa: On trace languages generated by spiking neural P systems. *BWMC2006*, vol. I, 207–224, and *Eighth International Workshop on Descriptional Complexity of Formal Systems* (DCFS 2006), June 21-23, 2006, Las Cruces, New Mexico, USA, 94–105.
12. H. Chen, T.-O. Ishdorj, Gh. Păun: Computing along the axon. *BWMC2006*, vol. I, 225–240, and *Pre-proceedings BIC-TA*, Wuhan, 2006, 60-70, and *Progress in Natural Science*, 17, 4 (2007), 418–423.
13. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. *BWMC2006*, vol. I, 241–266.
14. H. Chen, T.-O. Ishdorj, Gh. Păun, M.J. Pérez-Jiménez: Handling languages with spiking neural P systems with extended rules. *Romanian J. Information Sci. and Technology*, 9, 3 (2006), 151–162.
15. R. Freund, M. Ionescu, M. Oswald: Extended spiking neural P systems with decaying spikes and/or total spiking. *ACMC/FCT 2007 Workshop*, Budapest, *Intern. J. Found. Computer Sci.*, 19 (2008), 1223–1234.
16. R. Freund, M. Oswald: Spiking neural P systems with inhibitory axons. *AROB Conf.*, Japan, 2007.

17. R. Freund, M. Oswald: Regular ω-languages defined by extended spiking neural P systems. *Fundamenta Informaticae*, 83, 1-2 (2008), 65–73.
18. M. Garcia-Arnau, D. Pérez, A. Rodriguez-Patón, P. Sosik: On the power of elementary operations in spiking neural P systems. Submitted, 2008.
19. M. Garcia-Arnau, A. Rodriguez-Patón, D. Pérez, P. Sosik: Spiking neural P systems: Stronger normal forms. *BWMC2007*, 157–178.
20. M.A. Gutiérrez-Naranjo, A. Leporati: Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. *BWMC2008*, 193–210.
21. M.A. Gutiérrez-Naranjo, A. Leporati: Performing arithmetic operations with spiking neural P systems. In the present volume.
22. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez. A first model for Hebbian learning with spiking neural P systems. *BWMC2008*, 211–234.
23. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodriguez-Patón, P. Sosik, S. Woodworth: Normal forms for spiking neural P systems. *BWMC2006*, vol. II, 105–136, and *Theoretical Computer Sci.*, 372, 2-3 (2007), 196–217.
24. O.H. Ibarra, A. Păun, A. Rodriguez-Patón: Sequentiality induced by spike numbers in SN P systems. *Proc. 14th Intern. Meeting on DNA Computing*, Prague, June 2008, 36–46.
25. O.H. Ibarra, S. Woodworth: Characterizations of some restricted spiking neural P systems. and *Membrane Computing, WMC2006, Leiden, Revised, Selected and Invited Papers*, LNCS 4361, Springer, 2006, 424–442.
26. O.H. Ibarra, S. Woodworth: Spiking neural P systems: some characterizations. *Proc. FCT 2007*, Budapest, LNCS 4639, 23–37.
27. O.H. Ibarra, S. Woodworth: Characterizing regular languages by spiking neural P systems. *Intern. J. Found. Computer Sci.*, 18, 6 (2007), 1247–1256.
28. O.H. Ibarra, S. Woodworth, F. Yu, A. Păun: On spiking neural P systems and partially blind counter machines. *Proc. UC2006*, LNCS 4135, Springer, 2006, 113–129.
29. M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems. *Proc. DNA12* (C. Mao, Y. Yokomori, B.-T. Zhang, eds.), Seul, June 2006, 32–42, and *DNA Computing. 12th Intern. Meeting on DNA Computing, DNA12, Seoul, Korea, June 2006, Revised Selected Papers* (C. Mao, T. Yokomori, eds.), LNCS 4287, Springer, 2007, 1–16.
30. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
31. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems with an exhaustive use of rules. *Intern. J. Unconventional Computing*, 3, 2 (2007), 135–154.
32. M. Ionescu, D. Sburlan: Some applications of spiking neural P systems. *Proc. WMC8*, Thessaloniki, June 2007, 383–394, and *Computing and Informatics*, 27 (2008), 515–528.
33. M. Ionescu, C.I. Tîrnăucă, C. Tîrnăucă: Dreams and spiking neural P systems. *Romanian J. Inform. Sci. and Technology*, 12 (2009), in press.

34. T.-O. Ishdorj, A. Leporati: Uniform solutions to SAT and 3-SAT by spiking neural P systems with pre-computed resources. *Natural Computing*, to appear.
35. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zeng, X. Zhang: Deterministic solutions to `QSAT` and `Q3SAT` by spiking neural P systems with pre-computed resources. In the present volume.
36. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M.J. Pérez-Jiménez: Uniform solutions to SAT and Subset-Sum by spiking neural P systems. Submitted, 2007.
37. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: On the computational power of spiking neural P systems. *BWMC2007*, 227–246.
38. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical NP-complete problems with spiking neural P systems. *Proc. WMC8*, Thessaloniki, June 2007, 405–424.
39. V.P. Metta, K. Krithivasan: Spiking neural P systems and Petri nets. Submitted, 2008.
40. J.M. Mingo: Una approximacion al control neural del sueno de ondas lentas mediante spiking neural P systems. Submitted, 2008.
41. J.M. Mingo: Sleep-awake switch with spiking neural P systems: A basic proposal and new issues. In the present volume.
42. T. Neary: A small universal spiking neural P system. *Intern. Workshop. Computing with Biomolecules* (E. Csuhaj-Varju et al., eds.), Viena, 2008, 65–74.
43. T. Neary: On the computational complexity of spiking neural P systems. *Unconventional Computation. 7th Intern. Conf. Vienna, 2008* (C.S. Calude at al., eds.), LNCS 5204, 2008, 189–205.
44. A. Obtulowicz: Spiking neural P systems and modularization of complex networks from cortical neural network to social networks. In the present volume.
45. L. Pan, Gh. Păun: New normal forms for spiking neural P systems. In the present volume.
46. L. Pan, Gh. Păun: Spiking neural P systems with anti-spikes. In the present volume.
47. L. Pan, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with neuron division and budding. In the present volume.
48. L. Pan, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems: A short introduction and new normal forms, *Advanced Computational Tehcnologies* (C. Enăchescu, F. Filip, B. Iantovics, eds.), Ed. Academiei, Bucureşti, 2009.
49. L. Pan, J. Wang, H.J. Hoogeboom: Excitatory and inhibitory neural P systems. Submitted, 2007.
50. A. Păun, Gh. Păun: Small universal spiking neural P systems. *BioSystems*, 90, 1 (2007), 48–60.
51. Gh. Păun: Languages in membrane computing. Some details for spiking neural P systems. *Proc. 10th DLT Conf.* (invited talk), Santa Barbara, USA, 2006, LNCS 4036, Springer, Berlin, 2006, 20–35.
52. Gh. Păun: Twenty six research topics about spiking neural P systems. *BWMC2007*, 263–280.

53. Gh. Păun: A quick overview of membrane computing with some details about spiking neural P systems. *Frontiers of Computer Science in China*, 1,1 (2007), 37–49.
54. Gh. Păun: Spiking neural P systems. A tutorial. *Bulletin of the EATCS*, 91 (Febr. 2007), 145–159.
55. Gh. Păun: Spiking neural P systems. Power and efficiency. *Bio-Inspired Modeling of Cognitive Tasks, Proc. IWINAC 2007* (J. Mira, J.R. Alvarez, eds.), Mar Menor, 2007, LNCS 4527, 153–169.
56. Gh. Păun: Spiking neural P systems used as acceptors and transducers. *CIAA 2007, 12th Conf.*, Prague, July 2007, LNCS 4783 (J. Holub, J. Zdarek, eds.), Springer, Berlin, 2007, 1–4.
57. Gh. Păun: Spiking neural P systems with astrocyte-like control. *JUCS*, 13, 11 (2007), 1707–1721.
58. Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems. Recent results, research topics. Submitted, 2007.
59. Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems. An overview. *Advancing Artificial Intelligence through Biological Process Applications* (A.B. Porto, A. Pazos, W. Buno, eds.), Medical Information Science Reference, Hershey, New York, 2008, 60–73.
60. 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.
61. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Computing morphisms by spiking neural P systems. *Intern. J. Found. Computer Sci.*, 18, 6 (2007), 1371–1382.
62. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Infinite spike trains in spiking neural P systems. Manuscript, 2005.
63. Gh. Păun, M.J. Pérez-Jiménez, A. Salomaa: Bounding the indegree of spiking neural P systems. *TUCS Technical Report* 773, 2006.
64. Gh. Păun, M.J. Pérez-Jiménez, A. Salomaa: Spiking neural P systems. An early survey. *Intern. J. Found. Computer Sci.*, 18 (2007), 435–456.
65. D. Ramirez-Martinez, M.A. Gutiérrez-Naranjo: A software tool for dealing with spiking neural P systems. *BWMC2007*, 299–314.
66. J. Wang, T.-O. Ishdorj, L. Pan: Efficiency of spiking neural P systems. In the present volume.
67. X. Zeng, X. Zhang, L. Pan: Homogeneous spiking neural P systems. Submitted, 2009.
68. X. Zhang, T.-O. Ishdorj, X. Zeng, L. Pan: Solving PSPACE-complete problems by spiking neural P systems with pre-computed resources. Submitted, 2008.
69. X. Zhang, Y. Jiang, L. Pan: Small universal spiking neural P systems with exhaustive use of rules. *Proc. Third Intern. Conf. on Bio-Inspired Computing. Theory and Appl.*, Adelaide, 2008, 117–127.
70. X. Zhang, J. Wang, L. Pan: A note on the generative power of axon P systems. *Intern. J. CCC*, 4, 1 (2009), 92–98.

71. X. Zhang, X. Zeng, L. Pan: On string languages generated by SN P systems with exhaustive use of rules. *Natural Computing*, 7 (2008), 535–549.
72. X. Zhang, X. Zeng, L. Pan: Smaller universal spiking neural P systems. *Fundamenta Informaticae*, 87 (2008), 117–136.
73. X. Zhang, X. Zeng, L. Pan: On string languages generated by asynchronous spiking neural P systems. *Theoretical Computer Science*, DOI: 10.1016/j.tcs.2008.12.055.

Introducing a Space Complexity Measure for P Systems

Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, Claudio Zandron

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano-Bicocca
viale Sarca 336, 20126 Milano, Italy
{porreca, leporati, mauri, zandron}@disco.unimib.it

Summary. We define space complexity classes in the framework of membrane computing, giving some initial results about their mutual relations and their connection with time complexity classes, and identifying some potentially interesting problems which require further research.

1 Introduction

Until now, research on the complexity theoretic aspects of membrane computing has mainly focused on the time resource. In particular, since the introduction of P systems with active membranes [5], various results concerning time complexity classes defined in terms of P systems with active membranes were given, comparing different classes obtained using various ingredients (such as, e.g., polarizations, dissolution, uniformity, etc.). Other works considered the comparisons between them and the usual complexity classes defined in terms of Turing machines, either from the point of view of time complexity [8, 3, 11], or space complexity classes [10, 1, 9].

Despite the vivid interest on this subject, up to now no investigations concerning space complexity classes defined in terms of P systems have been carried out in formal terms. Of course, the evident relation between time and space in P systems with active membranes is informally acknowledged: all results concerning solutions to **NP**-complete problems are solved using an exponential workspace obtained in polynomial time. Nonetheless, there is no formal definition of space complexity classes for P systems and, as a consequence, no formal results concerning the relations between space and time.

In this paper, we make the first steps in this direction, first by defining the space requirements for a given P system on a specific computation, and then by formally defining space complexity classes for P systems. We will then give a first set of results concerning relations among complexity classes for P systems, some of them directly following from the definitions, and others which can be derived

by considering space requirements of various solutions proposed in the literature which make use of P systems with active membranes.

In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems. For a systematic introduction, we refer the reader to [6]. A survey and an up-to-date bibliography concerning P systems can be found at the web address `http://ppage.psystems.eu`.

The rest of the paper is organized as follows. In Section 2 we give basic definitions for membrane systems which will be used throughout the rest of the paper. In Section 3 we give formal definitions of space complexity classes in terms of P systems. In Section 4 we present some results concerning such complexity classes, which follow immediately from the definitions, while in Section 5 we present some results which can be obtained by considering known results for time complexity classes in the framework of P systems with active membranes. Section 6 concludes the paper by presenting some conjectures and open problems concerning space complexity.

2 Definitions

We begin by recalling the formal definition of P systems with active membranes and the usual process by which they are used to solve decision problems. Moreover, we recall the main definitions related to time complexity classes in this framework.

Definition 1. *A* P system with active membranes *of degree* $m \geq 1$ *is a structure*

$$\Pi = (\Gamma, \Lambda, \mu, w_1, \ldots, w_m, R)$$

where:

- Γ *is a finite alphabet of symbols or objects;*
- Λ *is a finite set of labels;*
- μ *is a membrane structure (i.e., a rooted, unordered tree) of* m *membranes, labeled with elements of* Λ*; different membranes may be given the same label;*
- $w_1, \ldots, w_m$ *are multisets over* Γ *describing the initial contents of the* m *membranes in* μ*;*
- R *is a finite set of developmental rules.*

The polarization *of a membrane can be* $+$ *(positive),* $-$ *(negative) or* 0 *(neutral); each membrane is assumed to be initially neutral.*

Developmental rules are of the following six kinds:

- Object evolution rule *of the form* $[a \to w]_h^\alpha$
  It can be applied to a membrane labeled by h*, having polarization* α *and containing an occurrence of the object* a*; the object* a *is rewritten into the multiset* w *(i.e.,* a *is removed from the multiset in* h *and replaced by the objects in* w*).*

- Communication rule *of the form* $a\,[\,]_h^\alpha \to [b]_h^\beta$
  It can be applied to a membrane labeled by h, having polarization α and such that the external region contains an occurrence of the object a; the object a is sent in to h becoming b and, simultaneously, the polarization of h is changed to β.
- Communication rule *of the form* $[a]_h^\alpha \to [\,]_h^\beta\, b$
  It can be applied to a membrane labeled by h, having polarization α and containing an occurrence of the object a; the object a is sent out from h to the outside region becoming b and, simultaneously, the polarization of h is changed to β.
- Dissolution rule *of the form* $[a]_h^\alpha \to b$
  It can be applied to a membrane labeled by h, having polarization α and containing an occurrence of the object a; the membrane h is dissolved and its content is left in the surrounding region unaltered, except that an occurrence of a becomes b.
- Elementary division rule *of the form* $[a]_h^\alpha \to [b]_h^\beta\,[c]_h^\gamma$
  It can be applied to an elementary membrane labeled by h, having polarization α and containing an occurrence of the object a; the membrane is divided into two membranes having label h and polarizations β and γ; the object a is replaced, respectively, by b and c while the other objects in the initial multiset are copied to both membranes.
- Non-elementary division rule *of the form*

$$\big[[\,]_{h_1}^+ \cdots [\,]_{h_k}^+ [\,]_{h_{k+1}}^- \cdots [\,]_{h_n}^- \big]_h^\alpha \to \big[[\,]_{h_1}^\delta \cdots [\,]_{h_k}^\delta \big]_h^\beta \big[[\,]_{h_{k+1}}^\epsilon \cdots [\,]_{h_n}^\epsilon \big]_h^\gamma$$

  It can be applied to a non-elementary membrane labeled by h, having polarization α, containing the positively charged membranes $h_1, \ldots, h_k$ and the negatively charged membranes $h_{k+1}, \ldots, h_n$; no other non-neutral membrane may be contained in h. The membrane h is divided into two copies with polarization β and γ; the positive children are placed inside the former, their polarizations changed to δ, while the negative ones are placed inside the latter, their polarizations changed to ϵ. Any neutral membrane inside h is duplicated and placed inside both copies.

A configuration *of a P system with active membranes Π is given by a membrane structure and the multisets contained in its regions. In particular, the* initial configuration *is given by the membrane structure μ and the initial contents of its membranes $w_1, \ldots, w_m$. A computation step leads from a configuration to the next one according to the following principles:*

- *the developmental rules are applied in a* maximally parallel *way: when one or more rules can be applied to an object and/or membrane, then one of them* must *be applied. The only elements left untouched are those which cannot be subject to any rule;*
- *each object can be subject to only one rule during that step. Also membranes can be subject to only one rule, except that* any *number of object evolution rules can be applied inside them;*

- *when more than one rule can be applied to an object or membrane, then the one actually applied is chosen nondeterministically. Thus multiple, distinct configurations may be reachable by means of a computation step from a single configuration;*
- *when a dissolution or division rule is applied to a membrane, the multiset of objects to be released outside or copied is the one* after *any application of object evolution rules inside such membrane;*
- *the skin membrane cannot be divided or dissolved, nor any object can be sent in from the environment surrounding it (i.e., an object which leaves the skin membrane cannot be brought in again).*

A sequence of configurations, each one reachable from the previous one by means of developmental rules, is called a computation*. Due to nondeterminism, there may be multiple computations starting from the initial configuration, thus producing a computation tree. A computation halts when no further configuration can be reached, i.e., when no rule can be applied in a given configuration.*

Families of *recognizer* P systems can be used to solve decision problems as follows.

Definition 2. *Let Π be a P system whose alphabet contains two distinct objects yes and no, such that every computation of Π is halting and during each computation exactly one of the objects yes, no is sent out from the skin to signal acceptance or rejection. If all the computations of Π agree on the result, then Π is said to be* confluent*; if this is not necessarily the case, then it is said to be* non-confluent *and the global result is acceptance iff there exists an accepting computation.*

Definition 3. *Let $L \subseteq \Sigma^\star$ be a language, $\mathcal{D}$ a class of P systems and let $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ be a family of P systems, either confluent or non-confluent. We say that $\mathbf{\Pi}$* decides *L when, for each $x \in \Sigma^\star$, $x \in L$ iff Π_x accepts.*

Complexity classes for P systems are defined by imposing a uniformity condition on $\mathbf{\Pi}$ and restricting the amount of time available for deciding a language.

Definition 4. *Consider a language $L \subseteq \Sigma^\star$, a class of recognizer P systems $\mathcal{D}$, and let $f\colon \mathbb{N} \to \mathbb{N}$ be a proper complexity function. We say that L belongs to the complexity class $\mathbf{MC}^\star_{\mathcal{D}}(f)$ if and only if there exists a family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L such that*

- *$\mathbf{\Pi}$ is* semi-uniform*, i.e., there exists a deterministic Turing machine which, for each input $x \in \Sigma^\star$, constructs the P system Π_x in polynomial time;*
- *$\mathbf{\Pi}$ operates in time f, i.e., for each $x \in \Sigma^\star$, every computation of Π_x halts within $f(|x|)$ steps.*

In particular, a language $L \subseteq \Sigma^\star$ belongs to the complexity class $\mathbf{PMC}^\star_{\mathcal{D}}$ iff there exists a semi-uniform family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L in polynomial time.

The analogous complexity classes for non-*confluent P systems are denoted by $\mathbf{NMC}^\star_{\mathcal{D}}(f)$ and $\mathbf{NPMC}^\star_{\mathcal{D}}$.*

Another set of complexity classes is defined in terms of *uniform* families of recognizer P systems:

Definition 5. *Consider a language $L \subseteq \Sigma^\star$, a class of recognizer P systems $\mathcal{D}$, and let $f\colon \mathbb{N} \to \mathbb{N}$ be a proper complexity function. We say that L belongs to the complexity class $\mathbf{MC}_{\mathcal{D}}(f)$ if and only if there exists a family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L such that*

- $\mathbf{\Pi}$ *is* uniform*, i.e., for each $x \in \Sigma^\star$ deciding whether $x \in L$ is performed as follows: first, a polynomial-time deterministic Turing machine, given the length $n = |x|$ as a unary integer, constructs a P system Π_n with a distinguished input membrane; then, another polynomial-time DTM computes a coding of the string x as a multiset w_x, which is finally added to the input membrane of Π_n, thus obtaining a P system Π_x accepting iff $x \in L$.*
- $\mathbf{\Pi}$ *operates in time f, i.e., for each $x \in \Sigma^\star$, every computation of Π_x halts within $f(|x|)$ steps.*

In particular, a language $L \subseteq \Sigma^\star$ belongs to the complexity class $\mathbf{PMC}_{\mathcal{D}}$ iff there exists a uniform family of confluent P systems $\mathbf{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L in polynomial time.

The analogous complexity classes for non-*confluent P systems are denoted by $\mathbf{NMC}_{\mathcal{D}}(f)$ and $\mathbf{NPMC}_{\mathcal{D}}$.*

3 A Measure of Space Complexity for P Systems

In order to define the space complexity of P systems, we first need to establish a measure of the size of their configurations. The first definition we propose is based on an hypothetical implementation of P systems by means of real biochemical materials (cellular membranes and molecules). Under this assumption, every single object takes some constant physical space: this is equivalent to using a unary coding to represent multiplicities.

Definition 6. *Let $\mathcal{C}$ be a configuration of a P system Π, that is, a rooted, unordered tree μ representing the membrane structure of Π, whose vertices are labeled with the multisets describing the contents of each region. The* size $|\mathcal{C}|$ of $\mathcal{C}$ *is then defined as the sum of number of membranes in μ and the total number of objects they contain.*

An alternative definition focuses on the simulative point of view, i.e., on the implementation of P systems *in silico*, where it is not necessary to actually store every single object (using a unary representation), but we can just store their multiplicity as a binary number, thus requiring exponentially less space for each kind of symbol.

Definition 7 (Alternative). *Let $\mathcal{C}$ be a configuration of a P system Π, that is, a rooted, unordered tree μ representing the membrane structure of Π, whose vertices*

are labeled with the multisets describing the contents of each region. The size $|\mathcal{C}|$ of $\mathcal{C}$ *is then defined as the sum of number of membranes in* μ *and the total number of bits required to store the objects they contain.*

In the following discussion we will assume the first definition; however notice that the actual results might or might not depend on the precise choice between Definitions 6 and 7 (a thorough analysis of the differences involves a clarification of the relative importance of the number of membranes and the number of objects in various classes of P systems, and it is left as an open problem).

Once a notion of configuration size is established, we need to take account of all possible computation paths which can develop even in confluent recognizer P systems; the following definitions are given in the spirit of those concerning time complexity for P systems [7].

Definition 8. *Let* Π *be a (confluent or non-confluent) recognizer P system, and let* $\boldsymbol{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_m)$ *be a halting computation of* Π*, that is, a sequence of configurations starting from the initial one and such that every subsequent one is reachable in one step by applying developmental rules in a maximally parallel way. The* space required by $\boldsymbol{\mathcal{C}}$ *is defined as*

$$|\boldsymbol{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_m|\}.$$

The space required by Π *itself is then*

$$|\Pi| = \max\{|\boldsymbol{\mathcal{C}}| : \boldsymbol{\mathcal{C}} \text{ is a halting computation of } \Pi\}.$$

Definition 9. *Let* $\boldsymbol{\Pi} = \{\Pi_x : x \in \Sigma^\star\}$ *be a uniform or semi-uniform family of recognizer P systems, each* Π_x *deciding the membership of the string* x *in a language* $L \subseteq \Sigma^\star$*; also let* $f\colon \mathbb{N} \to \mathbb{N}$*. We say that* $\boldsymbol{\Pi}$ operates within space bound f *iff* $|\Pi_x| \leq f(|x|)$ *for each* $x \in \Sigma^\star$*.*

We are now ready to define space complexity classes for P systems.

Definition 10. *Let* $\mathcal{D}$ *be a class of confluent recognizer P systems; let* $f\colon \mathbb{N} \to \mathbb{N}$ *and* $L \subseteq \Sigma^\star$*. Then* $L \in \mathbf{MCSPACE}^\star_{\mathcal{D}}(f)$ *iff* L *is decided by a semi-uniform family* $\boldsymbol{\Pi} \subseteq \mathcal{D}$ *of P systems operating within space bound* f*.*

The corresponding class for uniform families of confluent P systems is denoted by $\mathbf{MCSPACE}_{\mathcal{D}}(f)$*, while in the non-confluent case we have the classes* $\mathbf{NMCSPACE}^\star_{\mathcal{D}}(f)$ *and* $\mathbf{NMCSPACE}_{\mathcal{D}}(f)$ *respectively.*

As usual, we provide a number of abbreviations for important space classes.

Definition 11. *The classes corresponding to polynomial and exponential space, in the semi-uniform and confluent case, are*

$$\mathbf{PMCSPACE}^\star_{\mathcal{D}} = \mathbf{MCSPACE}^\star_{\mathcal{D}}(\mathrm{poly}(n))$$
$$\mathbf{EXPMCSPACE}^\star_{\mathcal{D}} = \mathbf{MCSPACE}^\star_{\mathcal{D}}\big(2^{\mathrm{poly}(n)}\big).$$

The definitions are analogous in the uniform and non-confluent cases.

4 Basic Results

From the above definitions, some results concerning space complexity classes and their relations with time complexity classes follow immediately. We state them only for semi-uniform families, but they also hold in the uniform case.

The first two propositions can be immediately derived from the definitions.

Proposition 1. *The following inclusions hold:*

$$\mathbf{PMCSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{EXPMCSPACE}^{\star}_{\mathcal{D}}$$
$$\mathbf{NPMCSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{NEXPMCSPACE}^{\star}_{\mathcal{D}}.$$

Proposition 2. $\mathbf{MCSPACE}^{\star}_{\mathcal{D}}(f) \subseteq \mathbf{NMCSPACE}^{\star}_{\mathcal{D}}(f)$ *for each* $f\colon \mathbb{N} \to \mathbb{N}$*, and in particular*

$$\mathbf{PMCSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{NPMCSPACE}^{\star}_{\mathcal{D}}$$
$$\mathbf{EXPMCSPACE}^{\star}_{\mathcal{D}} \subseteq \mathbf{NEXPMCSPACE}^{\star}_{\mathcal{D}}.$$

The following results mirror those which hold for for Turing machines, and they describe closure properties and provide an upper bound for time requirements of P systems operating in bounded space.

Proposition 3. $\mathbf{PMCSPACE}^{\star}_{\mathcal{D}}$, $\mathbf{NPMCSPACE}^{\star}_{\mathcal{D}}$, $\mathbf{EXPMCSPACE}^{\star}_{\mathcal{D}}$, *and* $\mathbf{NEXPMCSPACE}^{\star}_{\mathcal{D}}$ *are all closed under polynomial-time reductions.*

Proof. Let $L \in \mathbf{PMCSPACE}^{\star}_{\mathcal{D}}$ and let M be the Turing machine constructing the family $\mathbf{\Pi}$ such that $L = L(\mathbf{\Pi})$. Let L' be reducible to L via the polynomial-time computable function f.

Now let M' be the following Turing machine: on input x of length n compute $f(x)$; then behave like M on input $f(x)$, thus constructing $\Pi_{f(x)}$. Since $|f(x)| \leq \mathrm{poly}(n)$, M' operates in polynomial time and $\Pi_{f(x)}$ in polynomial space; but then $\mathbf{\Pi}' = \{\Pi_{f(x)} \mid x \in \Sigma^{\star}\}$ is a polynomially semi-uniform family of P systems deciding L' in polynomial space. Thus $L' \in \mathbf{PMCSPACE}^{\star}_{\mathcal{D}}$.

The proof for the three other classes is analogous.

Proposition 4. $\mathbf{MCSPACE}^{\star}_{\mathcal{D}}(f)$ *is closed under complement for each function* $f\colon \mathbb{N} \to \mathbb{N}$.

Proof. Simply reverse the roles of objects *yes* and *no* in order to decide the complement of a language.

Proposition 5. *For each function* $f\colon \mathbb{N} \to \mathbb{N}$

$$\mathbf{MCSPACE}^{\star}_{\mathcal{D}}(f) \subseteq \mathbf{MC}^{\star}_{\mathcal{D}}\big(2^{O(f)}\big)$$
$$\mathbf{NMCSPACE}^{\star}_{\mathcal{D}}(f) \subseteq \mathbf{NMC}^{\star}_{\mathcal{D}}\big(2^{O(f)}\big).$$

Proof. Let $L \in \mathbf{MCSPACE}^{\star}_{\mathcal{D}}(f)$ be decided by the semi-uniform family $\mathbf{\Pi}$ of recognizer P systems in space f; let $\Pi_x \in \mathbf{\Pi}$ with $|x| = n$ and let $\mathcal{C}$ be a configuration of Π_x. Then $\mathcal{C}$ can be described with a string of length at most $k \cdot f(n)$ over a finite alphabet, say with $b \geq 2$ symbols, and there are less than $b^{k \cdot f(n)+1}$ such strings. Since Π_x is a recognizer P system, by definition every computation halts: then it must halt within $b^{k \cdot f(n)+1}$ steps in order to avoid repeating a previous configuration (thus entering an infinite loop). This number of steps is $2^{O(f)}$.

The same proof also works in the non-confluent case (only the acceptance criterion is different).

5 Space Complexity of P Systems with Active Membranes

In this section we provide a brief review of part of the ample literature on complexity results about P systems with active membranes; our aim is to analyze existing polynomial-time solutions to hard computational problems in order to obtain space complexity results.

We first consider the class of P systems with active membranes which do not make use of membrane division rules, usually denoted by $\mathcal{NAM}$. It is a well known fact that such P systems are able to solve only problems in $\mathbf{P}$ (the so-called Milano theorem [11]); on the other hand, they can be used to solve *all* problems in $\mathbf{P}$ with a minimal amount of space, when a semi-uniform construction is considered:

Proposition 6. $\mathbf{P} \subseteq \mathbf{MCSPACE}^{\star}_{\mathcal{NAM}}(O(1))$.

Proof. Let $L \in \mathbf{P}$. Then there exists a deterministic Turing machine M deciding L in polynomial time. Now consider the family of P systems $\mathbf{\Pi} = \{\Pi_{no}, \Pi_{yes}\}$, where Π_{no} (resp. Π_{yes}) is the following trivial P system with active membranes:

- the membrane structure consists of the skin only, labeled by h;
- in the initial configuration, exactly one object a is located inside the skin;
- the only rule is $[a]^0_h \to [\,]^0_h\ no$ (resp. $[a]^0_h \to [\,]^0_h\ yes$).

It is clear that such P systems halt in one step and that the space they require is independent of the size of the instance they decide.

The family of P systems $\mathbf{\Pi}$ can be constructed in a semi-uniform way in order to decide L by a deterministic Turing machine which first simulates M (it can do so, since M operates in polynomial time), then outputs one of Π_{yes}, Π_{no} according to the result (acceptance or rejection, respectively).

One of the most powerful features of P systems with active membranes is the possibility of creating an exponential workspace in polynomial time by means of elementary membrane division rules; we denote the class of such P systems by $\mathcal{EAM}$. This feature was exploited for solving $\mathbf{NP}$-complete problems in polynomial (often even linear) time. In terms of space complexity, this can be stated as follows:

Proposition 7. $\mathbf{NP} \cup \mathbf{coNP} \subseteq \mathbf{EXPMCSPACE}^{\star}_{\mathcal{EAM}}$.

Proof. In [11] a polynomial-time semi-uniform solution to SAT is described; the number of membranes and objects required is exponential with respect to the length of the Boolean formula. The result then follows from closure under reductions and complement of $\mathbf{EXPMCSPACE}^{\star}_{\mathcal{EAM}}$.

This result can be improved when the use of non-elementary membrane division rules is allowed; indeed, all problems in **PSPACE** can be solved by such class of P systems with active membranes, denoted by $\mathcal{AM}$.

Proposition 8. $\mathbf{PSPACE} \subseteq \mathbf{EXPMCSPACE}^{\star}_{\mathcal{AM}}$.

Proof. In [10] a polynomial-time uniform solution to QBF (also known as QSAT), the canonical **PSPACE**-complete problem, is described; the space required by each P system is still exponential, and the result follows from the closure properties.

In [1] a *uniform* solution for the same problem was achieved, with the same space requirements; this provides a tighter upper bound to **PSPACE**:

Proposition 9. $\mathbf{PSPACE} \subseteq \mathbf{EXPMCSPACE}_{\mathcal{AM}}$.

Since standard P systems with active membranes are very powerful when division rules are allowed, but very weak otherwise, another line of research involves removing some other features, such as polarizations. Polarizationless P systems with active membranes have been proved able to solve QSAT uniformly in polynomial time by making use of both elementary and non-elementary division rules [2]. Since the space requirements are once again exponential, the following result is immediate:

Proposition 10. $\mathbf{PSPACE} \subseteq \mathbf{EXPMCSPACE}_{\mathcal{AM}^0}$, *where* $\mathcal{AM}^0$ *is the class of polarizationless P systems with active membranes and both kinds of division rules.* □

6 Open Problems

In P systems with active membranes, division rules are usually exploited by producing an exponential number of membranes in linear time, which then evolve in parallel; for instance, several solutions to **NP**-complete problems explore the full solution space (e.g., generating every possible truth assignment and then checking whether one of them satisfies a Boolean formula). It appears that membrane division may become much less useful when a polynomial upper bound on space is set; or, in other words,

Conjecture 1. The three classes $\mathbf{PMCSPACE}^{\star}_{\mathcal{NAM}}$, $\mathbf{PMCSPACE}^{\star}_{\mathcal{EAM}}$ and $\mathbf{PMCSPACE}^{\star}_{\mathcal{AM}}$ coincide.

An idea which might be useful in proving this conjecture is pre-computing the "final" membrane structure (which is obtained via division rules) during the construction phase. While this is straightforward when considering membrane divisions which always occur, the matter might be much more difficult in the case of "conditional" division (i.e., division rules are applied only when certain conditions are met) or when the P system exhibits a recurring behavior (e.g., a membrane divides, then one of the two copies is dissolved, and the process is repeated continuously).

Another interesting problem involves the relations between time and space complexity classes for P systems with active membranes. We know that Turing machines, once a polynomial space bound is fixed, are able to solve more problems in exponential time than in polynomial time (at least when $\mathbf{P} \neq \mathbf{PSPACE}$ is assumed). This fact has not been investigated yet in the setting of membrane computing, as all solutions to decision problems presented until now (up to the knowledge of the authors) require only a polynomial amount of time. Formally, the question we pose is the following:

Problem 1. Is $\mathbf{PMC}^{\star}_{\mathcal{D}} \neq \mathbf{PMCSPACE}^{\star}_{\mathcal{D}}$ for any class of P systems $\mathcal{D}$ among $\mathcal{NAM}, \mathcal{EAM}, \mathcal{AM}$? That is, do problems which can be solved in polynomial space but not in polynomial time exist?

Another important property of traditional computing devices is described by Savitch's theorem: nondeterministic space-bounded Turing machines can be simulated deterministically with just a polynomial increase in space requirements, and as a consequence $\mathbf{PSPACE} = \mathbf{NPSPACE}$ holds. The proof does not appear to be transferable to P systems in a straightforward way; nonetheless, an analogous result might hold even in this setting:

Problem 2. Does $\mathbf{PMCSPACE}^{\star}_{\mathcal{D}} = \mathbf{NPMCSPACE}^{\star}_{\mathcal{D}}$ hold for any class of P systems $\mathcal{D}$ among $\mathcal{NAM}, \mathcal{EAM}, \mathcal{AM}$?

The classes of P systems with active membranes we have considered in all the previous problems are only defined according to which kinds of membrane division rules are available (none, just elementary or both elementary and non-elementary). The same questions may be also worth posing about other restricted classes, such as P systems without object evolution or communication [12, 4], P systems with division but without dissolution, or even purely communicating P systems, with or without polarizations.

Finally, we feel that the differences between P systems and traditional computing devices deserve to be investigated for their own sake also from the point of view of space-bounded computations. We formulate this as an open-ended question:

Problem 3. What are the relations between space complexity classes for P systems and traditional ones, such as **P**, **NP**, **PSPACE**, **EXP**, **NEXP**, and **EXPSPACE**?

References

1. A. Alhazov, C. Martín-Vide, L. Pan: Solving a PSPACE-complete problem by recognizing P systems with restricted active membranes. *Fundamenta Informaticae*, 58, 2 (2003), 67–77.
2. A. Alhazov, M.J. Pérez-Jiménez: Uniform solution of QSAT using polarizationless active membranes. *Machines, Computations, and Universality, 5th International Conference, MCU 2007* (J. Durand-Lose, M. Margenstern, eds.), Orléans, France, LNCS 4664, Springer, 2007, 122–133.
3. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez, F.J. Romero-Campero: P systems with active membranes, without polarizations and without dissolution: A characterization of P. *Unconventional Computation* (C. Calude, M.J. Dinneen, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, eds.), Sevilla, Spain, LNCS 3699, Springer, 2005, 105–116.
4. A. Leporati, C. Ferretti, G. Mauri, M.J. Pérez-Jiménez, C. Zandron: Complexity aspects of polarizationless membrane systems. *Natural Computing*, Special issue devoted to IWINAC 2007, to appear.
5. Gh. Păun: P-systems with active membranes: attacking NP complete problems. *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference* (I. Antoniou, C. Calude, M.J. Dinneen, eds.), Brussel, Belgium, 13–16 December 2000, Springer, 2001.
6. Gh. Păun: *Membrane Computing. An Introduction*. Springer-Verlag, 2002.
7. M.J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini: Complexity classes in models of cellular computing with membranes. *Natural Computing*, 2, 3 (2003), 265–285.
8. M.J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini: The P versus NP problem through cellular computing with membranes. *Aspects of Molecular Computing* (N. Jonoska, Gh. Păun, G. Rozenberg, eds.), LNCS 2950, Springer, 2004, 338–352.
9. A.E. Porreca, G. Mauri, C. Zandron: Complexity classes for membrane systems. *RAIRO Theoretical Informatics and Applications*, 40, 2 (2006), 141–162.
10. P. Sosík: The computational power of cell division in P systems: Beating down parallel computers? *Natural Computing*, 2, 3 (2003), 287–298.
11. C. Zandron, C. Ferretti, G. Mauri: Solving NP-complete problems using P systems with active membranes. *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference* (I. Antoniou, C. Calude, M.J. Dinneen, eds.), Brussel, Belgium, 13–16 December 2000, Springer, 2001.
12. C. Zandron, A. Leporati, C. Ferretti, G. Mauri, M.J. Pérez-Jiménez: On the computational efficiency of polarizationless recognizer P systems with strong division and dissolution. *Fundamenta Informaticae*, 87, 1 (2008), 79–91.

Parallel Graph Rewriting Systems

Dragoş Sburlan

Ovidius University
Faculty of Mathematics and Informatics
Constantza, Romania
dsburlan@univ-ovidius.ro

1 Introduction

Nowadays an increasing interest regards the study of the development of biological systems in which more species of individuals interact (usually to perform a certain global task). Research ranging from completely different areas like the study of metapopulations (the study of groups of spatially separated populations of the same species which live in fragmented habitats and interact at some level) and HIV infections was done in essentially the same manner. Traditionally, such studies were done by employing continuous models where (partial) differential equations were used to capture the dynamics of these systems.

Currently, the usage of discrete models where the system dynamics is captured from the collective actions of individual entities has been shown to be a promising choice. This is based on the fact that living organisms are spatially discrete and the individuals occupy particular localities at a given time. The interactions between individuals are strongly connected with their neighborhood relations.

While characterizing these facts a basic issue regards the way the space is represented. Simple models that involve no detailed spatial structure are in general analytically easily solvable. However, as the complexity of the reaction-diffusion dynamics grows, the models based on partial differential equations become intractable to be analyzed.

On the other hand, integrating within the model a detailed spatial structure (as cellular automata models do, for instance) the setback comes in general from the impossibility to analyze the models except only by performing simulations. Although such models have much greater biological reality, they suffer from the difficulty of generalization (hence of finding the exact behavior). This is especially important while formulating some practical testable predictions regarding a given model.

P systems are formal computing devices that were initially inspired and abstracted from the cell functioning (see [4]). In general, P systems make use of multisets to represent the computational support. These multisets are placed in-

side the membranes which in their turn are disposed in some hierarchical tree structure. The (maximal) parallel applications of some multiset rewriting rules (particular to each membrane) were used to process the multisets.

Although these formal systems were extensively studied with respect to their computational power and efficiency, while representing some biological processes many difficulties arise. Representing the data support as multisets essential simplifies the structure of the environment and of the individuals from within (the neighboring relations between the individuals are completely ignored), the focus being over the system dynamics. However, in this case, two main assumptions are considered: the environment is homogeneous so that the concentration of the individuals do not change with respect to space and the number of individuals of each species in the environment is "adequately" large (hence the concentration of the individuals might be assumed to vary continuously over the time). Moreover, the rules that describe the interactions between the individuals are assumed to be executed in a maximal parallel manner and governed by a global clock that marks equal steps.

Even if all these simplifications are useful while defining a computing formal framework, they are questionable if the aim is to model and simulate actual biological systems. This is way many new features that are meaningful to biologists were added to the original paradigm in order to extend its functionality and versatility for modeling.

In order to cope with these issues, probabilistic/stochastic P systems were introduced (see [2], [6], [1]). In general, the main idea was to associate to the rules some weights describing how they should be applied at a given moment. For a particular rule, the weight gives the susceptibility of its execution at certain instant. Hence, applying this principle to all interaction rules it sets up more realistic bounds of the nondeterministic application of the rules. The ultimate goal of this approach is to integrate the structural and dynamical characteristics of a real biosystem into the way the rules of the model are selected to be applied and executed (preserving at the same time the unstructured computing support). Although this method has in general good simulation time complexity it is inadequate if the interacting species are poorly represented, when there exist many "inactive" individuals (that are not the subject of any rule) with respect to the entire population of individuals, or when the environment is not homogeneous.

2 Preliminaries

We assume the reader familiar with the basic notions of P systems (one can consult [4] for more details), so that here we only recall some notions regarding the abstract rewriting systems on multisets. ARM systems represent a variant of P systems which was proposed in order to perform simulations of some bio-chemical processes. Later on, due to its modeling flexibility, it was used to study some symbiotic mechanism of an ecological system and even for proposing a novel theory of evolution.

ARMS is a stochastic model that uses multisets to represent the bio-chemical support. Multiset rewriting rules are used to describe the bio-chemical reactions. As opposed to the classical definition of P systems where the rules are applied in a nondeterministic, maximal parallel manner and with competition on the objects composing the multisets, in ARMS the rules obey the Mass Action Law where the frequency of a reaction follows the concentration of bio-chemicals and a rate constant. Consequently, the rules to be applied are chosen probabilistically from the rules set and each probability is given by the ratio of the total number of colliding chemicals of a reaction to the sum of the total number of colliding chemicals of every reactions in the rule; the applications of the rules remain parallel and with competition on the objects.

More formally, an ARM system is a construct $\Pi = (O, w, R)$ where O is the alphabet of objects, w represents the multiset of objects at the beginning of computation, and R is a set of multiset rewriting rules of type $u \xrightarrow{k} v$, where $u \in O^+$, $v \in O^*$, and $k \in \mathbb{R}$ is the rate constant of the rule.

For example, in case of a cooperative rule of type $r_i : aA + bB \rightarrow cC + dD$ and a given multiset of objects M, the probability of rule execution is defined as $Prob(r_i) = \frac{k_i M_A^a M_B^b}{R}$, where k is the rate constant (determined experimentally) and R is a coefficient for normalizing the probabilities ($\sum_i Prob(r_i) = 1$). Similarly, probabilities can be defined for any type of rules.

The system Π starts to evolve from the initial configuration (represented by w) by applying the rules in parallel, randomly selecting the rules but according with the probabilities computed as above. Π is governed by an universal clock that marks equal time units.

We have run more tests using an ARM system Π where $O = \{A, B, C, D, X, F\}$, and the set of rules R is given bellow:

$$
\begin{array}{ll}
r_1 : AB \xrightarrow{k_1} X & r_8 : F \xrightarrow{k_8} F \\
r_2 : AC \xrightarrow{k_2} X & r_9 : A \xrightarrow{k_9} A \\
r_3 : BD \xrightarrow{k_3} X & r_{10} : B \xrightarrow{k_{10}} B \\
r_4 : CD \xrightarrow{k_4} F & r_{11} : C \xrightarrow{k_{11}} C \\
r_5 : FX \xrightarrow{k_5} FF & r_{12} : D \xrightarrow{k_{12}} D \\
r_6 : FA \xrightarrow{k_6} FF & r_{13} : X \xrightarrow{k_{13}} X \\
r_7 : FB \xrightarrow{k_7} FF &
\end{array}
$$

The initial configuration of Π was $w = A^{250}B^{250}C^5D^5$ and in our tests we used several values for k_i, $1 \leq i \leq 13$. The system attempts to simulate the behavior of some interacting individuals, represented here as the objects A, B, C, and D, sharing the same environment. In addition, the individuals corresponding to the objects C and D (which are much less than the individuals corresponding to the objects A and B) share a localized patch in the environment. Thus, we assumed the environment not to be homogeneous.

If at least once the objects C and D interact (i.e., the rule $CD \xrightarrow{k_4} F$ is applied) they will produce an object F which will trigger the conversion of all existing objects in the multiset into F (the rules r_5, r_6, and r_7). The rest of the rules (r_8 till r_{13}) are used to slow down the rate of parallelism.

Since we have assumed the existence of a patch in the environment of individuals corresponding to objects C and D, then we could make another further assumption: if the patch is "large enough" so that there exists at least two individuals C and D which are not interacting initially with the individuals A and B, then there exists a "significant" probability that the rule $CD \xrightarrow{k_4} F$ is executed. While using multisets to represent the individuals in the environment we lose the structure, hence when simulating such systems we actually have to relay on the probabilities of the executions of the rules (which in their turn depend on some constants experimentally determined). In Figure 1, one can notice the different behaviors of the same system and they are related to the usage of such probabilities. The charts shown on the right hand side present a simulation when the rule $CD \xrightarrow{k_4} F$ was executed at least once, while the charts on the right hand side present a simulation when the rule $CD \xrightarrow{k_4} F$ was not executed at all. Although the model considered is very simple a similar situation might happen when representing some complex systems. Even more, such situations might emerge during the system evolution and sudden shifts in the behavior might arise from some minor changes in the circumstances; if this is the case, then it would make almost impossible the precise identification of the rate constants associated to the rules.

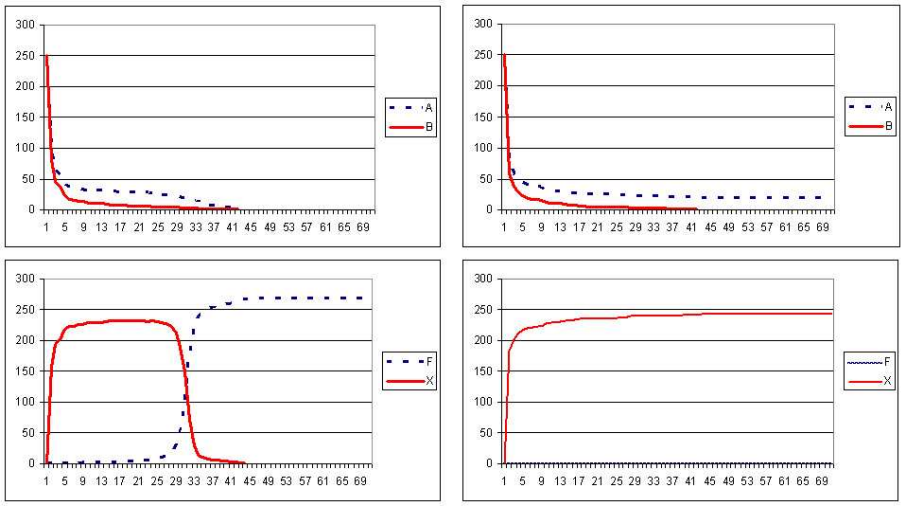


Fig. 1. Two runs of system Π. The results are presented on columns and they show the different behaviors of the same system when some minor changes in the circumstances happen.

Besides all of these issues, if the number of objects in the model decreases under a certain limit, the usage of probabilities to specify the way the rules are applied becomes inadequate.

3 PGR Systems

Aiming to tackle the mentioned issues, in this section we introduce a new model for simulating bio-systems composed by interacting individuals of various species in a given environment.

Denote by C the set of species in an environment represented here as a metric space (for simplicity, let $\mathbb{R}^k, k \geq 2$, be the environment). Let $V \subseteq L \times C$ be the finite set of labeled individuals in the environment (L denotes a finite set of labels that uniquely identify the individuals in the environment). In addition, let $f : V \to \mathbb{R}^k$, $k \geq 2$, be a bijective mapping; for a node $v = (n, l) \in V$, the value $h(v)$ denotes the position of the individual v in the environment. In addition let $r > 0$, $r \in \mathbb{R}$, be a positive constant.

Based on above definition one can represent the environment and the individuals from within as a graph $G_0 = (V_0, E_0)$ where $V_0 = V$ and the set of edges is constructed as follows: for two nodes $v_1, v_2 \in V$, if $h(v_1)$ belongs to the open ball centered in $h(v_2)$ and with radius r (i.e., $h(v_1) \in B(h(v_2), r)$) then there exists an edge from v_1 to v_2.

For simplicity we assume that G_0 is connected, that is, for any two nodes $m, n \in V$ there exists a sequence $m = v_0, v_1, \ldots, v_t = n \in V$ such that $h(v_i) \in B(h(v_{i-1}), r)$, for $1 \leq i \leq t$.

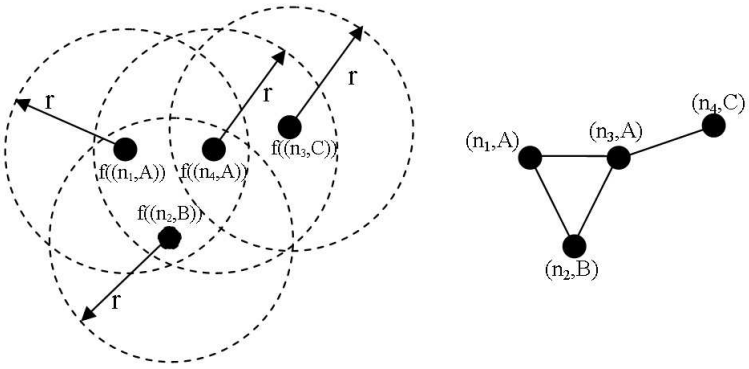


Fig. 2.

Motivated by these facts we can introduce the following model. A *parallel graph rewriting system* (in short, a PGR system) is a construct $\Gamma = (C, G_0, R)$ where

• $C = \{c_1, \ldots, c_k\}$ is a finite set of symbols;

• $G_0 = (V_0, E_0)$ is the *initial global graph* – a connected graph such that $V_0 \subseteq L \times C$ is a set of labeled nodes and $E_0 \subseteq V_0 \times V_0$ is a set of edges between nodes from V_0;

• R is a finite set of *graph rewriting rules.*

A graph rewriting rule $r \in R$ is of the following type:

$$r = (G_1 = (V_1, E_1), G_2 = (V_2, E_2)),$$

where $V_i \subseteq L_i \times C$, $E_i \subseteq V_i \times V_i$, $i \in \{1, 2\}$. The graphs G_1 and G_2 are connected graphs; G_1 represents the neighboring relations between the individuals that are required for an interaction to take place and G_2 represents the output of an actual interaction between individuals represented in G_1. In addition we will assume that G_1 and G_2 are not arbitrary graphs, but rather they obey some physical constraints: any node from G_1 and G_2 cannot be the subject of more than a constant $t_r \in \mathbb{N}$ edges – a condition that assume the nonexistence of more than t_r individuals in an open ball of radius r.

A graph rewriting rule $r = (G_1, G_2) \in R$ can be applied on a graph G if G_1 is *label isomorphic* with some subgraph $G_s = (V_s, E_s)$ of G, that is, there exists a bijective mapping $h : V_1 \to V_s$ such that $h((m, c)) = (n, c)$ and $h^{-1}((n, c)) = (m, c)$, where $(m, c) \in V_1$, $(n, c) \in V_s$ and such that any two nodes $u, v \in V_s$ are adjacent in G_s if and only if $h(u)$ and $h(v)$ are adjacent in G_1 (see Figure 3).

In other words, a graph rewriting rule r can be applied on G iff the left-hand side rule's graph is "contained" in G both as layout and as corresponding node labels (via an edge/label-preserving bijection).

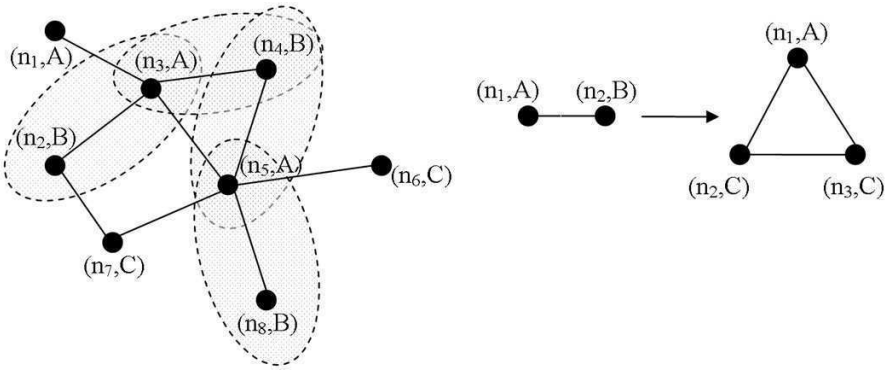


Fig. 3. A graph $G = (V, E)$ denoting the computing support and a graph rewriting rule $r = (G_1, G_2)$. The sites where the rule r can be applied in G are explicitly figured. If $G_s = (V_s = \{(n_4, B), (n_5, A)\}, E_s = \{((n_4, B), (n_5, A))\})$ then G_1 is label isomorphic with G_s. The neighborhood set of degree $k = 1$ of G_s is $B_1 = \{(n_3, A), (n_6, C), (n_7, C), (n_8, B)\}$.

Applying a rule r over G follows the following steps:

- eliminate G_s from G (all the nodes from V_s are eliminated from V; all the edges of the type (v, v_s), $v \in V$, $v_s \in V_s$ are deleted from E);
- add G_2 to G (some relabeling of the nodes from G_2 is required in order to avoid duplicates of nodes at multiple application of r). All the (relabeled) nodes and edges of G_2 are added to G;
- add a set of edges from some nodes of V_2 to some nodes of $V \setminus (V_s \cup V_2)$. The edges are established as described below.

For the graph G_s let us define the *neighborhood set* of degree k

$$B_k = \{v \in V \setminus V_s \mid \text{ there exists a path of length less or equal with } k \text{ from } v \text{ to a node } u \in V_s\}.$$

As we mentioned above, the output of the application of a rule consists of new individuals that, by hypothesis, at the moment of their apparition it is assumed to belong to the same vicinity. How big is that vicinity and how the new individuals are related to the rest depend on many factors among which we just mention the type of the rule and the environment. Consequently, in our framework, the set B_k is useful when defining the new neighborhood relations triggered by the application of a rule. By some straightforward physical arguments, the output graph G_2 of the rule r is likely to be "connected" to G via the nodes from B_k. However, for simplicity, we will consider the neighborhood set of degree 1 in our simulations.

Let $\overline{E} = \{(n_1, n_2) \in E \mid n_1 \in B_1, n_2 \in V_s\}$. Then, a number equals with $card(\overline{E})$ of random edges from the nodes of G_2 to the nodes from B_1 are added to G but such that any node considered is not the subject of more than t_r edges.

Starting from the initial configuration (the initial global graph G_0), the system evolves according to the rules from R and the current labeled graph in a non-deterministic parallel manner (but not necessarily maximal). The labeled graph of Γ at any given moment constitutes the configuration of the system at that moment. For two configurations G_A and G_B we can define a transition from G_A to G_B if we can pass from G_A to G_B by applying rules from R.

Determining whether two graphs are isomorphic is referred to as the *graph isomorphism problem*. Although this problem belong to **NP** it is neither known to be solvable in polynomial time nor it is **NP**-complete. A generalization of this problem (that is used in our formalism) is the subgraph isomorphism problem which is **NP**-complete; hence the known deterministic algorithms for this problem are exponential.

Remark 1. There is a physical motivation to assume that after applying a rule of the system, the newly produced objects (that correspond to the output nodes of the rule) belongs to the same vicinity, hence the left hand side graph of any rule should be complete.

Remark 2. For a given PGR system, as much as the radius r grows (hence the number of edges in the initial global graph is close to $\frac{n(n-1)}{2}$ where n is the number of the nodes, that is, the initial global graph is "almost" complete) and the degrees

of the neighboring sets grow as well, the result of a simulation is similar with one obtained using parallel multiset rewriting. This is because multisets can be seen in our formalism as complete graphs, hence any individual in the system is in a neighboring relation with any other individual (hence, they can interact if proper rules exist).

4 PGRS Simulator and a Test Case

The simulator implements the model introduced in Section 3. Its main characteristics regard the definition of the rules set by using an XML file, and the possibility to save/load intermediate configurations. The simulator is written in Java language hence it benefits of cross-platform compatibility, parallelism, and possibility to distribute the computational effort.

The task that has the most computational resource consumption is the subgraph isomorphism problem which is addressed whenever a rule $r = (G_1, G_2)$ is selected for application and the set S of all the subgraphs of the global graph that are isomorphic with G_1 has to be determined. Even more, whenever a subgraph $\overline{G} \in S$ is selected to be rewritten by r, a run through all the elements of S has to be performed in order to eliminate those subgraphs that have some nodes from $\overline{G}$. Considering all these matters for all the rules from the rule set and a relatively small global graph, the overall time complexity for simulating just one computational step is exponential. Nevertheless, if the left hand side graphs of the rules from the rule set are very simple (i.e., less than 4 nodes) and the global graph contains at most hundreds of nodes, the problem is feasible. Moreover, taking into account that the problem can be easily parallelized one can divide the the problem into smaller instances and distribute them over a network.

Let us consider the following PGR system $\Gamma = (C, G_0, R)$ where
- $C = \{A, B, C, D, F, X\}$,
- $R = \{r_1, r_2, r_3, r_4, r_5, r_6, r_7, r_8\}$ is defined as follows:

In our tests, the initial global graph G_0 was build to obeying some properties. First of all, a random graph G_0' was generated and this graph contains 500 nodes labeled only with A and B (the apparition of these labels are equally probable) and 2000 edges. A second graph G_0'' was generated and this graph contains 10 nodes labeled only with C and D (the apparition of these labels are equally probable) and 30 edges. Finally, G_0' and G_0'' were merged together in order to form G_0 by connecting 10 randomly chosen nodes from G_0' with 10 randomly chosen nodes from G_0''.

We ran the simulator for 100 times, considering for each run a new initial global graph generated as above. In Figure 4 are represented the minimal and the maximal values at each step of the simulation for the objects A, B, C, and D. Any particular simulation graphic from our test case lay between the boundaries established.

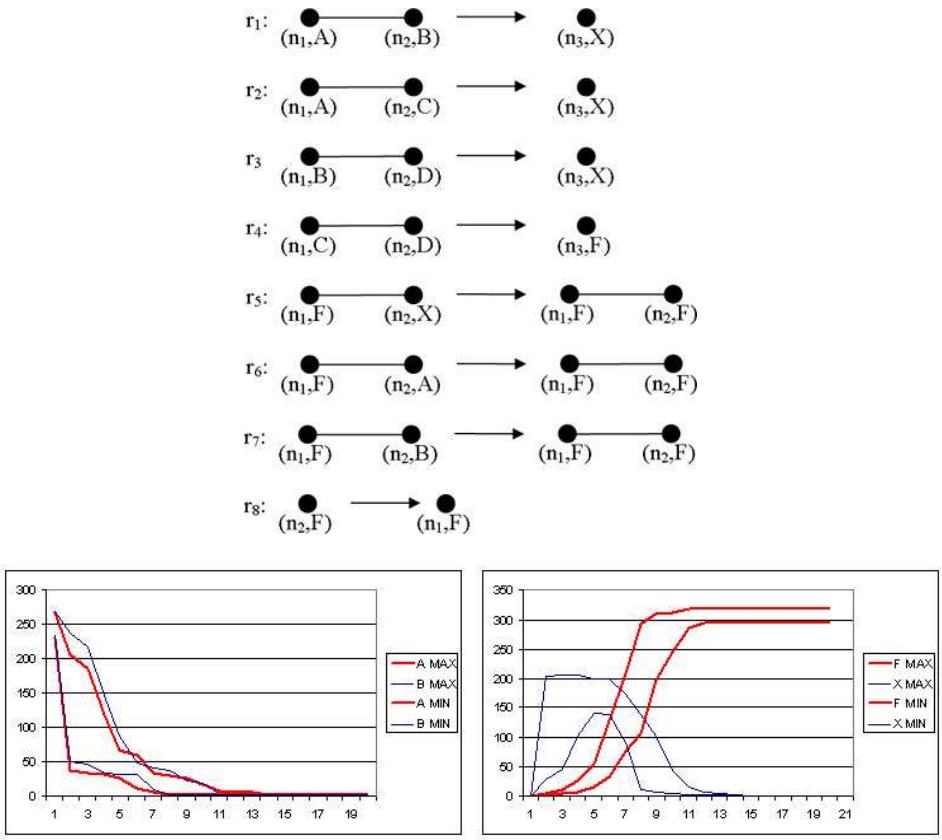


Fig. 4. The results of 100 simulations of different GPR systems but having the same properties. The minimal and maximal obtained values are explicitly marked.

5 Conclusions

Simulations performed using PGR systems in some cases give more accurate answers than ARMS simulations because they explicitly use the spatial distribution of individuals (hence the neighborhood relations can be extensively expressed). However the price to pay while using PGR systems regards the computational effort which in their case is exponential as time complexity. Nevertheless, for some cases when the number of interacting individuals in the environment is small and they are not dense, the PGR systems might be useful for performing simulations.

In order to handle these issues, a hybrid system combining features from the ARM and PGR systems might be proposed. Two directions could be taking into account:

- one can use alternatively an ARMS-type simulation whenever the number of individuals from all the species is large and a PGRS-type simulation whenever

the number of individuals from certain species goes below some threshold; in this case the newly obtained system uses in a more careful manner the probabilities for the rules executions.

- one can use in parallel an ARMS-type simulation over a multiset of many individuals and a PGRS-type simulation on relatively small instances of graphs. Then one can consider a time sequence and at each moment in the sequence one can merge the ARMS configuration with the multiset of labels of the nodes from the graph (or one can exchange some data between these simulations). In this way, the newly obtained hybrid systems become more robust against some unexpected changes in the behavior (which might be triggered by some minor changes).

References

1. D. Besozzi, P. Cazzaniga, D. Pescini, G. Mauri: Modelling metapopulations with stochastic membrane systems. *BioSystems*, 91 (2008), 499–514.
2. M. Cavaliere, I.I. Ardelean: Modeling biological processes by using a probabilistic P system software. *Natural Computing*, 2, 2 (2003), 173–197.
3. D.T. Gillespie: Exact simulation of coupled chemical reactions. *J. Physical Chemistry*, 81 (1977), 2340–2361.
4. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
5. D. Pescini, D. Besozzi, G. Mauri, C. Zandron: Dynamical probabilistic P systems. *Int. J. Found. Comput. Sci.*, 17, 1 (2006), 183–204.
6. Y. Suzuki, H. Tanaka: Modelling p53 Signaling Pathways by Using Multiset Processing. *Applications of Membrane Computing* (G. Ciobanu, Gh. Păun, M. Pérez-Jiménez, eds.), Springer, Berlin, 2006.
7. Y. Suzuki, J. Takabayashi, H. Tanaka: Investigation of a tritrophic system in ecological systems by using an artifcial chemistry. *J. Artif. Life Robot.*, 6 (2002), 129–132.
8. Y. Suzuki, S. Tsumoto, H. Tanaka: Analysis of cycles in symbolic chemical system based on abstract rewriting systems on multisets. *Proc. Intern. Conf. on Artificial Life 5* (Alife 5), 1996, 482–489.

About the Efficiency of Spiking Neural P Systems

Jun Wang[1], Tseren-Onolt Ishdorj[2], Linqiang Pan[1,3]

[1] Key Laboratory of Image Processing and Intelligent Control
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, People's Republic of China
junwangjf@gmail.com, lqpan@mail.hust.edu.cn
[2] Computational Biomodelling Laboratory
Åbo Akademi University
Department of Information Technologies
20520 Turku, Finland
tishdorj@abo.fi
[3] Research Group on Natural Computing
Department of CS and AI, University of Sevilla
Avda Reina Mercedes s/n, 41012 Sevilla, Spain
Institute of Mathematics of the Romanian Academy

Summary. Spiking neural P systems were proved to be Turing complete as function computing or number generating devices. Moreover, it has been considered in several papers that spiking neural P systems are also computationally efficient devices working in a non-deterministic way or with exponential pre-computed resources. In this paper, neuron budding rules are introduced in the framework of spiking neural P systems, which is biologically inspired by the growth of dendritic tree of neuron. Using neuron budding rules in SN P systems is a way to trade space for time to solve computational intractable problems. The approach is examined here with a deterministic and polynomial time solution to SAT problem without using exponential pre-computed resources.

1 Introduction

Computational efficiency of spiking neural P systems (in short, SN P systems) has been investigated in a series of works [1, 6, 8, 9, 10], recently. In the framework of SN P systems, most of the solutions to computationally hard problems are based on non-determinism [9, 10, 11] or exponential pre-computed resources [1, 6, 8, 7]. The present paper proposes a rather different way to address this issue in a sense that no pre-computed resource is used but it is computed by a SN P system.

It has been claimed in [11] that an SN P system of polynomial size cannot solve in a deterministic way in a polynomial time an **NP**-complete problem (unless **P=NP**). Hence, under the assumption that $\mathbf{P} \neq \mathbf{NP}$, efficient solutions to **NP**-complete problems cannot be obtained without introducing features which enhance

the efficiency (pre-computed resources, ways to exponentially grow the workspace during the computation, non-determinism, and so on).

A possibility of using spiking neural P systems for solving computationally hard problems, under the assumption that some (possibly exponentially large) pre-computed resources are given in advance has been presented in [6]. Specially, in [6], a uniform family of spiking neural P systems was proposed which can be used to address the **NP**-complete problems, in particular, to solve all the instances of SAT which can be built using n Boolean variables and m clauses, in a time which is quadratic in n and linear in m.

In the present paper, we continue the study considered in [6] and particularly focus on a possible way to construct an SN P system such that the system can compute the necessary resources (exponentially large work space) to be used in advance by itself. For this purpose, we extend the SN P systems [6] by introducing neuron budding rules. We show that the SN P systems with budding rules can (pre-)compute the exponential work space in polynomial time with respect to the size of the instances of the problem we want to solve, however, the problem is solved too by the same system. All the systems we will propose work in a *deterministic* way.

The biological motivation of the mechanism for expanding the work space (net structure) of SN P systems by introducing neuron budding comes from the growth of dendritic tree of neural cells [15]. The brain is made up of about 100 billion cells. Almost all brain cells are formed before birth. Dendrites (from Greek, tree) are the branched projections of a neuron. The point at which the dendrites from one cell contact the dendrites from another cell is where the miracle of information transfer (communication) occurs. *Brain cells can grow as many as 1 million billion dendrite* connections – a universe of touch points. The greater the number of dendrites, the more information that can be processed. Dendrites grow as a result of stimulation from and interaction with the environment. With limited stimulation there is limited growth. With no stimulation, dendrites actually retreat and disappear. These microscope photographs illustrated in Figure 1 show actual dendrite development. Dendrites begin to emerge from a single neuron (brain cell) developing into a cluster of touch points seeking to connect with dendrites from other cells.

In the framework of SN P systems, the dendrite connection points are considered as abstract neurons and the branches of dendrite tree are consider as abstract synapses. The new connection between dendrites from two different neuron cells is understood as new created synapses. In this way, new neurons and synapses can be produced during the growth of dendrite tree.

The formal definition of neuron budding rule and its semantics will be given in Section 2.

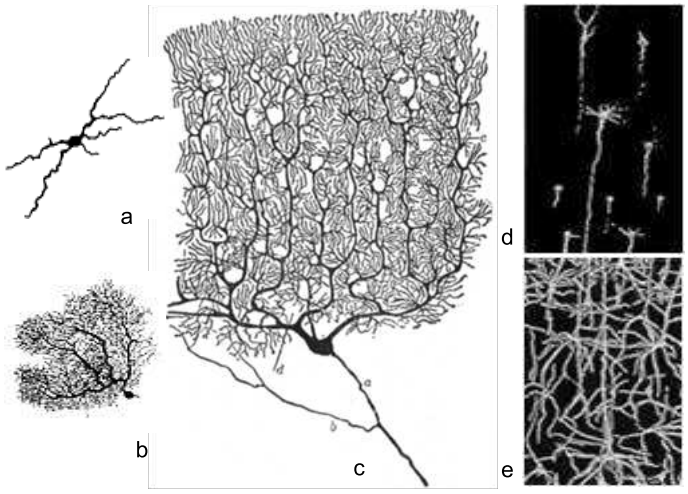


Fig. 1. Growing neuron: a. dendrites begin to emerge from a single neuron, b. developed into a cluster of touch points; c. Ramon y Cajal, Santiago. Classical drawing: Purkinje cell; d. newborn neuron dendrites, e. 3 months later. Photos from Tag Toys [15]

2 SN P systems with neuron budding rules

A *spiking neural P system with neuron budding* of (initial) degree $m \geq 1$ is a construct of the form

$$\Pi = (O, \Sigma, H, syn, R, in, out),$$

where:

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$ is a finite set of initial neurons;
3. H is a finite set of *labels* for neurons;
4. $syn \subseteq H \times H$ is a finite set of *synapses*, with $(i, i) \notin syn$ for $i \in H$;
5. R is a finite set of *developmental rules*, of the following forms:
   (1) *extended firing* (also called *spiking*) rule $[E/a^c \rightarrow a^p; d]_i$, where $i \in H$, E is a regular expression over a, and $c \geq 1$, $p \geq 0$, $d \geq 0$, with the restriction $c \geq p$;
   (2) *neuron budding rule* $x[\ ]_i \rightarrow y[\ ]_j$, where $x \in \{(k, i), (i, k), \lambda\}$, $y \in \{(i, j), (j, i)\}$, $i, j, k \in H$, $i \neq k$, $i \neq j$.
6. $in, out \in H$ indicate the *input* and the *output* neurons of Π.

The way of presentation of SN P system is here slightly different from the usual definition present in the literature, where the neurons presented initially in the system are explicitly listed as $\sigma_i = (n_i, R_i)$, $1 \leq i \leq m$ and R_i are the rules

associated with neuron with label i. In what follows we will refer to neuron with label $i \in H$ also denoting it with σ_i.

If an extended firing rule $[E/a^c \to a^p; d]_i$ has $E = a^c$, then we will write it in the simplified form $[a^c \to a^p; d]_i$; similarly, if a rule $[E/a^c \to a^p; d]_i$ has $d = 0$, then we can simply write it as $[E/a^c \to a^p]_i$; hence, if a rule $[E/a^c \to a^p; d]_i$ has $E = a^c$ and $d = 0$, then we can write $[a^c \to a^p]_i$. A rule $[E/a^c \to a^p]_i$ with $p = 0$ is written in the form $[E/a^c \to \lambda]_i$ and is called *extended forgetting* rule. Rules of the types $[E/a^c \to a; d]_i$ and $[a^c \to \lambda]_i$ are said to be *standard*.

If a neuron σ_i contains k spikes and $a^k \in L(E), k \geq c$, then the rule $[E/a^c \to a^p; d]_i$ is enabled and it can be applied. This means consuming (removing) c spikes (thus only $k - c$ spikes remain in neuron σ_i); 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, etc. 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 (this corresponds to the refractory period from neurobiology), 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). In the step $t + d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t + d + 1$, when the neuron can again apply rules). Once emitted from neuron σ_i, the p spikes reach immediately all neurons σ_j such that there is a synapse going from σ_i to σ_j and which are open, that is, the p spikes are replicated and each target neuron receives p spikes; as stated above, spikes sent to a closed neuron are "lost", that is, they are removed from the system. In the case of the output neuron, p 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. If the rule is a forgetting one of the form $[E/a^c \to \lambda]_i$, then, when it is applied, $c \geq 1$ spikes are removed. When a neuron is closed, none of its rules can be used until it becomes open again.

If a neuron σ_i has only synapse x where $x \in \{(i, k), (k, i), \lambda\}$, $i \neq k$, then rule $x[\,]_i \to y[\,]_j$ is enabled and can be applied, where $y \in \{(i, j), (j, i)\}$. The synapse x describes the interaction environment of neuron σ_i with another neuron. As a result of the rule application, a new neuron σ_j and a synapse y are established provided that they do not exist already; if a neuron with label j does already exist in the system but no synapse of type y exists, then only the synaptic connection y between the neurons σ_i and σ_j is established, no new neuron with label j is budded. We stress here that the application of budding rules depends on the environment of the associated neuron, instead of the spikes contained in the associated neuron; a budding rule can be applied only if the associated neuron has the environment exactly as the rule described; in other words, even if the environment has a proper sub-environment that enables a budding rule, but the whole environment does not enables the budding rule, then the rule cannot be applied. The rules of such type are applied in a maximal parallel way: if the environment of neuron σ_i enables several budding rules, then all these rules are applied; as a result, several new neurons and synapses are produced (which corresponds to have several branches at

a touch point in the dendrite tree). Note that the way of using neuron budding rules is different with the usual way in P systems with cell division or cell creation, where at most one rule division rule or creation rules can be applied to one membrane or one object, respectively.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R *must* be used. If several spiking rules are enabled in neuron σ_i, then only one of them is chosen non-deterministically. If the environment of neuron σ_i enables several budding rules, then all these rules are applied. If both spiking rules and budding rules are enabled in the same step, then one type of rules is chosen non-deterministically. When a spiking rule is used, the state of neuron σ_i (open or closed) depends on the delay d. When a neuron budding rule is applied, at this step the associated neuron is closed, it cannot receive spikes. In the next step, the neurons obtained by budding will be open and can receive spikes.

The *configuration* of the system is described by the topology structure of the system, the number of spikes associated with each neuron, and the *state* of each neuron (open or closed). Using the rules as described above, one 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.

In the following, we give an example to make the usage of budding rules transparent, where neither spike nor spiking rule is of interest.

An example. The system Π_1 has initial topological structure shown in Figure 2(a), and the budding rules $(1,3)[\,]_3 \to (3,4)[\,]_4$, $(1,3)[\,]_3 \to (3,5)[\,]_5$, $(2,1)[\,]_2 \to (6,2)[\,]_6$, $(3,4)[\,]_4 \to (4,7)[\,]_7$ and $(6,2)[\,]_6 \to (6,5)[\,]_5$.

In the initial topological structure, neuron σ_3 has two synapses $(1,3)$ and $(2,1)$, and no other synapses are associated with it; as the environment of neuron σ_3 enables both rules $(1,3)[\,]_3 \to (3,4)[\,]_4$ and $(1,3)[\,]_3 \to (3,5)[\,]_5$, the rules are applied in the maximal parallel application manner. As a result, two new neurons σ_4 and σ_5, and two synapses $(3,4)$ and $(3,5)$ are produced. At the same time, the rule $(2,1)[\,]_2 \to (6,2)[\,]_6$ is applied to neuron σ_2 with a synapse $(2,1)$, thus, neuron σ_6 and synapse $(6,2)$ are produced. The structure is shown in 2(b) after step 1.

At the second step, the rules $(1,3)[\,]_3 \to (3,4)[\,]_4$ and $(1,3)[\,]_3 \to (3,5)[\,]_5$ cannot apply again as the two newly created synapses $(3,4)$ and $(3,5)$ going out from neuron σ_3 have changed the environment of it. Similarly, the rule $(2,1)[\,]_2 \to (6,2)[\,]_6$ cannot be used again. As neuron σ_4 has only synapse $(3,4)$, its environment enables the rule $(3,4)[\,]_4 \to (4,7)[\,]_7$ to be applied to it, then a new neuron σ_7 and a synapse $(4,7)$ are produced. Neuron σ_6 has only synapse $(6,2)$, then rule $(6,2)[\,]_6 \to (6,5)[\,]_5$ is enabled and applied. Since a neuron with label 5 already exist, no new neuron with label 5 is budded instead, a synapse $(6,5)$ to neuron σ_5 from neuron σ_6 is established, this is the principle of neuron budding rules. The corresponding structure is shown in Figure 2(c). Now no rule is enabled by any neuron interaction environment, thus the system halts.

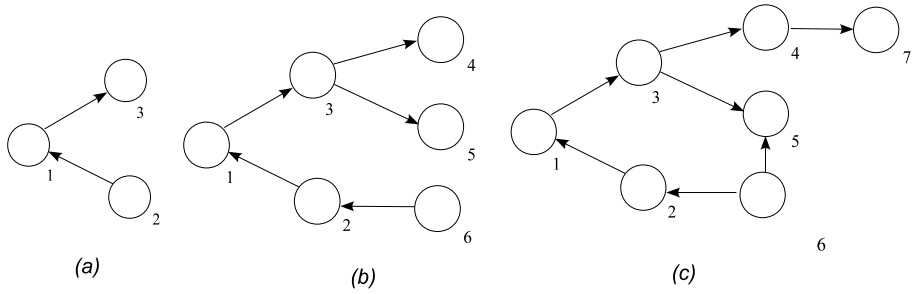


Fig. 2. Structure transition of SN P system Π_1

3 Brief of pre-computed SN P systems solving SAT

As we mentioned in Section 1, a way to solve **NP** hard problems by SN P systems is to assume an exponential working space has been pre-computed in advance, based on that given work space a family of SN P systems solves all the possible instances of the problem in polynomial time, [6].

Let us recall here the basic description of SAT (satisfiability) problem, a well know **NP**-complete problem. An instance of SAT is a propositional formula $\gamma_n = C_1 \wedge C_2 \wedge \cdots \wedge C_m$, expressed in the conjunctive normal form as a conjunction of m clauses, where each clause is a disjunction of literals built using the Boolean variables $x_1, x_2, \ldots, x_n$. An *assignment* of the variables $x_1, x_2, \ldots, x_n$ 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 gives 1 (*true*) as a result.

Let us denote by $\text{SAT}(n, m)$ the set of instances of SAT which have n variables and m clauses. In [6], a uniform family $\{\Pi_{SAT}(\langle n, m\rangle)\}_{n,m\in\mathbf{N}}$ of SN P systems was built such that for all $n, m \in \mathbb{N}$ the system $\Pi_{SAT}(\langle n, m\rangle)$ solves all the instances of $\text{SAT}(n, m)$ in a number of steps which is quadratic in n and linear in m.

Let us first briefly summarize here the overview of the considered system $\Pi_{SAT}(\langle n, m\rangle)$ from [6], and its structure and functioning that solves all the possible instances of $\text{SAT}(n, m)$.

The system structure is composed by $n + 5$ layers, see Figure 3. The first layer (numbered by 0) is composed by a single input neuron, that is used to insert the representation of the instance $\gamma_n \in \text{SAT}(n, m)$ to be solved. Note that layer 1, as well as the subsequent $n - 1$ layers, is composed by a sequence of n neurons, so that the layer contains the representation of one clause of the instance. In layer n, we have got 2^n copies of the subsystem; each subsystem contained in this layer is bijectively associated to one possible assignment to variables $x_1, x_2, \ldots, x_n$. Simply say, the neurons in a subsystem are two types: f and t; the types indicate that the corresponding Boolean variable is assigned with the Boolean values $t(rue)$ or $f(alse)$, respectively. However, the all subsystems of layer n are injectively distin-

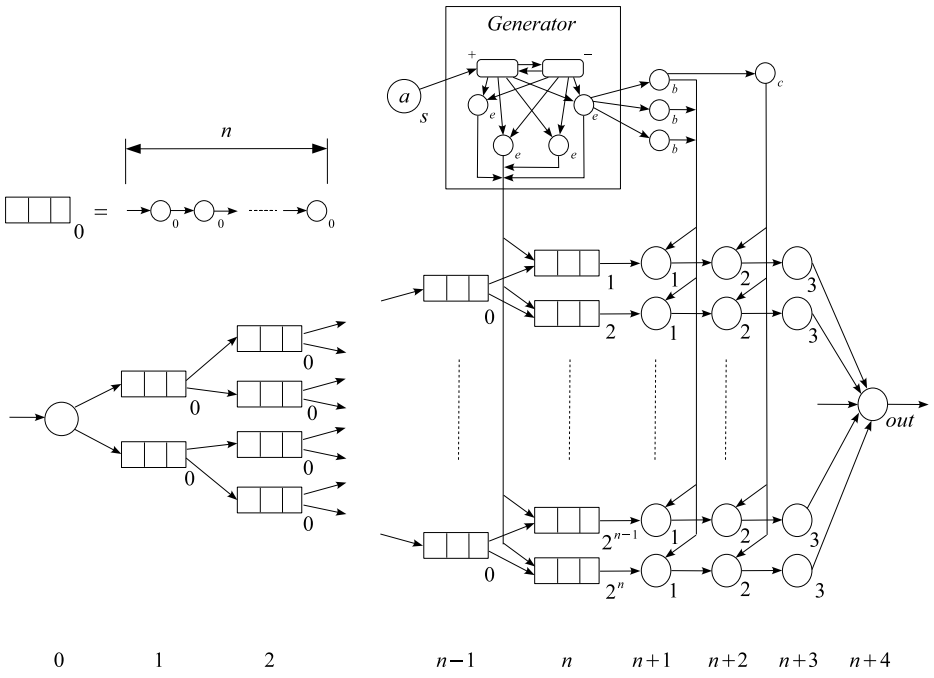


Fig. 3. SN P system structure which solves SAT(m,n). From [6].

guished from each other with respect to the all possible different truth assignments for variables $x_1, x_2, \ldots, x_n$ represented by each subsystem. The subsystems that occur in layer n together with the so called *generator* have a very specific functions such that all possible assignments are tested in here in parallel against the clause. The assignment is performed by sending 3 spikes to all the neurons labelled with t, and 4 spikes to all the neurons labelled with f from the generator. This means that the generator have three synapses going to neurons t and four synapses towards neurons f.

Those assignments that satisfy the clause produce a (single) spike in the corresponding neuron 2 (that occurs in the same row, in layer $n+2$), which is accumulated in the associated neuron 3, that operates like a counter. When the first clause of γ_n has been processed, the second takes place in the subsystems in layer n in n steps, and all possible assignments are tested, etc. When all the m clauses of γ_n have been processed, neurons 3 in layer $n+3$ contain each the number of clauses which are satisfied by the corresponding assignment. The neurons that contain m spikes fire, sending one spike to neuron *out*, thus signalling that their corresponding assignment satisfies all the clauses of the instance. Neuron *out* operates like an OR gate: it fires if and only if it contains at least one spike, that is, if and only if at least one of the assignments satisfies all the clauses of γ_n.

In the next section, in particular, we aim to show the fact that the *assumed* pre-computed work space used in [6] to solve SAT *can be* pre-computed practically in advance in polynomial time by SN P systems with budding rules. Then, a solution to SAT problem is given by the systems with already pre-computed work space.

4 Uniform solution to SAT by (dendritic) SN P systems

Our SN P system with budding rules is composed of two subsequent subsystems: construction of a SN P system structure which meant to solve SAT problem uniformly and the SN P systems family, [6], which solves the SAT problem efficiently – for the sake of simplicity, we avoid the neuron budding and the spike firing rules are used at the same time in each subsystem.

$$\Pi = (O, \Sigma, H, syn, R, soma, out)$$

where:

1. $O = \{a\}$ is the singleton alphabet;
2. H is a finite set of labels for neurons,
   $H \supseteq H_0 = \{soma, out, e_0, e_1, e_2, e_3, b_1, b_2, b_3, c, s, +, -\}$ is the labels for neurons initially given;
3. $\Sigma = \{\sigma_i \mid i \in H_0\}$ is the set of initial neurons;
4. $syn \subseteq H \times H$ is a finite set of synapses, with $(i, i) \notin syn$ for $i \in H$),
   $syn \supseteq syn_0 = \{(e, e_i) \mid 0 \leq i \leq 3, e \in \{+, -\}\} \cup \{(e_0, b_i) \mid 1 \leq i \leq 3\} \cup \{(b_3, c), (s, +), (+, -), (-, +), \lambda\}$ is the set of synapses initially in use;
5. R is a set of *neuron budding* and *extended spiking* rules specified as follows.

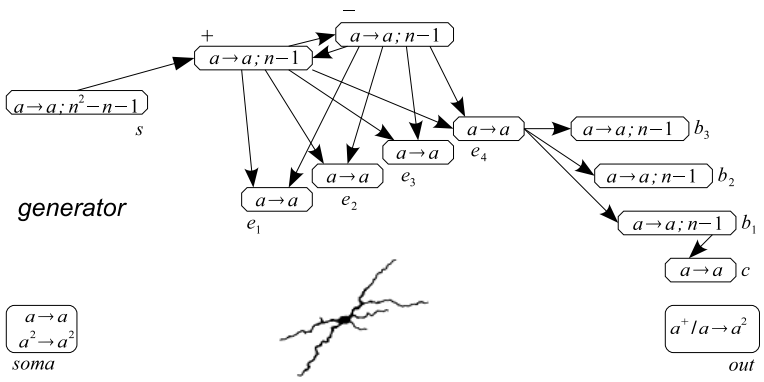


Fig. 4. The initial topological structure (new born dendrite) of the system Π: *soma* and *out* neurons, *generator*.

Constructing the system structure

The system initially contains an input neuron σ_{soma}, an output neuron σ_{out}, and a sub-structure so-called generator block G composed of the set of neurons Σ and the set of synapses syn_0, $|\Sigma| = |syn_0| = 13$, the corresponding topological structure is illustrated in Figure 4.

The generation mechanism is governed by only neuron budding rules and controlled by the labels of budding neurons and the created synapses. The labels of each neuron in a subsystem in layer n encodes an associated truth assignment.

The system construction algorithm consists of two main parts:

A. To generate the *dendritic-tree* sub-structure (the layers 0 – n in Figure 3, exponentially large in n) and the truth assignments for n Boolean variables. The process starts from the initial neuron σ_{soma} (the root node).

B. To complete the network structure. The subsystems in nth layer of the system establish connections to the *generator block* according to the truth assignments represented in those subsystems, and they are expanded by further three layers, finally converged to the output neuron σ_{out}.

A. The dendritic-tree generation process, controlled by the labels of neurons as well as the synapses, starts from the initial neuron σ_{soma} (cell body). It is noteworthy that since the truth assignments associated with the subsystems in nth layer are encoded in the labels of those neurons compose each subsystem, the truth assignments are being generated while the dendritic-tree has been constructed.

The label of a neuron σ_c is a sequence of the form

$$c = (k, j, x_k^{(p)}) = (k, j, x_k(1) = p) = (k, j, p, x_{k2}, \ldots, x_{kk}),$$

$p \in \{t, f\}$, where the first pair (k, j) indicates the location of the neuron on the dendritic-tree: k is the layer number, j is the place where the neuron is in its subsystem, the subsequence $x_k^{(p)}$ represents a string of length k formed by Boolean values t and f being generated. Whereas p in $x_k^{(p)}$ indicates that the first entry of the subsequence is exactly p – which is later importantly used in the budding rules to distinguish the being generated truth assignments from a same neuron. Moreover, we stress again that the labels of neurons of a chain of length k in a layer k represents a truth-assignment v of length k, precisely, v is a sequence formed by $x_k(j)$, $1 \leq j \leq k$, of the neuron labels $c = (k, j, x_k)$ of a chain. However, hence each chain or subsystem of a layer structure is a separate unit and associates with a truth assignment, all the truth assignments represented in a layer are distinguishable from each other. In other words, the truth assignments are encoded in both the labels of neurons of the chains and its layer structure of composing units too. We do not care which assignment is associated with which subsystem of the layer.

In this phase of computation three types of budding rules are performed for the role: budding rules of type $\mathbf{a_0}$) applied to the neuron σ_{soma} which initiates the generation of the structure; the *dendritic-tree* structure is constructed from the layer 0 towards the layer n, for each layer two types of rules such as $\mathbf{a_1}$) $n - 1$ times and $\mathbf{a_2}$) once, are alternated, total $n \times n$ steps needed to complete.

a₀) $[\ ]_{c_{soma}} \to (c_{soma}, c_{(1,1,t)})[\ ]_{c_{(1,1,t)}}$,
$[\ ]_{c_{soma}} \to (c_{soma}, c_{(1,1,f)})[\ ]_{c_{(1,1,f)}}$,
where $(c_{soma}, c_{(1,1,t)}), (c_{soma}, c_{(1,1,f)}) \in syn$.
The initial neuron $\sigma_{c_{soma}}$ buds two new neurons as the rules a_0) apply to it simultaneously. The newly produced neurons are: $\sigma_{c_{(1,1,t)}}$ with a synapse $(c_{soma}, c_{(1,1,t)})$ coming from the father neuron and $\sigma_{c_{(1,1,f)}}$ connected with the father neuron by a synapse $(c_{soma}, c_{(1,1,f)})$, respectively. Where the symbols t and f in the neuron labels indicate truth values $t(rue)$ and $f(alse)$, respectively, hence the two truth assignments (t) and (f) of length 1 for a single Boolean variable y_1 are formed. Note that the left hand side of each rule a_0) (where $\lambda \in syn_0$ is omitted) requires its interaction environment is empty i.e no synapse exists connected to the neuron $\sigma_{c_{soma}}$. Once the rules have applied, the interaction environment of the neuron $\sigma_{c_{soma}}$ has been evolved having two new synapses going out are created, which makes those rules are not applicable to this neuron anymore. Thus, the base of the first layer of the dendritic-tree has been established, at the first step of the computation.
An almost complete system structure for SAT$(2, m)$ is depicted in Figure 5, which is worth to follow during the construction.
To complete the established layer 1 (in general, i, $1 \le i \le n$), the rules of type a_1) generate the 2 (in general 2^i number of) subsystems or the chains of n neurons.

a₁) $(c_{(k,j-1,x_k^{(p)})}, c_{(k,j,x_k^{(p)})})[\ ]_{c_{(k,j,x_k^{(p)})}} \to (c_{(k,j,x_k^{(p)})}, c_{(k,j+1,x_k^{(p)})})[\ ]_{c_{(k,j+1,x_k^{(p)})}}$,
$p \in \{t, f\}$, $1 \le j \le n-1$, $1 \le k \le n$, $c_{(k,0,x_k^{(p)})} = c_{(k-1,n,x_{k-1})}$, $x_k^{(p)} = (p, x_{k-1}) \in \{t, f\}^k$.
The chains composed of n neurons in a layer k are generated by iterative applications of the rules of type a_1) in $n-1$ steps. This rule can be applied in a neuron of type $\sigma_{c_{(k,j,x_k^{(p)})}}$, $1 \le j \le n-1$, when its interaction environment is provided in which exists a single synapse $(c_{(k,j-1,x_k^{(p)})}, c_{(k,j,x_k^{(p)})})$ coming to the neuron. Then each rule buds a single neuron $\sigma_{c_{(k,j+1,x_k^{(p)})}}$ with a synaptic connection $(c_{(k,j,x_k^{(p)})}, c_{(k,j+1,x_k^{(p)})})$, where the second entry $(j+1)$ of the neuron label differs from the father neuron as its corresponding label entry as (j), otherwise the rest of the labeling sequence is inherited from the father neuron's label; x_k is a truth assignment of length k over $\{t, f\}$. The newly created synapse changes the interaction environment of the father neuron, which prevents another application of the rule.
As soon as the last neurons, whose second entry of the label is n, of the layer are produced, the next two types of rules are enabled to apply to those neurons as follows.

a₂) $(c_{(k,n-1,x_k^{(p)})}, c_{(k,n,x_k^{(p)})})[\ ]_{c_{(k,n,x_k^{(p)})}} \to (c_{(k,n,x_k^{(p)})}, c_{(k+1,1,t,x_k^{(p)})})[\ ]_{c_{(k+1,1,t,x_k^{(p)})}}$,
$(c_{(k,n-1,x_k^{(p)})}, c_{(k,n,x_k^{(p)})})[\ ]_{c_{(k,n,x_k^{(p)})}} \to (c_{(k,n,x_k^{(p)})}, c_{(k+1,1,f,x_k^{(p)})})[\ ]_{c_{(k+1,1,f,x_k^{(p)})}}$,
$p \in t, f$, and $1 \leq k \leq n-1$.

Those two rules of type a_2) apply simultaneously to each last neuron of type $\sigma_{c_{(k,n,x_k^{(p)})}}$ of each chain in the current layer k, the interaction environments must satisfy the rule condition. As a result, each neuron buds two new neurons with respective synapses. The next layer of the system is thus established. Hence the interaction environment of each father neuron extended by two new synapses, none of these rules is possible to apply again to those neurons. We shall look at the labels of newly produced pairs of type $\sigma_{c_{(k+1,1,t,x_k^{(p)})}}$ and $\sigma_{c_{(k+1,1,f,x_k^{(p)})}}$, the labels are formed as follows: first of all the pair $(k+1,1)$ corresponds to the neuron location where $k+1$ indicates the new layer number while 1 says the neuron is the very first one in its corresponding chain of length n in the new layer; the rest of the labeling sequence as $(t, x_k^{(p)})$ or $(f, x_k^{(p)})$ represents a new truth assignment for Boolean variables $x_1, x_2, \ldots, x_{k+1}$, where the newly inserted symbol t or f associates with a truth value $t(rue)$ or $f(alse)$, respectively, while $x_k^{(p)}$ is an heritage from the father neuron. Thus, all the possible 2^{k+1} different truth assignments are generated in layer $k+1$.

The truth assignment generation steps for two Boolean variables y_1, y_2 can be observed as described in Figure 5.

The rules of type a_1) are enabled in turn to complete the newly established layer by continued generation of the chains of length n.

By the alternated applications of the rules of types $\mathbf{a_1}$) $(n-1$ times) and $\mathbf{a_2}$) (once), in n^2 steps the layers from 0 to n are, the dendritic-tree, constructed by means an exponential work space and all the truth assignments of length n are generated.

Now, we come to the part B of the algorithm.

B. The pre-computation to construct the SN P system structure continues until it converges to the output neuron in a further few steps. The main function of this part of the algorithm is to design the substructure which is devoted to the test of the satisfiability of truth assignments against the clauses and to the exploration of the possibility whether any solution to the clauses of the propositional formula exists.

The very first task in part **B** is to connect the layer n to the generator block appropriately according to the truth assignments formed in this layer. We recall here that, in layer n, there are 2^n number of subsystems each one is composed of a sequence of n neurons (chains). However, each subsystem injectively corresponds to a different truth assignment of length n.

More precisely, taking the label of a neuron $\sigma_{c_{(n,j,x_n)}}$ in layer n, where the subsequence $x_n = (x_{n1}, x_{n2}, \ldots, x_{nn}) \in \{t,f\}^n$ represents a truth assignment. We associate jth entry of x_n with the jth neuron of considering subsystem, thus,

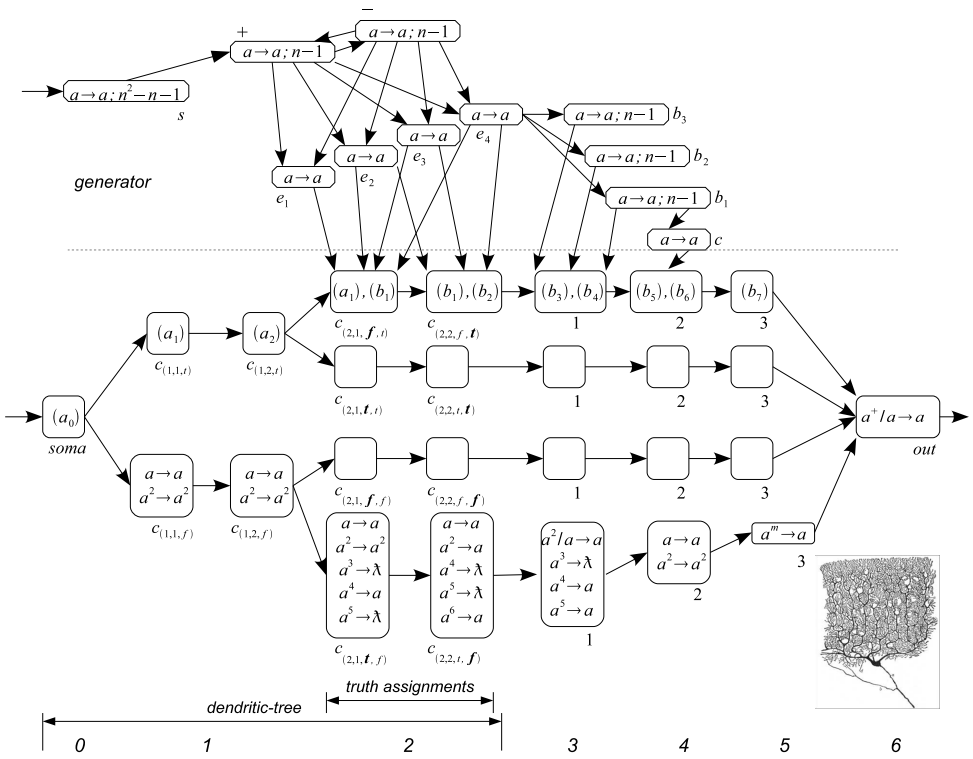


Fig. 5. An almost complete structure of Π system for SAT$(2, m)$ (maturated dendrite tree). The neuron budding rules used in each computation step are indicated by their labels in the corresponding neurons, while the spiking rules are presented too.

each neuron is indicated by an abstract triple $(n, j, x_n(j))$, where $1 \leq j \leq n$, and associated with a truth value x_{nj}. This way, a truth assignment of length n is represented by the n neurons (labels) of a subsystem.

For instance, in a case $n = 2$ as described in Figure 5, $2^2 = 4$ different truth assignments of length 2 have been generated for two Boolean variables y_1 and y_2 and each one is associated with a subsystem of layer $n = 2$. Technically, the first subsystem is composed of two neurons with labels $c_{(2,1,f,t)}$ and $c_{(2,2,f,t)}$, respectively. Whereas the former one associates with Boolean $t(rue)$ value as $x_2 = (f, t)$ and $x_2(1) = f$, while the later one with $f(alse)$ value as $x_2(2) = t$, and then altogether forms an assignment (f, t); the case with other subsystems are the same where (t, t), (f, f), (t, f), respectively, are generated; one can see that the four truth assignments are well distinguished from each other by the layer structure of four subsystems (chains).

The next synapse creation (budding) rules establish three synapses coming to the neurons which associate with a Boolean $t(rue)$ value while four synapses to the neurons associated with $f(alse)$ value, from the generator block.

b₁) $(c_{(n,j-1,x_n)}, c_{(n,j,x_n)})[\ ]_{c_{(n,j,x_n(j)=p)}} \rightarrow (c_{(n,j,x_n(j)=p)}, c_{e_i})[\ ]_{e_i}$,
$1 \leq j \leq n$, $p \in \{t, f\}$ and $s \leq i \leq 3$, where $s = 1$ if $p = t$, $s = 0$ if $p = f$, $c_{(n,0,x_n)} = c_{(n-1,n,x_n)}$.
Those neurons $\sigma_{c_{(n,j,x_n(j)=t)}}$ whose interaction environment satisfies the condition of the rules b_1) create three synapses coming from the neurons $\sigma_{c_{e_i}}$, $1 \leq i \leq 3$, while the neurons $\sigma_{c_{(n,j,x_n(j)=f)}}$ establish synapses coming to it from the four neurons $\sigma_{c_{e_i}}$, $0 \leq i \leq 3$, of the generator block. The synapse creation rules of type b_1) and the neuron budding rules of type a_1) are applied to the same neurons in layer n at the same time in a consequent $n-1$ steps as their interaction environments coincide.
Again looking at Figure 5, neuron $\sigma_{c_{(2,1,f,t)}}$ associates with $f(alse)$ value gets 4 synapses from the generator as neuron $\sigma_{c_{(2,1,f,t)}}$ gets 3 cause its identity of $t(rue)$ value.

b₂) $(c_{(n,n-1,x_n)}, c_{(n,n,x_n)})[\ ]_{c_{(n,n,x_n)}} \rightarrow (c_{(n,n,x_n)}, 1)[\ ]_1$.
The rule of type b_2) applies parallel to the last neurons of the layer n and produce the neurons σ_1 forming a new layer $n+1$. Meantime the rules of type b_1) create synapses from the same neurons of layer n to the generator block at last.

b₃) $(c_{(n,n,x_n)}, 1)[\ ]_1 \rightarrow (1, 2)[\ ]_2$,

b₄) $(c_{(n,n,x_n)}, 1)[\ ]_1 \rightarrow (b_i, 1)[\ ]_{b_i}$, $1 \leq i \leq 3$.
As rules of type b_3) apply to the neurons σ_1 and bud neurons σ_2, rules of type b_4) apply too and create three synapses coming from the neurons σ_{b_i}, $1 \leq i \leq 3$, to each neuron σ_1. Thus, layer $n+2$ is formed.

b₅) $(1, 2)[\ ]_2 \rightarrow (2, 3)[\ ]_3$,

b₆) $(1, 2)[\ ]_2 \rightarrow (c, 2)[\ ]_c$.
The rules of types b_5) and b_6) apply simultaneously to a neuron σ_2 with a synapse (1,2). As a result, the former one buds a new neuron σ_3, while the later one makes a new connection coming from the neuron σ_c as $(c, 2)$. All other neurons σ_2 get the same effect by the rules as the maximal parallel applications of the rules.

b₇) $(2, 3)[\ ]_3 \rightarrow (3, out)[\ ]_{out}$.
The pre-computation of the SN P system structure construction is completed by forming the converged connections from the neurons σ_3 to the output neuron σ_{out}, by means the rules of type b_7) are applied parallel to all neurons of layer $n+3$.

Thus, the SN P system device structure totally empty of spikes which is to solve all the instances of SAT(n, m), has been (pre-)computed in a polynomial time. The next computation stage (post-computation) to solve SAT(n, m) is plugged-in as follows.

Solving SAT

Any given instance γ_n of SAT(n, m) is encoded in a sequence of spikes. Each clause C_i of γ_n can be seen as a disjunction of at most n literals: for each $j \in \{1, 2, \ldots, m\}$, either y_j occurs in C_i, or $\neg y_j$ occurs, or none of them occurs. In order to distinguish these three situations we define the *spike variables* α_{ij}, for $1 \leq i \leq m$ and $1 \leq j \leq n$, as variables whose values are amounts of spikes, and we assign to them the following values:

$$\alpha_{ij} = \begin{cases} a & \text{if } y_j \text{ occurs in } C_i \\ a^2 & \text{if } \neg y_j \text{ occurs in } C_i \\ \lambda & \text{otherwise.} \end{cases}$$

In this way, clause C_i will be represented by the sequence $\alpha_{i1}\alpha_{i2}\cdots\alpha_{in}$ of spike variables; in order to represent the entire formula γ_n we just concatenate the representations of the single clauses, thus obtaining the sequence $\alpha_{11}\alpha_{12}\cdots\alpha_{1n}\alpha_{21}\alpha_{22}\cdots\alpha_{2n}\cdots\alpha_{m1}\alpha_{m2}\cdots\alpha_{mn}$. As an example, the representation of $\gamma_3 = (y_1 \vee \neg y_2) \wedge (y_1 \vee y_3)$ is $aa^2\lambda a\lambda a$.

The spiking rules residing in the neurons of the system which perform for solving the introduced problem are listed below with a brief description for each. But we do not go detailed explanation of each rule functions here, we prefer it refer to Section 3 and the paper [6], also the neuron budding rules are out of usage in this stage.

A given instance $\gamma_n \in$ SAT(n, m) encoded in a spike sequence is introduced into the system structure and will be processed by the spiking rules according to their roles in each step of the computation.

c₁) $[a \to a]_{c_{soma}}$; $[a^2 \to a^2]_{c_{soma}}$;
$[a \to a; n^2 - n - 1]_s$.
At each computation step of introducing the input, we insert 0, 1 or 2 spikes into the system through the input neuron σ_{soma}, according to the value of the spike variable α_{ij} we are considering in the representation of γ_n. Meantime we insert a single spike a into neuron σ_s once, which excites the generator block.

c₂) $[a \to a]_{c_{(k,j,x_k)}}$; $[a^2 \to a^2]_{c_{(k,j,x_k)}}$
$1 \leq k \leq n-1, 1 \leq j \leq n, x_k \in \{t, f\}^k$.
Each spike inserted into the input neuron is duplicated here and transmit along the first layer of the system towards next layers. When a spike passes a touching point – neuron with label of type $c_{(k,n,x_k)}$, it is duplicated and enter into next layer, etc., finally 2^n copies of them will take place at the layer n.

Once the copies of a clause are taken place on the neurons of the chains of length n in layer n, the combined functioning of the generator block and the layer n tests the assignments against each copy of the clause in consideration. For this purpose, the rules **c₃)** $-$ **c₅)** are used.

c₃) $[a \to a]_{e_i}; 0 \leq i \leq 3$,
$[a \to a; n-1]_+$; $[a \to a; n-1]_-$.

The generator block and its spiking rules. The generator block provides 3 and 4 spikes, respectively, in each n steps to the neurons associated with truth values t and f, of layer n, in order to test the satisfiability of the truth assignments against a clause which has been taken place through the layer.

c_4) $[a \to a]_{t_t}$; $[a^3 \to \lambda]_{t_t}$; $[a^2 \to a^2]_{t_1}$;
$[a^4 \to a]_{t_t}$; $[a^5 \to \lambda]_{t_t}$; $[a^2 \to a]_{t_0}$;
$t_t = c_{(n,j,x_n(j)=t)}, 1 \leq j \leq n$,
$t_1 = c_{(n,j,x_n(j)=t)}, 1 \leq j \leq n-1$,
$t_0 = c_{(n,n,x_n(n)=t)}, x_n \in \{t,f\}^n$.
The spiking rules residing in the neurons which associate Boolean $t(rue)$ value in layer n. The rules $a^2 \to a^2$, $a^2 \to a$, and $a \to a$ used to transmit the spike variables a, a^2 along the chains. Once a clause placed, each neuron associated with $t(rue)$ value contains either of a spike a or a^2 or *empty*. As a spike variable a represents a truth variable y, to which a spike *true* value a^3 sent by the generator is assigned and it results an *yes* answer as a^4, then it passes to the neuron σ_1 along the chain with a saying that a truth variable of the clause is satisfied by true value of a truth assignment or simply the clause is satisfied by a truth assignment of the corresponding chain. Meanwhile, a true value a^3 is assigned to the spike variables a^2 associates to truth variable $\neg y$ and *empty* wherever, which give a result *no* by means the rules $a^3 \to \lambda$ and $a^5 \to \lambda$ are performed.

c_5) $[a \to a]_{f_f}$; $[a^4 \to \lambda]_{f_f}$; $[a^2 \to a^2]_{f_1}$;
$[a^5 \to \lambda]_{f_f}$; $[a^6 \to a]_{f_f}$; $[a^2 \to a]_{f_0}$;
$f_f = c_{(n,j,x_n(j)=f)}, 1 \leq j \leq n$,
$f_1 = c_{(n,j,x_n(j)=f)}, 1 \leq j \leq n-1$,
$f_0 = c_{(n,n,x_n(n)=f)}, x_n \in \{t,f\}^n$.
The spiking rules residing in the neurons which associate with Boolean $f(alse)$ value in layer n. The functioning of the rules is similar as rules c_5).

c_6) $[a \to a; n-1]_{b_i}; 1 \leq i \leq 3$,
$[a^2/a \to a]_1$; $[a^3 \to \lambda]_1$;
$[a^4 \to a]_1$; $[a^5 \to a]_1$.
Whether an assignment satisfies the considered clause or not is checked by a combined functioning of the neurons with label 1 in layer $n+1$ and the neurons with label $b_i, 1 \leq i \leq 3$.

c_7) $[a \to \lambda]_2$; $[a^2 \to a]_2$;
$[a \to a]_c$.
With a combined function of neuron σ_c, neuron σ_2 emits a spike into neuron σ_3 if the corresponding assignment satisfies the under consideration clause here, otherwise no spike is emitted.

c_8) $[a^m \to a]_3$;
$[a^+/a \to a]_{out}$.
Neurons with label 3 count how many clauses of the instance γ_n are satisfied by the associated truth assignments. If any of those neurons get m spikes, which fire, hence the number of spikes that reach neuron *out* is the number of

assignments that satisfy all the clauses of γ_n. Thus, the output neuron fires if it has got at least one spike by means the problem has a positive solution, otherwise there is no assignment satisfies the instance γ_n.

This stage of the computation ends at the $n^2 + nm + 4$th step. The entire system halts in total at most in $2(n^2 + nm + 4)$ number of computation steps.

Thus, we got a full (deterministic, polynomial time and uniform) solution to SAT(n,m) in the framework of SN P systems.

5 Discussion

The present paper concerns the efficiency of SN P systems, we proposed a way to solve **NP**-complete problems, particularly SAT, in polynomial time. Specifically, the *neuron budding rule* is introduced in the framework of SN P systems, which a new feature enhances the efficiency of the systems to generate necessary work space. Neuron budding rules drive the mechanism of neuron production and synapse creation according to the interaction of a neuron with its environment (described by its synapses connected to other neurons). A very restricted type of rule of neuron budding, at most one synapse is involved in an environment, is used, but it is powerful enough to solve the considered problem, SAT. The solution to SAT by SN P systems with neuron budding contains two computation stages: first, constructing the device structure which has no spikes inside, second, introducing the considered problem to be solved encoded in spikes into the device. The system works in deterministic and maximally parallel manner. The whole mechanism we considered here for solutions to computational intractable problems is elegant from computational complexity theory point of view as the designed algorithm can be computed by a deterministic Turing machine in polynomial time; the operation of neuron budding is well motivated by neural biology.

We believe that SN P systems use the restricted budding rules can be an efficient computing tool to solve other **NP** hard problems.

The SN P systems with neuron budding rules can be extended by introducing more general rules, which in some sense capture the dynamic interaction of neurons with their environment. One possible form of such general rules is as follows: $A_i[\ ]_i B_i \rightarrow C_j[\ ]_j D_j$, where A_i, B_i and C_j, D_j are the set of synapses coming to and going out from, respectively, the specified neurons σ_i and σ_j. Clearly, in such general rules, more than one synapses are involved in the environment of the considered neuron.

Acknowledgments

The work of Tseren-Onolt Ishdorj was supported by the project "Biotarget", it is a joint project between U. of Turku and Åbo Akademi University funded by the Academy of Finland. The work of L. Pan was supported by National Natural Science Foundation of China (Grant Nos. 60674106, 30870826,

60703047, and 60533010), Program for New Century Excellent Talents in University (NCET-05-0612), Ph.D. Programs Foundation of Ministry of Education of China (20060487014), Chenguang Program of Wuhan (200750731262), HUST-SRF (2007Z015A), and Natural Science Foundation of Hubei Province (2008CDB113 and 2008CDB180).

References

1. H. Chen, M. Ionescu, T.-O. Ishdorj: On the efficiency of spiking neural P systems. *Proc. 8th Inter. Conf. on Electronics, Information, and Communication*, Ulanbator, Mongolia, June 2006, 49–52.
2. H. Chen, M. Ionescu, T.-O. Ishdorj, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Spiking neural P systems with extended rules. *Fourth Brainstorming Week on Membrane Computing* (M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds.), vol. I, RGNC Report 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 2006, 241–266.
3. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. *Fundamenta Informaticae* 75 (2007), 141–162.
4. V. Danos, J. Feret, W. Fontana, R.Harmer and J. Krivine, *Investigation of a Biological Repair Scheme*, In Proceedings of 9th International Workshop on Membrane Computing, Edinburgh, UK, July 28-31, 2008, LNCS 5391, 1–12, 2009.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2–3 (2006), 279–308.
6. T.-O. Ishdorj, A. Leporati: Uniform solutions to `SAT` and `3-SAT` by spiking neural P systems with pre-computed resources. *Natural Computing*, 7, 4 (2008), 519–534.
7. T.-O. Ishdorj, A. Leporati, L. Pan, X. Zhang, X. Zeng: Uniform solutions to `QSAT` and `Q3SAT` by spiking neural P systems with pre-computed resources. In the present volume.
8. M.A. Gutiérrez-Naranjo, A. Leporati: Solving numerical NP-complete problems by spiking neural P systems with pre-computed resources. *Proceedings of Sixth Brainstorming Weeks on Membrane Computing*, Sevilla University, Sevilla, February 2-8, 2008, 193–210.
9. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical NP-complete problem with spiking neural P systems. *Membrane Computing, International Workshop, WMC8, Thessaloniki, Greece, 2007, Selected and Invited Papers* (G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa, eds.), LNCS 4860, Springer-Verlag, Berlin, 2007, 336–352.
10. A. Leporati, G. Mauri, C. Zandron, Gh. Păun, M. J. Pérez-Jiménez: Uniform solutions to SAT and SUBset SUM by spiking neural P systems. Submitted.
11. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: On the computational power of spiking neural P systems. *International Journal of Unconventional Computing*, 2007, in press.
12. Gh. Păun: *Membrane Computing – An Introduction*. Springer-Verlag, Berlin, 2002.
13. Gh. Păun: Twenty six research topics about spiking neural P systems. *Fifth brainstorming week on membrane computing*, Fenix Editora, Sevilla, 2007, 263-280.
14. M. Sipser: *Introduction to the Theory of Computation*. PWS Publishing Company, Boston, 1997.
15. Think and Grow Toys: `http://www.tagtoys.com/dendrites.htm`

Author Index