INTERNATIONAL JOURNAL

of

COMPUTERS, COMMUNICATIONS & CONTROL

With Emphasis on the Integration of Three Technologies

IJCCC
A Quarterly Journal

Year: 2009 Volume: IV Number: 3 (September)

Agora University Editing House

CCC Publications

Licensed partner: EBSCO Publishing

www.journal.univagora.ro

Publisher & Editorial Office
CCC Publications, Agora University
Piata Tineretului 8, Oradea, jud. Bihor, Romania, Zip Code 410526
Tel: +40 259 427 398, Fax: +40 259 434 925, E-mail: ccc.journal@gmail.com
Website: www.journal.univagora.ro
ISSN 1841-9836, E-ISSN 1841-9844

International Journal of Computers, Communications and Control (IJCCC) is published from 2006 and has 4 issues/year (March, June, September, December), print & online.

Founders of IJCCC: I. Dziţac, F.G. Filip and M.J. Manolescu (2006)

This publication is subsidized by:

1. Agora University
2. The Romanian Ministry of Education and Research / The National Authority for Scientific Research

CCC Publications, powered by Agora University Publishing House, currently publishes the "International Journal of Computers, Communications & Control" and its scope is to publish scientific literature (journals, books, monographs and conference proceedings) in the field of Computers, Communications and Control.

IJCCC is indexed and abstracted in a number of databases and services including:

1. ISI Thomson Reuters - Science Citation Index Expanded (also known as SciSearch®)
2. ISI Thomson Reuters - Journal Citation Reports/Science Edition
3. ISI Thomson Reuters - Master Journal List
4. SCOPUS
5. Computer & Applied Science Complete (EBSCO)
6. Vocational Studies Complete (EBSCO)
7. Current Abstracts (EBSCO)
8. Collection of Computer Science Bibliographies(CCSB)
9. Informatics portal io-port.net (FIZ KARLSRUHE)
10. MathSciNet
11. Open J-Gate
12. Google Scholar
13. Romanian Journals CNCSIS (cat.A, cod 849)
14. Information Systems Journals (ISJ)
15. Urlich's Periodicals Directory
16. Genamics JournalSeek
17. ISJ- Journal Popularity
18. Magistri et Scholares
19. SCIRUS
20. DOAJ

Contents

SPECIAL ISSUE ON MEMBRANE COMPUTING

Seventh Brainstorming Week on Membrane Computing

The present volume contains a selection of papers resulting from the Seventh Brainstorming Week on Membrane Computing (BWMC7), held in Sevilla, from February 2 to February 6, 2009. The meeting was organized by the Research Group on Natural Computing (RGNC) from Department of Computer Science and Artificial Intelligence of Sevilla University. The previous editions of this series of meetings were organized in Tarragona (2003), and Sevilla (2004 – 2008). 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 has 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.

Membrane computing is an area of natural computing which studies models of computation inspired by the structure and functioning of living cells, and organization of cells in tissues and other structures. The resulting models (called P systems) are distributed parallel computing devices, processing multisets in compartments defined by membranes. Most classes of P systems are computationally universal and, if an exponential working space can be produced in polynomial time (e.g., by membrane division), then they are able to solve computationally hard problems in a feasible time. A series of applications were recently reported, especially in biology and medicine, but also in computer graphics, cryptography, linguistics, economics, approximate optimization, etc. Several simulation programs (useful in applications) are available by now. A comprehensive information about this research area (considered in 2003 by ISI as “fast emerging research front in computer science") can be found at the website `http://ppage.psystems.eu`.

At this web address one can also find the volumes published after each BWMC, with the papers resulting from these meetings, including the volume with all papers related to BWMC7.

For the present volume we have selected only a few of these papers; they have been thoroughly reworked after the meeting and then they went through the standard refereeing procedure of the journal. The selection also intended to provide a good image of the research in membrane computing, so that the volume contains both theoretical and applicative papers, dealing with computing power, computational complexity, “classic" cell-like P systems and the recently introduced spiking neural P systems, programming, simulation of biological processes, and so on.

*

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.

*

The present volume is dedicated to the 60th birthday anniversary of Professor Mario de Jesús Pérez-Jiménez, the head of the Research Group on Natural Computing from Sevilla University, and one of the most active researchers in membrane computing. A really unique combination of enthusiasm and mathematical talent, of scientific devotion and altruism, Mario not only contributed a lot to the research in membrane computing, with fundamental results especially related to computational complexity issues and also related to many other theoretical research questions, to applications in biology and eco-systems, to programming, etc., but he also created one of the strongest research groups in natural computing in general, in membrane computing in special; it also deserves to be mentioned the organization, for already several years in a row, of the Brainstorming Week on Membrane Computing – all these making Sevilla a place of current "pilgrimage" of researchers in membrane computing, from Europe, Asia, America.

For all those who know Mario personally, it is hard to believe that he has already six decades: he is so active, enthusiastic and hard working that he looks as young as decades ago, and the theoretical possibility to get retired (according to Spanish regulations, this is possible for Mario) looks like a non-sensical joke... And, for all who know Mario personally, it is impossible not to own to him a lot, from science to daily life (one of the sayings which circulate around is that if you have a need, it is wiser not to tell it loudly, because Mario will try immediately to help you...).

Happy Birthday, Mario, and many happy returns!

Guest Editors:
Giancarlo Mauri, Milan, Italy
Gheorghe Păun, Bucharest, Romania
Agustín Riscos-Núñez, Seville, Spain
(Sevilla, June 2009)

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 206-213

Dictionary Search and Update by P Systems with String-Objects and Active Membranes

Artiom Alhazov, Svetlana Cojocaru, Ludmila Malahova, Yurii Rogozhin

Artiom Alhazov, Svetlana Cojocaru, Ludmila Malahova, Yurii Rogozhin
Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
E-mail: {artiom,sveta, mal, rogozhin}@math.md

Artiom Alhazov
IEC, Department of Information Engineering, Graduate School of Engineering
Hiroshima University, Higashi-Hiroshima 739-8527 Japan

Yurii Rogozhin
Rovira i Virgili University, Research Group on Mathematical Linguistics
Av. Catalunya, 35, Tarragona 43002 Spain
E-mail: yrogozhin@yahoo.com

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: Membrane computing is a formal framework of distributed parallel computing. In this paper we implement the work with the prefix tree by P systems with strings and active membranes. We present the algorithms of searching in a dictionary and updating it implemented as membrane systems. The systems are constructed as reusable modules, so they are suitable for using as sub-algorithms for solving more complicated problems.

Keywords: Membrane computing, P systems, active membranes, dictionary, prefix tree

1 Introduction

Solving most problems of natural language processing is based on using certain linguistic resources, represented by corpora, lexicons, etc. Usually, these collections of data constitute an enormous volume of information, so processing them requires much computational resources. A reasonable approach for obtaining efficient solutions is that based on applying parallelism; this idea has been promoted already in 1970s. For instance, the possibilities of applying massive parallelism in Machine Translation are considered in [5, 2]. Many of the stages of text processing (from tokenization, segmentation, lematizing to those dealing with natural language understanding) can be carried out by parallel methods. This justifies the interest to the methods offered by the biologically inspired models, and by membrane computing in particular.

However, there are some issues that by their nature do not allow complete parallelization, yet exactly they are often those “computational primitives" that are inevitably used during solving major problems, like the elementary arithmetic operations are always present in solving difficult computational problems. Among such “primitives" in the computational linguistics we mention handling of the dictionaries, e.g., dictionary lookup and dictionary update. Exactly these problems constitute the subject of the present paper. In our approach we speak about dictionary represented by a prefix tree.

P (membrane) systems are a convenient framework of describing computations on trees. Since membrane systems are an abstraction of living cells, the membranes are arranged hierarchically, yielding a tree structure.

2 Definitions

Membrane computing is a recent domain of natural computing initiated by Gh. Păun in [12]. The components of a membrane system are a cell-like membrane structure, in the regions of which one places multisets of objects which evolve in a synchronous maximally parallel manner according to given evolution rules associated with the membranes. The necessary definitions are given in the following subsection; see also [4] for an overview of the domain and [6] for a comprehensive bibliography.

2.1 Computing by P systems

Let O be a finite set of elements called symbols; then set of words over O is denoted by O^*, and the empty word is denoted by λ.

Definition 1. A P system with string-objects and input is a tuple

$\Pi = \big(O, \Sigma, H, E, \mu, M_1, \cdots, M_p, R, i_0\big)$, where:

- O is the working alphabet of the system (the objects are strings over O),
- Σ is an input alphabet,
- H is an alphabet whose elements are called labels, i_0 identifies the input region,
- E is the set of polarizations,
- μ is a membrane structure (a rooted tree) consisting of p membranes injectively labeled by elements of H,
- M_i is an *initial* multiset of strings over O associated with membrane i, $1 \leq i \leq p$,
- R is a finite set of rules defining the behavior of objects from O^* and of membranes labeled by elements of H.

A configuration of a P system is its "snapshot", i.e., the current membrane structure and the multisets of string-objects present in regions of the system. The initial configuration is $C_0 = (\mu, M_1, \cdots, M_p)$. Each subsequent configuration C' is obtained from the previous configuration C by maximally parallel application of rules to objects and membranes. This is denoted by $C \Rightarrow C'$ (no further rules are applicable together with the rules that transform C into C'). A computation is thus a sequence of configurations starting from C_0, respecting relation $\Rightarrow$ and ending in a halting configuration (i.e., such one that no rules are applicable). If M is a multiset of strings over the input alphabet $\Sigma \subseteq O$, then the *initial configuration* of a P system Π with an input M over alphabet Σ and input region i_0 is $(\mu, M_1, \ldots, M_{i_0-1}, M_{i_0} \cup M, M_{i_0+1}, \ldots, M_p)$.

2.2 P systems with active membranes

To speak about P systems with active membranes, we need to specify the rules, i.e., the elements of the set R in the description of a P system. Due to the nature of the problem of this paper, the standard model was generalized in the following:

- Cooperative rules: a rule operates on a substring of an object (otherwise, the system cannot even distinguish different permutations of a string); this feature is represented by a superscript * in the rule types;
- String replication (to return the result without removing it from the dictionary);

- Membrane creation (to add words to the dictionary).

Hence, the rules can be of the following forms:

(a^*) $[\, a \to b \,]_h^e$ for $h \in H, e \in E, a, b \in O^*$- evolution rules
(associated with membranes and depending on the label and the polarization of the membranes, but not directly involving the membranes: the membranes are neither taking part in the application of these rules nor are they modified by them);

(a_r^*) $[\, a \to b || c \,]_h^e$ for $h \in H, e \in E, a, b, c \in O^*$ (as above, but with string replication);

(b^*) $a[\ \ ]_h^{e_1} \to [\, b \,]_h^{e_2}$ for $h \in H, e_1, e_2 \in E, a, b \in O^*$ - communication rules
(an object is introduced into the membrane, possibly modified; the polarization of the membrane can be modified, but not its label);

(c^*) $[\, a \ \,]_h^{e_1} \to [\ \ ]_h^{e_2} b$ for $h \in H, e_1, e_2 \in E, a, b \in O^*$ - communication rules
(an object is sent out of the membrane, possibly modified; also the polarization of the membrane can be modified, but not its label);

(d^*) $[\, a \ \,]_h^e \to b$ for $h \in H, e \in E, a, b \in O^*$ - dissolving rules
(in reaction with an object, a membrane can be dissolved, while the object specified in the rule can be modified);

(g^*) $[\, a \ \to [\, b \ \,]_g^{e_2} \,]_h^{e_1}$ for $g, h \in H,\ e_1, e_2 \in E,\ a, b \in O^*$ - membrane creation rules
(an object is moved into a newly created membrane, possibly modified).

Additionally, we will write $\emptyset$ in place of some strings on the right-hand side of the rules, meaning that the entire string is deleted.

The rules of types $(a^*), (a_r^*)$ and (g^*) are considered to only involve objects, while all other rules are assumed to involve objects and membranes mentioned in their left-hand side. An application of a rule consists in replacing a substring described in the left-hand side of a string in the corresponding region (i.e., associated to a membrane with label h and polarization e for rules of types $(a^*), (a_r^*)$ and (d^*), or associated to a membrane with label h and polarization e_1 for rules of type (c^*), or immediately outer of such a membrane for rules of type (b^*)), by a string described in the right-hand side of the rule, moving the string to the corresponding region (that can be the same as the source region immediately inner or immediately outer, depending on the rule type), and updating the membrane structure accordingly if needed (changing membrane polarization, creating or dissolving a membrane). Only the rules involving different objects and membranes can only be applied in parallel; such parallelism is maximal if no further rules are applicable in parallel.

3 Dictionary

Dictionary search represents computing a string-valued function $\{u_i \longrightarrow v_i \mid 1 \le i \le d\}$ defined on a finite set of strings.

We represent such a dictionary by the skin membrane containing the membrane structure corresponding to the prefix tree of $\{u_i \mid 1 \le i \le d\}$, with strings $\$v_i\$'$ in regions corresponding to the nodes associated to u_i. Let A_1, A_2 be the source and target alphabets: $u_i \in A_1^*$, $v_i \in A_2^*$, $1 \le i \le d$. Due to technical reasons, we assume that for every $l \in A_1$, the skin contains a membrane with label l. We also suppose that the source words are non-empty.

For instance, the dictionary $\{\text{bat} \longrightarrow \text{flying}, \text{bit} \longrightarrow \text{stored}\}$ is represented by

$$[\,[\ \,]_a^o[\,[\,[\,\$flying\$'\,]_t^o\,]_a^o[\,[\,\$stored\$'\,]_t^o\,]_i^o\,]_b[\ \,]_c^o \cdots [\ \,]_z^o\,]_o^o$$

Consider a P system corresponding to the given dictionary:

$$\begin{array}{lcl} \Pi & = & \big(O,\Sigma,H,E,\mu,M_1,\cdots,M_p,R,i_0\big), \\ O & = & A_1\cup A_2\cup\{?,?',\$,\$',\$_1,\$_2,\mathtt{fail}\}\cup\{?_i \mid 1\le i\le 11\}\cup\{!_i \mid 1\le i\le 4\}, \\ \Sigma & = & A_1\cup A_2\cup\{?,?',!,\$,\$'\}, \\ H & = & A_1\cup\{0\},\ E=\{0,+,-\},\ i_0=1, \\ \mu & \text{and} & \text{sets } M_i,\ 1\le i\le p, \text{ are defined as described above.} \end{array}$$

So only the rules and input semantics still have to be defined.

3.1 Dictionary search

To translate a word u, input the string $?u?'$ in region 1. Consider the following rules.

S1 $?_l[\ ]_l^0 \to [\ ?\ ]_l^0,\ l\in A_1$

Propagation of the input into the membrane structure, reaching the location corresponding to the input word.

S2 $[\ ??'\ ]_l^0 \to [\ ]_l^-\emptyset,\ l\in A_1$

Marking the region corresponding to the source word.

S3 $[\ \$\to \$_1\|\$_2\ ]_l^-,\ l\in A_1$

Replicating the translation.

S4 $[\ \$_2\ ]_l^e \to [\ ]_l^0\$_2,\ l\in H,\ e\in\{-,0\}$

Sending one copy of the translation to the environment.

S5 $[\ \$_1\to\$\ ]_l^0,\ l\in A_1$

Keeping the other copy in the dictionary.

The system will send the translation of u in the environment. This is a simple example illustrating search. If the source word is not in the dictionary, the system will be blocked without giving an answer. The following subsection shows a solution to this problem.

3.2 Search with fail

The set of rules below is considerably more involved than the previous one. However, it handles 3 cases: a) the target word is found, b) the target word is missing in the target location, c) the target location is unreachable.

F1 $[\ ?\to ?_1\|?_2\ ]_0^0$

Replicate the input.

F2 $[\ ?_2\to ?_3\ ]_0^0$

Delay the second copy of the input for one step.

F3 $?_1l[\ ]_l^0 \to [\ ?_1\ ]_l^+,\ l\in A_1$

Propagation of the first copy towards the target location, changing the polarization of the entered membrane to $+$.

F4 $?_3 l[\ ]_l^+ \to [\ ?_3\ ]_l^{\mathrm{o}}, l \in A_1$

Propagation of the second copy towards the target location, restoring the polarization of the entered membrane.

F5 $[\ ?_1 l \to [\ ?_4\ ]_l^-\ ]_k^{\mathrm{o}}, l,k \in A_1$

If a membrane corresponding to some symbol of the source word is missing, then the first copy of the input remains in the same membrane, while the second copy of the input restores its polarization. Creating a membrane to handle the failure.

F6 $[\ ?_1 ?' \to ?_7\ ]_l^{\mathrm{o}}, l \in A_1$

Target location found, marking the first input copy.

F7 $[\ ?_7\ ]_l^{\mathrm{o}} \to [\ ]_l^- \emptyset, l \in A_1$

Marking the target location.

In either case, some membrane has polarization $-$. It remains to send the answer out, or fail if it is absent. The membrane should be deleted in the fail case.

F8 $[\ \$ \to \$_1 || \$_2\ ]_l^-, l \in A_1$

Replicating the translation.

F9 $[\ \$_2\ ]_l^e \to [\ ]_l^{\mathrm{o}} \$_2, l \in H, e \in \{\mathrm{o}, -\}$

Sending one copy of the translation out.

F10 $[\ \$_1 \to \$\ ]_l^{\mathrm{o}}, l \in A_1$

Keeping the other copy in the dictionary.

F11 $[\ ?_3 \to ?_5\ ]_l^-, l \in A_1$

The second copy of input will check if the translation is available in the current region.

F12 $?_3 l[\ ]_l^- \to [\ ?_5\ ]_l^-, l \in A_1$

The second copy of input enters the auxiliary membrane with polarization $-$.

By now the second copy of the input is in the region corresponding to either the search word, or to its maximal prefix plus one letter (auxiliary one).

F13 $[\ ?_5 \to ?_6\ ]_l^-, l \in A_1$

It waits for one step.

F14 $[\ ?_6 \to \emptyset\ ]_l^{\mathrm{o}}, l \in A_1$

If the target word has been found, the second copy of the input is erased.

F15 $[\ ?_6\ ]_l^- \to [\ ]_l^{\mathrm{o}} ?_8, l \in A_1$

If not, the search fails.

F16 $[\ ?_8\ ]_l^{\mathrm{o}} \to [\ ]_l^{\mathrm{o}} ?_8, l \in A_1$

Sending the fail notification to the skin.

F17 $[\ ?_8 l \to ?_8\ ]^{\mathrm{o}}_{\mathrm{o}}$

Erasing the remaining part of the source word.

F18 $[\ ?_8 ?'\ ]^{\mathrm{o}}_{\mathrm{o}} \to [\ ]^{\mathrm{o}}_{\mathrm{o}}\texttt{fail}$

Answering fail.

F19 $[\ ?_4 \to ?_9\ ]^{-}_{l}$, $l \in A_1$

F20 $[\ ?_9 \to ?_{10}\ ]^{-}_{l}$, $l \in A_1$

F21 $[\ ?_{10} \to ?_{11}\ ]^{-}_{l}$, $l \in A_1$

If the target location was not found, the first input copy waits for 3 steps while the membrane with polarization $-$ handles the second input copy.

F22 $[\ ?_{11}\ ]^{\mathrm{o}}_{l} \to \emptyset$, $l \in A_1$

Erasing the auxiliary membrane.

3.3 Dictionary update

To add an entry $u \longrightarrow v$ to the dictionary, input the string $!u\$v\$'$ in region 1. Consider the following rules.

U1 $[\ ! \to !_1 || !_2\ ]^{\mathrm{o}}_{\mathrm{o}}$

Replicate the input.

U2 $[\ !_2 \to !_3\ ]^{\mathrm{o}}_{\mathrm{o}}$

Delay the second copy of the input for one step.

U3 $!_1 l[\ ]^{\mathrm{o}}_{l} \to [\ !_1\ ]^{+}_{l}$, $l \in A_1$

Propagation of the first copy towards the target location, changing the polarization of the entered membrane to $+$.

U4 $!_3 l[\ ]^{+}_{l} \to [\ !_3\ ]^{\mathrm{o}}_{l}$, $l \in A_1$

Propagation of the second copy towards the target location, restoring the polarization of the entered membrane.

U5 $[\ !_1 \to !_4\ ]^{\mathrm{o}}_{l}$, $l \in A_1$

If a membrane corresponding to some symbol of the source word is missing, then the first copy of the input remains in the same membrane, while the second copy of the input restores its polarization. Marking the fist copy of the input for creation of missing membranes.

U6 $[\ !_4 l \to [\ !_4\ ]^{+}_{l}\ ]^{\mathrm{o}}_{k}$, $l,k \in A_1$

Creating missing membranes.

U7 $[\ !_4\$ \to \$\ ]^{\mathrm{o}}_{l}$, $l \in A_1$

Releasing the target word in the corresponding location.

U8 $[\ !_3\$ \to \emptyset\ ]^{\mathrm{o}}_{l}, l \in A_1$

Erasing the second copy of the input.

We underline that the constructions presented above also hold in a more general case, i.e., when the dictionary is a multi-valued function. Indeed, multiple translations can be added to the dictionary as multiple strings in the region associated to the input word. The search for a word with multiple translations will lead to all translations sent to the environment. The price to pay is that the construction is no longer deterministic, since the order of application of rules S4 or F9 to different translations is arbitrary. Nevertheless, the constructions remain "deterministic modulo the order in which the translations are sent out". All constructions work in linear time with respect to the length of the input. The parallelism is vital for checking for the absence of a needed submembrane, or for checking for the absence of a translation of a given word; sending multiple translation results out is also parallel.

4 Discussion

In this paper we presented the linear-time algorithms of searching in a dictionary and updating it implemented as membrane systems. We underline that the systems are constructed as reusable modules, so they are suitable for using as sub-algorithms for solving more complicated problems.

The scope of handling dictionaries is not limited to the dictionaries in the classical sense. Understanding a dictionary as introduced in Section 3, i.e., a string-valued function defined on a finite set of strings, leads to direct applicability of the proposed methods to handle alphabets, lexicons, thesauruses, dictionaries of exceptions, and even databases. At last, it is natural to consider these algorithms together with morphological analyzer and morphological synthesizer.

Acknowledgments All authors gratefully acknowledge the support by the Science and Technology Center in Ukraine, project 4032. Artiom Alhazov gratefully acknowledges the support of the Japan Society for the Promotion of Science and the Grant-in-Aid for Scientific Research, project 20·08364. Yurii Rogozhin gratefully acknowledges the support of the European Commission, project MolCIP, MIF1-CT-2006-021666.

Bibliography

[1] G. Ciobanu, G. Păun, M.J. Pérez-Jiménez Eds., Applications of Membrane Computing, Springer-Verlag, 2006.

[2] H. Kitano, Challenges of Massive Parallelism, Proceedings of the *13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993, vol. 1, 813–834.

[3] Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences* **61**(1), 2000, 108–143.

[4] Gh. Păun, *Membrane Computing. An Introduction*, Springer-Verlag, 2002.

[5] E. Sumita, K. Oi, O. Furuse, H. Iida, T. Higuchi, N. Takahashi, H. Kitano, Example-Based Machine Translation on Massively Parallel Processors, Proceedings of the *13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993, vol. 2, 1283–1289.

[6] P systems webpage. `http://ppage.psystems.eu/`.

Artiom Alhazov (born on October 11, 1979), graduated in Mathematics and Computer Science (The State University of Moldova, Chişinău, Moldova) and received Ph.D. in Languages and Information Systems (Rovira i Virgili University, Tarragona, Spain). A researcher at the Institute of Mathematics and Computer Science of the Academy of Sciences of Moldova. He completed a postdoc in Åbo Akademi University, Turku, Finland, and currently has a postdoc in Hiroshima University, Higashi-Hiroshima, Japan. His main research interests are theoretical computer science, formal language theory, parallel distributed computational models, and in particular the descriptional complexity of P systems with weak forms of interaction. He published over 90 research papers (collaborating with more than 30 researchers from many countries in Europe and Asia). He won numerous prizes for programming in school and university years, and the National Youth Prize in Science, Technics, Literature and Arts in 2006 for a collection of research works.

Svetlana Cojocaru (born on July 26, 1952), graduated in Mathematics (The State University of Moldova, Chişinău, Moldova, 1974), received Ph.D. in Computer Science (Institute of Cybernetics, Ukrainian Academy of Sciences, Kiev, 1982) and doctor in habilitation in Computer Science (Institute of Mathematics and Computer Science, Academy of Sciences of Moldova, 2007). A deputy director at the Institute of Mathematics and Computer Science of the Academy of Sciences of Moldova. Her main research interests are formal languages and grammars, natural language processing, computer algebra, molecular computing. She published over 120 research papers.

Ludmila Malahova (born on July 22, 1947), graduated in Computer Science (The State University of Moldova, Chişinău, Moldova, 1970). A researcher at the Institute of Mathematics and Computer Science of the Academy of Sciences of Moldova. She has a significant experience in computer science including computer graphics, formal languages, computer algebra, natural language processing, and molecular computing with more than 80 papers published in international journals, books and conference proceedings.

Yurii Rogozhin (born on November 13, 1949), graduated in Mathematics (The Kuban State University, Krasnodar, Russia, 1971), received Ph.D. in Mathematical Cybernetics (Computer Center of the Russian Academy of Sciences, Moscow, 1981) and doctor in habilitation in Computer Science (Moscow State University, Russia, Department of Computational Mathematics and Cybernetics, 1999). A principal researcher at the Institute of Mathematica and Computer Science of the Academy of Sciences of Moldova Republic and Marie Curie IIF researcher at Rovira i Virgili University, Research Group on Mathematical Linguistics, Tarragona, Spain. His main research interests are mathematics, theoretical computer science, formal language theory and its applications, natural (biomolecular) computing and nanotechnology. He published over 110 research papers (collaborating with more than 45 researches from many countries in Europe).

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 214-223

P Systems with Endosomes

Roberto Barbuti, Giulio Caravagna, Andrea Maggiolo-Schettini, Paolo Milazzo

Università di Pisa
Dipartimento di Informatica
Largo Pontecorvo 3, 56127 Pisa, Italy
E-mail: {barbuti, caravagn, maggiolo, milazzo}@di.unipi.it

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: P Systems are computing devices inspired by the structure and the functioning of a living cell. A P System consists of a hierarchy of membranes, each of them containing a multiset of objects, a set of evolution rules, and possibly other membranes. Evolution rules are applied to the objects of the same membrane with maximal parallelism. In this paper we present an extension of P Systems, called P Systems with Endosomes (PE Systems), in which endosomes can be explicitly modeled. We show that PE Systems are universal even if only the simplest form of evolution rules is considered, and we give one application example.
Keywords: P systems, PE Systems, Endosomes

1 Introduction

P Systems were introduced by Păun in [10] as distributed parallel computing devices inspired by the structure and the functioning of a living cell. A P System consists of a *hierarchy of membranes*, each of them containing a multiset of *objects*, representing molecules, a set of *evolution rules*, representing chemical reactions, and possibly other membranes. For each evolution rule there are two multisets of objects, describing the reactants and the products of the chemical reaction. A rule in a membrane can be applied only to objects in the same membrane. Some objects produced by the rule remain in the same membrane, where each membrane is identified by its labels, others are sent *out* of the membrane, others are sent *into* the inner membranes. Evolution rules are applied with *maximal parallelism*, meaning that it cannot happen that some evolution rule is not applied when the objects needed for its triggering are available.

Many variants and extensions of P Systems exist that include features to increase their expressiveness and that are based on different evolution strategies. Among the most common extensions we mention P Systems with dissolution rules that allow a membrane to disappear and release in the surrounding membrane all the objects it contains. We mention also P Systems with priorities, in which a priority relationship exists among the evolution rules of each membrane and can influence the applicability of such rules, and P Systems with promoters and inhibitors, in which the applicability of evolution rules depends on the presence of at least one occurrence and on the absence, respectively, of a specific object. See [11] for the definition of these (and other) variants of P Systems and [14] for a complete list of references to the bibliography of P Systems.

In this paper we present another extension of P Systems, called P Systems with Endosomes (PE Systems), with the following features:

- objects can be contained inside the regions delimited by the membranes and on the surfaces of the membranes (as in P Systems with peripheral proteins [6, 13] and as in membrane systems with surface objects [1, 2]);

- rules are contained on the surfaces of the membranes (they can rewrite objects outside/on/into the membranes);
- endosomes can be explicitly created in order to model a biologically inspired transportation mechanism.

The definition of this extension of P Systems has a biological inspiration. In fact, the *endocytosis* of macromolecules is the process by which cells absorb material (molecules such as proteins) from outside the cell by engulfing it with their cell membrane. It is used by all cells because most substances important to them are large polar molecules that cannot pass through the hydrophobic plasma membrane or cell membrane. There exist three kinds of endocytosis: *phagocytosis*, *pinocytosis*, and *receptor–mediated endocytosis*. In particular, phagocytosis (literally, cell–eating) is the process by which cells ingest large objects, such as cells which have undergone apoptosis, bacteria, or viruses. The membrane folds around the object, and the object is sealed off into a large vacuole known as a phagosome. Pinocytosis (literally, cell–drinking) is concerned with the uptake of solutes and single molecules such as proteins, and, finally, receptor–mediated endocytosis is a more specific active event where the cytoplasm membrane folds inward to form coated pits. These inward budding vesicles bud to form cytoplasmic vesicles [11]. Figure 1 summarizes the kinds of endocytosis. From the point of view of the modeler, these three processes are made possible by vesicles (in fact, this transportation mechanism is known as *vesicle–mediated transportation*) which, in the most general case, engulf the macromolecules together with molecules from the surface of the membranes (i.e., receptors). This leads to the creation of *endosomes* containing the engulfed molecules. The endosomes transfer their content inside the cell by possibly interacting with other components. The endosomes could also be degraded by the interaction with the lysosomes. We define an extension of P Systems (PE Systems) which can explicitly model the creation of endosomes and their interaction inside the cells and, consequently, can easily model these three kinds of endocytosis.

This variant of P Systems, together with other modeling features such as the modeling of exocytosis (the biologically counterpart of endocytosis), and enriched with channel–mediated communication [3], would provide a powerful and complete modeling language for naturally describing transportation mechanism of molecules inside cells.

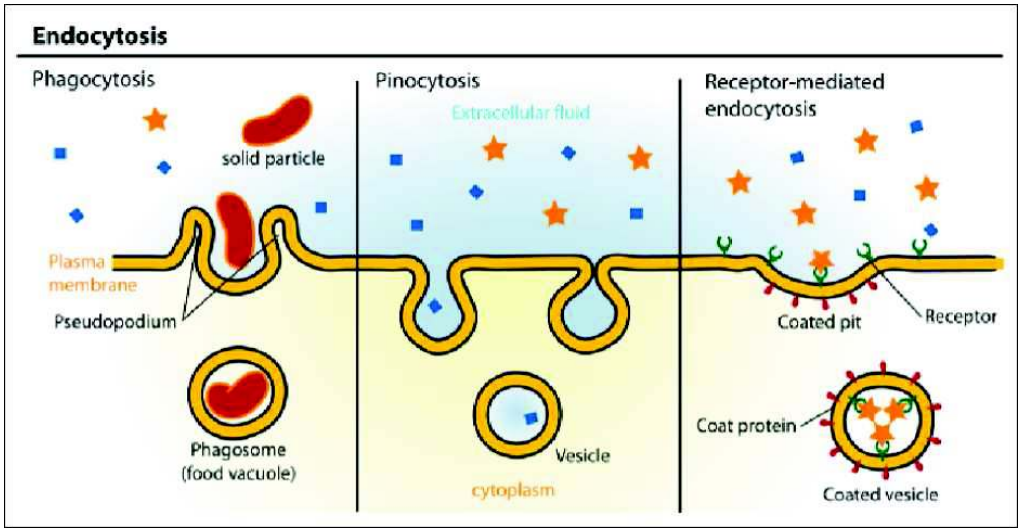


Figure 1: Three kind of endocytosis: *phagocytosis*, *pinocytosis* and *receptor–mediated endocytosis*. Picture taken from http://cellbiology.med.unsw.edu.au/units/science/lecture0806.htm

We show that PE Systems are universal even if only the simplest form of evolution rules is considered, namely non–cooperative rules. Finally, we give one application example to show that endosomes can ease the description of biological systems when PE Systems are used as a modeling formalism.

2 P Systems with Endosomes

In this section we formally define P Systems with endosomes (PE Systems). We assume the reader to be familiar with the standard definition of P Systems [11]. We start by assuming the same membrane structure μ of a P System. As regards objects, similarly to P Systems with peripheral proteins [6, 13], we assume that objects can be contained inside a membrane (as in classical P Systems) and on the surface of a membrane. In order to qualify in an evolution rule the position of an object with respect to a membrane, we use *in* to identify the object inside the membrane, *out* to identify the object outside the membrane and *here* to identify the object on the surface of the membrane. Let TAR be the set of message targets $\{in, out, here\}$; given a set of objects V we denote with V_{tar} the corresponding set of messages $O \times TAR$.

We can now introduce the evolution rules of PE Systems; rules are conceptually divided in evolution rules (in the same sense of P Systems) and rules for the creation of endosomes. We recall that, differently from P Systems, the rules of PE Systems are conceptually associated with the surfaces of the membranes of the system. Evolution rules are of the form $u \rightarrow v$ where $u \in V_{tar}^{+}$, $v \in V_{tar}^{*}$, $V_{tar}^{+} = V_{tar}^{*} \backslash \{\varepsilon\}$ and ε is the empty string. The definition of cooperative and non–cooperative rules are the same as for P Systems.

Notice that this format for evolution rules, which are syntactically different from those of P Systems, may seem to be less expressive than the one of P Systems, in particular for rule moving objects into specific regions enclosed by membranes (communication rules). In order to show that this is not the case, let us assume an hypothetical membrane structure μ such that $(l, l') \in \mu$, namely a membrane structure in which l' is nested into l. In order to give a rule which moves an object inside membrane l' we cannot use the identifier $in_{l'}$ in a rule of the surface of the membrane l (as in P Systems) because we cannot use the identifier l' as subscript to *in*. However, the same behaviour can be obtained by replacing the rule $u \rightarrow (v, in_{l'})$ in the membrane l, as in usual P Systems, with the PE System rule $(u, out) \rightarrow (v, in)$ on the surface of the membrane l'. The behaviour modeled by this rule, which is in some sense an "attraction" by the nested membrane rather than the "sending" from the top membrane, leads to results analogous to those obtained by P Systems, namely to the transportation of the object inside the nested membranes.

The rules for creating endosomes are of the form $endo_E(u, v)$, where $u, v \in V^*$ and:

- E is a set of evolution rules for the endosome;
- u is the multiset of objects that must appear on the surface of the membrane containing the rule;
- v is the multiset of objects that must appear outside the membrane containing the rule.

Note that each endosome has got its own evolution rules in set E. These rules model the behaviour of the endosome. As regards the creation of an endosome, it is necessary that objects in u are present on the surface of the membranes (they can be seen as the receptors) and that objects in v are present outside of the membrane creating the endosome (they can be seen as the molecules to be engulfed). We remark that in our endosome rules, the objects inside and on the surface of the created endosome are explicitely defined. This is different from the approach of [5] in which in the case of pino rules the surface objects are randomly distributed to the two resulting membranes. More formally, the applicability of an endosome rule is possible in the following general case: let $(j, i) \in \mu$ and let $endo_E(u, v)$ be a rule belonging to the surface of the membrane i, then it can be applied only if u is a submultiset of the objects contained on the surface of the membrane i, and only if v is a submultiset of the objects contained inside the membrane j. The result of the application of such a rule is the creation of an endosome inside membrane i containing u on its surface and containing v inside. The endosome itself behaves like a membrane having on its surface rules E.

We can now formally define a PE System as follows.

Definition 1. A *PE System* Π is a tuple $(V, \mu, w_1, \ldots, w_n, z_1, \ldots, z_n, R_1, \ldots, R_n)$ where:

- V is an *alphabet* whose elements are called *objects*;

- $\mu \subset \mathbb{N} \times \mathbb{N}$ is a *membrane structure*;
- w_i with $1 \leq i \leq n$ are strings from V^* representing multisets over V associated with the content of membranes $1,2,\ldots,n$ of μ;
- z_i with $1 \leq i \leq n$ are strings from V^* representing multisets over V associated with the surfaces of membranes $1,2,\ldots,n$ of μ;
- R_i with $1 \leq i \leq n$ are finite sets of *evolution* and *endosome* rules associated with the surfaces of the membranes $1,2,\ldots,n$ of μ.

The notions of (successful) computation and of result of computations of PE Systems are the same as for standard P Systems.

3 Universality of PE Systems

In this section we prove a universality result for PE Systems by showing that any matrix grammar with appearance checking can be simulated by a PE System. Before giving the result and its proof, we recall from [11] the definition of this variant of matrix grammars and some related notions.

3.1 Matrix grammars with appearance checking

A (context-free) matrix grammar with appearance checking is a tuple $G = (N,T,S,M,F)$, where N and T are disjoint alphabets of non–terminals and terminals, respectively, $S \in N$ is the axiom, M is a finite set of matrices, namely sequences of the form $(A_1 \to x_1,\ldots,A_n \to x_n)$ of context–free rules over $N \cup T$ with $n \geq 1$, and F is a set of occurrences of rules in the matrices of M. For a string w, a matrix $m : (r_1,\ldots,r_n)$ can be executed by applying its rules to w sequentially in the order in which they appear in m. Rules of a matrix occurring in F can be skipped during the execution of the matrix if they cannot be applied, namely if the symbol in their left–hand side is not present in the string.

Formally, given $w,z \in (N \cup T)^*$, we write $w \Longrightarrow z$ if there is a matrix $(A_1 \to x_1,\ldots,A_n \to x_n)$ in M and the strings $w_i \in (N \cup T)^*$ with $1 \leq i \leq n+1$ such that $w = w_1$, $z = w_{n+1}$ and, for all $1 \leq i \leq n$, either (1) $w_i = w'_i A_i w''_i$ and $w_{i+1} = w'_i x_i w''_i$, for some $w'_i, w''_i \in (N \cup T)^*$, or (2) $w_i = w_{i+1}$, A_i does not appear in w_i and the rule $A_i \to x_i$ appears in F. We remark that F consists of *occurrences* of rules in M, that is, if the same rule appears several times in the matrices, it is possible that only some of these occurrences are contained in F.

The language generated by a matrix grammar with appearance checking G is defined as $L(G) = \{w \in T^* \mid S \Longrightarrow^* w\}$, where $\Longrightarrow^* w$ is the reflexive and transitive closure of $\Longrightarrow$. The family of languages of this form is denoted by MAT^{λ}_{ac}, when rules having the empty string λ as right hand side (λ–rules) are allowed, and by MAT_{ac} when such rules are not allowed. Moreover, the family of languages generated by matrix grammars without appearance checking (i.e., with $F = \emptyset$) is denoted by MAT^{λ}, when λ–rules are allowed, and by MAT, when such rules are not allowed. It is known that (1) $MAT \subset MAT_{ac} \subset CS$; (2) $MAT^{\lambda} \subset MAT^{\lambda}_{ac} = RE$, where CS and RE are the families of languages generated by context–sensitive and arbitrary grammars, respectively.

Let $ac(G)$ be the cardinality of F in G and let $|x|$ denote the length of the string x. A matrix grammar with appearance checking $G = (N,T,S,M,F)$ is said to be in the *strong binary normal form* if $N = N_1 \cup N_2 \cup \{S,\#\}$, with these sets mutually disjoint, $ac(G) \leq 2$ and the matrices in M are in one of the following forms:

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

4. $(X \to \lambda, A \to x)$, with $X \in N_1, A \in N_2, x \in T^*, |x| \leq 2$.

Moreover, there is only one matrix of type 1, and F consists exactly of all rules $A \to \#$ appearing in matrices of type 3. We remark that # is a trap symbol, namely once introduced it cannot be removed, and a matrix of type 4 is used only once, in the last step of a derivation.

For each matrix grammar (with or without appearance checking) there exists an equivalent matrix grammar in the strong binary normal form. Consequently, for each language $L \in RE$ there exists a matrix grammar with appearance checking G satisfying the strong binary normal form and such that $L(G) = L$.

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

Since the grammar is in strong binary normal form, we have (at most) two symbols $B^{(1)}$ and $B^{(2)}$ in N_2 such that the rules $B^{(j)} \to \#$ appear in matrices m_i with $k+1 \leq i \leq n$.

We remark that in matrix grammars in strong binary normal forms we can assume that all symbols $X \in N_1$ appear as the left-hand side of a rule from a matrix: otherwise, the derivation is blocked after introducing such a symbol, hence we can remove these symbols and the matrices involving them.

3.2 Universality

We prove that PE Systems are universal by showing that the family, denoted $PsE_2(ncoo)$, of sets $Ps(\Pi_e)$ of results computed by PE Systems with at least two membranes and with non–cooperative rules is equivalent to the family, denoted $PsRE$, of the images of all the languages in RE obtained through the Parikh mapping (this is the family of recursively enumerable sets of vectors of natural numbers). As P Systems with non-cooperative rules are not universal, our result implies that universality is due to the presence of endosomes.

Theorem 2. $PsE_2(ncoo) = PsRE$.

Proof. It is enough to show that for a matrix grammar G in strong binary normal form there is a PE System Π_G such that $Ps(\Pi_G) = \Psi_T(L(G))$. We assume that the output of this PE System is given by the objects sent out from the skin membrane. The alphabet of objects V we take into consideration is given by $T \cup N_1 \cup N_2 \cup \{c\} \cup \{c_i, d_i, d'_i \mid i = 1, 2\}$. We build Π_G as a system with a root membrane, labeled 1, and one child membrane labeled 2, namely $\mu = \{(1, 2)\}$. All the objects encoding the grammar will be stored inside membrane 1 and the matrices will be simulated by membrane 2. The initial configuration is given by the objects corresponding to X_{init} and A_{init} contained in membrane 1, namely objects of w_1, and by the token c contained on the surface of membrane 2, namely $z_2 = \{c\}$. Differently, w_2 and z_1 are initially empty multisets.

This PE System works as follows: it has a cyclic behaviour such that, at the beginning of the cycle, at most one endosome in membrane 2 can be created and, if possible, all terminal symbols inside membrane 1 are sent out as output symbols. The created endosome can start a series of steps resulting in the interpretation of the application of a matrix or, differently, it can start a checking phase to model the fact that, if there exist non–terminal symbols which cannot be rewritten by any grammar, then the computation will not halt. In the case in which Π_G starts the simulation of a matrix of type 2 or 4 (a matrix m_i with $1 \leq i \leq k$), the involved non–terminals are taken by the endosome which contains as rules the ones interpreting the matrix. Objects will be sent into membrane 2 by these rules creating the result of the

application of the the corresponding matrix to the non–terminals. Subsequently, these objects are sent out to membrane 1 to restart the cyclic behaviour. We recall that during this process no other endosomes can be created, hence no other matrices can be simulated. Differently, in the case in which a matrix of type 3 (a matrix m_i with $k+1 \leq i \leq n$) is applied, the single non–terminal of N_1 is taken into the endosome. The endosome will work in the same sense of the endosomes interpreting matrices of type 2 and 4 even though, at the end of the application of this matrix, instead of restarting with the cyclic behaviour, a checking process is started. This process checks, by creating endosomes, the presence of the proper non–terminal symbol $B^{(j)}$. If this symbol is found, a special endosome is created which will introduce a trap symbol in this PE System so that the computation will not halt. Analogously, if the symbol is not found, an endosome will restore the configuration of this PE System so that the cyclic behaviour can start again.

We list now the rules of Π_G. Membrane 1 contains just one single set of rules to create the output of the PE System:

1. $\{(a,in) \rightarrow (a,out) \mid \forall a \in T\}$. All terminal objects in membrane 1 are sent out as output.

The simulation of any matrix is done by the rules of membrane 2 which are the following:

1. $\forall X \in N_1 \cup N_2$. $endo_{(X,in)\rightarrow(\#,out)}(c,X)$. If any non–terminal is present in membrane 1, Π_G will always be able to create, by using an endosome, a trap symbol inside membrane 2. This will ensure that, if a derivation of G reaches a deadlock configuration, then Π_G can always enter an endless configuration.
2. $\forall a \in N_1 \cup N_2 \cup T$. $(a,in) \rightarrow (a,out)$. Every terminal and non–terminal present inside membrane 2 is sent out to membrane 1.
3. $(c,in) \rightarrow (c,here)$. Object c inside membrane 2 is restored on the surface of membrane 2 so that other endosomes can be created.
4. $(\#,in) \rightarrow (\#,in)$. The trap symbol lets this computation not to be recognized such.
5. $m_i : (X \rightarrow \alpha, A \rightarrow x), 1 \leq i \leq k$. $endo_{(X,in)\rightarrow(\alpha,out),(A,in)\rightarrow(x,out),(c,here)\rightarrow(c,out)}(c,XA)$. For any rule of type 2 and 4, we create an endosome by taking XA from membrane 1 and c from the surface of membrane 2 (this locks the creation of other endosomes). The endosome contains rules to rewrite X and A with the result of applying the matrix. Object c is not consumed and sent out to membrane 2 together with α and x.
6. $m_i : (X \rightarrow Y, A \rightarrow \#), k+1 \leq i \leq n$. $endo_{(X,in)\rightarrow(Y,out),(c,here)\rightarrow(c_i,out)}(c,X)$. For any rule with appearance checking, we create an endosome by taking only X from membrane 1 and c from the surface of membrane 2 (this locks the creation of other endosomes). The endosome contains rules to rewrite X with Y and c with c_i. Both objects are sent out to membrane 2.
7. $(c_i,in) \rightarrow (c_i,here)(d_i,here)$. Object c_i, together with a new object d_i, is moved on the surface of membrane 2.
8. $endo_{(B^{(i)},in)\rightarrow(\#,out)}(c_i,B^{(i)})$. This implements the appearance checking feature of grammar G. We create, if possible, an endosome by taking only $B^{(i)}$ from membrane 1 and c_i from the surface of membrane 2. The endosome creates a trap symbol in membrane 2; this will make Π_G start an endless computation.
9. $(d_i,here) \rightarrow (d_i',here)$. The symbol d_i is rewritten in the same place as d_i'. This is done even if also rule 8 can be applied. However, in the case that rule 8 cannot be applied (namely $B^{(i)}$ was not present), this completes the appearance checking operation and lets Π_G start an operation which will restart its cyclic behaviour.
10. $endo_{(c_i,here)\rightarrow(c,out),(d_i',here)\rightarrow\lambda}(c_i d_i', \emptyset)$. This endosome lets Π_G restart its cyclic behaviour. We create an endosome by simply taking both the control symbols only c_1 and d_i' from the surface of membrane 2. The endosome destroys d_i' and rewrites c_i with c in membrane 2 (restarting Π_G will be obtained by applying rule 3).

It is clear that these rules, applied in a proper order, provide the correct interpretation of the application of any matrix to the starting symbols of the grammar and, consequently, we get $Ps(\Pi_G) = \Psi_T(L(G))$ which concludes the proof. □

4 An Application: the EGF Signaling Pathway

In this section we give an application of PE systems to the description of the initial phases of the EGFR signalling cascade.

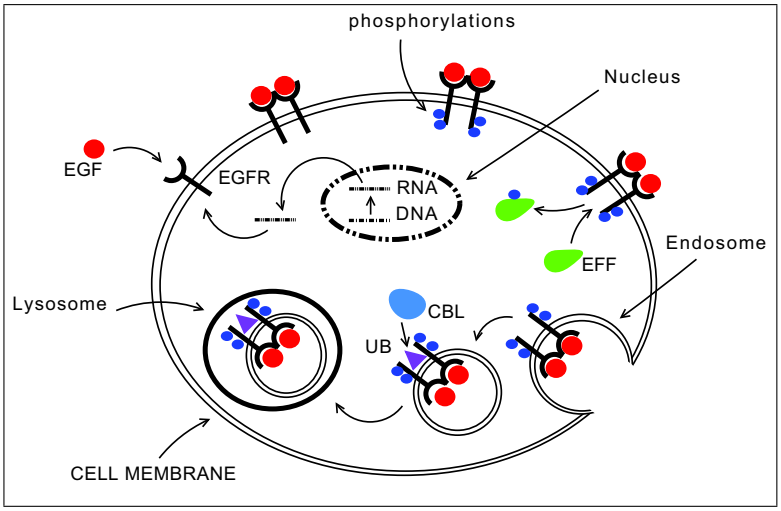


Figure 2: The EGF signaling pathway (picture taken from [3]).

In Biology, signal transduction refers to any process by which a cell converts one kind of signal or stimulus into another. Signals are typically proteins that may be present in the environment of the cell. In order to be able to receive the signal, namely to recognize that the corresponding ligand is available in the environment, a cell exposes some receptors on its external membrane. A receptor is a transmembrane protein that can bind to a signal protein on its extracellular end. When such a binding is established, the intracellular end of the receptor undergoes a conformational change that enables interaction with other proteins inside the cell. This typically causes an ordered sequence of biochemical reactions inside the cell, usually called signalling pathway, that are carried out by enzymes and may produce different effects on the cell behaviour.

A complex signal transduction cascade, that modulates cell proliferation, survival, adhesion, migration and differentiation, is based on a family of receptors called epidermal growth factor receptors (EGFRs). While EGFR signalling is essential for many normal morphogenic processes, the aberrant activity of these receptors has been shown to play a fundamental role in proliferation of tumor cells. Epidermal growth factor receptors ($EGFR$) are produced by specific genes in the DNA (through the RNA) and they are located on the cell surface. Receptors are activated by the binding with a specific ligand (epidermal growth factor, EGF) to form a EGFR (ligand-receptor) complex (COM). Upon activation, EGFR undergoes a transition from a monomeric form to an active dimeric one (DIM). EGFR dimerization stimulates its intracellular phosphorylation ($DIMp$) which activates signalling proteins. These activated signalling proteins (effector proteins) initiate several signal transduction cascades, leading to DNA synthesis and cell proliferation. After the activation of effector proteins, ligand-receptor dimers are internalized in endosomes. An ubiquitin ligase, known as Cbl, binds an ubiquitin protein (UB) to the dimer (ubiquitination). The ubiquitin protein targets the dimers for lysosomal degradation (see Figure 2).

The PE system modeling the EGF is given in Figure 3. Membrane 1 models the environment external

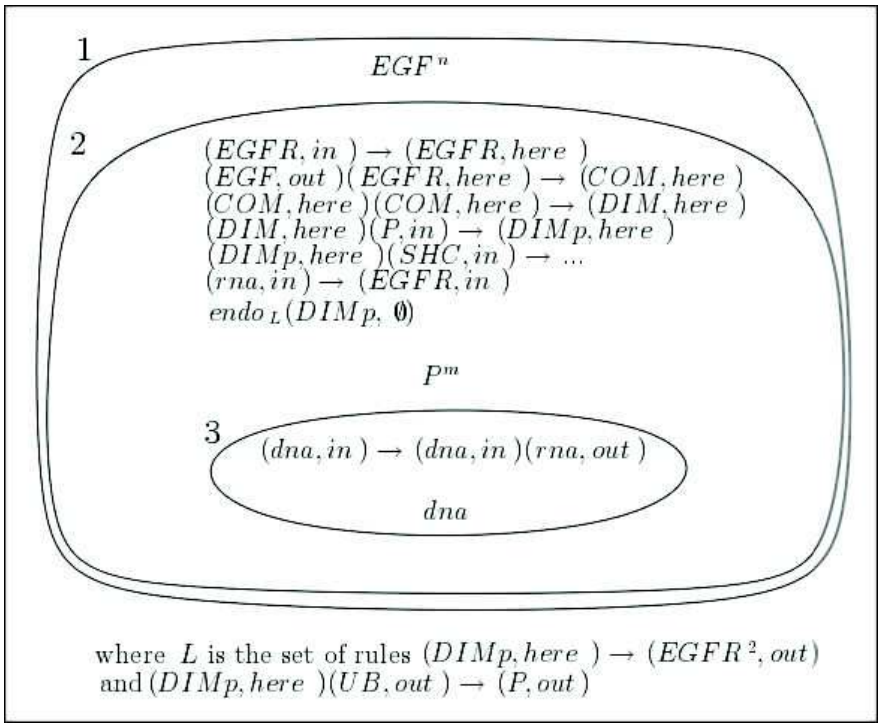


Figure 3: A PE systems model of the EGF signaling pathway. The rules are represented inside the membranes.

to the cell, membrane 2 represents the cell surface and membrane 3 is the nucleus. In the external environment EGF corresponds to the epidermal growth factor EGF which can bind the receptor on the surface of the cell. The receptor is modeled by $EGFR$ in membrane 2, which can move on the surface of the membrane. The complex of EGF with the receptor is obtained by rewriting EGF and $EGFR$ with the complex COM on the surface of membrane 2. After the binding of two complexes we can bind them leading to a dimer DIM. Such a dimer, present on the surface of the membrane, can be phosphorylated by a phosphorus P inside the cell. Such phosphorilated dimer $DIMp$ could interact with protein SHC and start a chain of interactions we do not model here aimed at activating cell proliferation. Furthermore, it can be enclosed in an endosome which could either decompose the $DIMp$ dimer into its original components (in order to recycle the two $EGFR$ proteins) or, if ubiquitine UB is present, degradate the $DIMp$ dimer and release the phosporus. The nucleus of the cell (membrane 3) is responsible for the production of $EGFR$ through the DNA and RNA (*dna* and *rna*). The *rna* reaches the cell cytoplasm and there it produces $EGFR$ which is sent, again, to the cell surface.

5 Future Work and Conclusions

In this paper we have presented an extension of P Systems, called P Systems with Endosomes (PE Systems), in which endosomes can be explicitly modeled. PE Systems uses some ideas taken from other variants of P Systems, in particular as regards objects which can be stored on the surface of the membranes we got inspiration by P Systems with peripheral proteins [6, 13] and by membrane systems with surface objects [1, 2]. Furthermore, as regards other calculi, operations for modeling transportation mechanisms have already been introduced in Brane Calculi [4] and in P Systems with transport and embedded proteins [9]. Although similar, PE Systems permit to model in a clearer way these mechanisms. An analysis of PE Systems and Brane Calculi [4] (and also some of their variants like projective Brane

Calculi [7]) could be done along the line of the one done in [5, 12] for P Systems and Brane Calculi.

As regards expressiveness of this formalism, we have shown that PE Systems are universal even if only the simplest form of evolution rules is considered, namely non–cooperative rules. This expressiveness is achieved by the use of endosomes as classical P Systems with this kind of rules are shown not to be universal [10].

At the end of the paper we have given an application example describing the modeling of the initial phases of the EGFR signalling cascade.

Bibliography

[1] Aman, B., Ciobanu, G.: Membrane Systems With Surface Objects. Proceedings of the Int. Workshop on Computing with Biomolecules (CBM 2008), Vienna, 17–29, 2008.

[2] Aman, B., Ciobanu, G.: Mutual Mobile Membrane Systems With Objects on Surface. Proceedings of the Seventh Brainstorming Week on Membrane Computing (BWMC09), Seville, 2009.

[3] Barbuti, R., Maggiolo–Schettini, A., Milazzo, P. Tini, S.: P Systems with Transport and Diffusion Membrane Channels. Int. Workshop on Concurrency, Specification and Programming (CS&P'08), Gross Vaeter, Germany, September, 2008.

[4] Cardelli, L.: Brane calculi. Interactions of biological membranes. In: Danos, V., Schachter, V. (Eds.), LNCS **3082** (2005), pp. 257-280.

[5] Cardelli, L., Păun, G.: An universality result for a (mem)brane calculus based on mate/drip operations. Internat. J. Found. Comput. Sci. **17**(1), pp. 49–68.

[6] Cavaliere, M., Seawards, S.: Membrane systems with peripheral proteins: transport and evolution. Proc. of the First Workshop on Membrane Computing and Biologically Inspired Process Calculi (MeCBIC 2006), ENTCS **171** (2007), pp. 37–53.

[7] Danos, V. Pradalier, S.: Projective Brane Calculus. Proc. of the Fourth Conference on Computational Methods in Systems Biology (CMSB04), LNCS **3082** (2005), pp. 134–148.

[8] Freund, R., Oswald, M.: P systems with activated/prohibited membrane channels. Proc. of WMC 2002, LNCS **2597** (2003), pp. 261–269.

[9] Krishna, S.N.: Membrane computing with transport and embedded proteins. Theoretical Computer Science **410** (2009), pp. 355–375.

[10] Păun, G.: Computing with membranes. Journal of Computer and System Sciences **61** (2000), pp. 108–143

[11] Păun, G.: Membrane Computing. An Introduction. Springer (2002).

[12] Păun, G.: Membrane computing and brane calculi. Old, new, and future bridges. Theoretical Computer Science **404**(1-2), pp. 19–25.

[13] Păun, A., Popa, B.: P Systems with Proteins on Membranes. Fundamenta Informaticae **72**(4) (2006), pp. 467 – 483.

[14] P Systems, web page. `http://ppage.psystems.eu/`.

Roberto Barbuti was born in 1953. He received a Laurea degree in Computer Science in 1977 from the University of Pisa. In 1982 he became Assistant Professor at University of Pisa. He took the position of Associate Professor in 1989, and the position of Full Professor in 2000. Presently, he is working in the research fields of Formal Methods for Systems Biology.

Giulio Caravagna was born in 1982. He received a Bachelor and a Master Degree in Computer Science from the Univeristy of Pisa in years 2005 and 2007, respectively. At present, he is a PhD student in Computer Science working in the fields of Formal Methods for Systems Biology and Natural Computing.

Andrea Maggiolo Schettini was born in 1938. He received a laurea degree in Physics from the University of Genova and from 1966 to 1968 he was a researcher of the Italian National Institute for Nuclear Physics in Bologna. After shifting his interest to fundamental studies in computer science, from 1968 to 1981 he was a researcher of the National Research Council in Naples and in Pisa. From 1981 to 1983 he has been full professor of Computer Science at the University of Turin and since 1983 he is full professor of Computer Science at the University of Pisa. He has done research in computability theory, semantics of programming languages, specification and verification of concurrent and distributed systems. His present interests include Systems Biology and Natural Computing.

Paolo Milazzo was born in 1979. He received a Master Degree in Computer Science in 2003 at the University of Bologna and a Ph.D. in Computer Science in 2007 at the University of Pisa. Actually, he is research fellow at the Department of Computer Science of the University of Pisa. working on Computational Systems Biology and Natural Computing with a particular focus on the definition and application of Formal Methods.

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 224-233

Variants of P Colonies with Very Simple Cell Structure

Lucie Ciencialová, Erzsébet Csuhaj-Varjú, Alica Kelemenová, György Vaszil

Lucie Ciencialová, Alica Kelemenová
Institute of Computer Science, Faculty of Philosophy and Science, Silesian University in Opava
Bezručovo nám. 13, 74601 Opava, Czech Republic
E-mail: {lucie.ciencialova, alica.kelemenova}@fpf.slu.cz

Erzsébet Csuhaj-Varjú, György Vaszil
Computer and Automation Research Institute of the Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
E-mail: {csuhaj, vaszil}@sztaki.hu

Erzsébet Csuhaj-Varjú
Department of Algorithms and Their Applications, Faculty of Informatics, Eötvös Loránd University
Pázmány Péter sétány 1/c, 1117 Budapest, Hungary

Alica Kelemenová
Department of Computer Science, Catholic University Ružomberok
Nám. A. Hlinku 56, 03401 Ružomberok, Slovakia

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: We study two very simple variants of P colonies: systems with only one object inside the cells, and systems with insertion-deletion programs, so called P colonies with senders and consumers. We show that both of these extremely simple types of systems are able to compute any recursively enumerable set of vectors of non-negative integers.
Keywords: P systems, colonies, P colonies, register machines

1 Introduction

P colonies form a class of abstract computing devices modeling a community of simple agents acting and evolving in a shared environment. They were introduced in [5] as very simple membrane systems, similar in simplicity and architecture to so called colonies of formal grammars. (See [7] for more information on membrane systems and [2, 4] for details on grammar systems theory.)

A P colony consists of a collection of cells, each having a number of objects inside and an associated set of rules through which it can process these objects. Communication between the cells is only possible indirectly through the environment which is common to all of them.

The capabilities of the computing agents are very restricted, and the number of objects present inside a cell during the functioning of the system is previously fixed: it is usually one, two or three. The rules are also of a very simple form. As we will see, they allow the transformation of objects inside the cells and the transportation of objects between the cells and the environment. The rules are grouped into programs. A program contains exactly as many rules, as the number of objects allowed to be present inside the cell. The rules of the programs are applied to the objects inside the associated cells in parallel, and this also affects the objects which are in the environment.

The P colony executes a computation by synchronously applying the programs to the objects inside the cells and outside in the environment until a halting configuration is reached. The result of the computation is obtained as the vector of copies of certain "final" objects present in the environment after the system halts.

Research supported in part by the Czech Science Foundation, "GAČR", project no. 201/09/P075 (L. Ciencialová), by the Hungarian Scientific Research Fund, "OTKA", project no. K75952 (E. Csuhaj-Varjú, Gy. Vaszil), and by the Scientific Grant Agency of Slovakia, "VEGA", project

In the following, after providing the formal definitions, we first give a short overview of results on the computational completeness of the different P colony variants. Then we present new results about two types of systems: first about the simplest possible P colonies, those which only have one object inside every cell, and then about a new type called P colonies with senders and consumers, which have special rules for insertion-deletion. We show that both kinds of these very simple devices are able to compute any recursively enumerable set of vectors of non-negative integers.

2 Preliminaries

Let V be an alphabet, let V^* be the set of all words over V, and let ε denote the empty word. We denote the number of occurrences of a symbol $a \in V$ in w by $|w|_a$. The set of non-negative integers is denoted by $\mathbb{N}$.

A multiset over an arbitrary (not necessarily finite) set V is a mapping $M : V \to \mathbb{N}$ which assigns to each object $a \in V$ its multiplicity $M(a)$ in M. The support of M is the set $supp(M) = \{a \mid M(a) \geq 1\}$. If V is a finite set, then M is called a finite multiset. A multiset M is empty if its support is empty, $supp(M) = \emptyset$. We will represent a finite multiset M over V by a string w over the alphabet V with $|w|_a = M(a)$, $a \in V$, and ε will represent the empty multiset.

We will also need the notion of a register machine which consists of a finite number of registers each of which can hold an arbitrarily large non-negative integer (we say that the register is empty if it holds zero), and a set of labeled instructions which specify how the numbers stored in the registers can be changed.

Formally, a *register machine* is a construct $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halting label, and R is the set of instructions. Each label from H labels only one instruction from R. There are several types of instructions which can be used. For $l_i, l_j, l_k \in H$ and $r \in \{1, \ldots, m\}$ we have

- $l_i : (ADD(r), l_j, l_k)$ - *nondeterministic add*: Add one to register r and then go to one of the instructions with labels l_j or l_k, non-deterministically chosen.
- $l_i : (SUB(r), l_j, l_k)$ - *subtract*: If register r is non-empty, then subtract one from it and go to the instruction with label l_j, if the value of register r is zero, go to instruction l_k.
- $l_h : HALT$ - *halt*: Stop the machine.

A register machine M computes a set $N(M)$ of numbers in the following way: It starts with empty registers by executing the instruction with label l_0 and proceeds by applying instructions as indicated by the labels (and made possible by the contents of the registers). If the halt instruction is reached, then the number stored at that time in register 1 is said to be computed by M. Because of the non-determinism in choosing the continuation of the computation in the case of *ADD* instructions, $N(M)$ can be an infinite set.

It is known (see, e.g., [6]) that in this way we can compute all sets of numbers which are Turing computable.

If a set of output registers $i_1, \ldots, i_r$, $1 \leq r \leq m$, $i_j \in \{1, \ldots, m\}$ is specified, then M computes a set of vectors of non-negative integers as follows. If the halt instruction is reached, then $(v_1, \ldots, v_r)$, where v_k is the number stored in register i_k, $1 \leq k \leq r$, is the vector of numbers computed by M, i.e., the result of that computation.

Now we recall the definition of a P colony from [5]. A *P colony* is a construct $\Pi = (V, e, F, C_1, \ldots, C_n)$, $n \geq 1$, where V is an alphabet (its elements are called *objects*). There are two kinds of distinguished objects: $e \in V$ (the environmental object), and the objects in $F \subseteq V$ (the set of final objects). The *cells* of the colony are denoted by $C_1, \ldots, C_n$. Each cell is a pair $C_i = (O_i, P_i)$, where O_i is a multiset over $\{e\}$

having the same cardinality *called capacity* (here we only consider $|O_i| \in \{1,2\}$) for all i, $1 \leq i \leq n$ (the initial state of the cell), and P_i is a finite set of *programs*. Each program consists of rules of the following forms:

- $a \to b$ (internal point mutation), specifying that an object $a \in V$ inside the cell is changed to $b \in V$.
- $c \longrightarrow d$ (one object exchange with the environment), specifying that if $c \in V$ is contained inside the cell and $d \in V$ is present in the environment, then c is sent out of the cell while d is brought inside.
- $c \longrightarrow d/c \longrightarrow d'$ (checking rule for one object exchange with the environment), specifying that if $c \in V$ is inside the cell then it is exchanged with $d \in V$ from the environment, or if there is no d outside but $d' \in V$ is present, then c is exchanged with d'.
- $c \longrightarrow d/c \to d'$ (checking rule for one object exchange with the environment or internal point mutation), specifying that if the exchange of $c \in V$ inside and $d \in V$ outside is not possible, then c is changed to $d' \in V$.

The programs contain one rule for each element of O_i, thus, the number of rules of a program coincides with the cardinality of O_i, $1 \leq i \leq n$.

In addition, P colonies with capacity of two may have programs of the form

- $\langle a, in; bc \to d \rangle$ with $a,b,c,d \in V$ (deletion programs), specifying that if bc is present inside the cell and a is present in the environment, then the objects inside are changed to d and a is brought in (a is "deleted" from the environment).
- $\langle a, out; b \to cd \rangle$ with $a,b,c,d \in V$ (insertion programs), specifying that if ab is inside the cell, then a is sent out (a is "inserted" into the environment) and b is changed to cd.

The programs of the cells are used in the non-deterministic maximally parallel manner: in each time unit, each cell which is able to use one of its programs should use one. The use of a program means the application of the rule(s) of the program to the object(s) in the cell.

This way, transitions among the configurations of the colony are obtained. A sequence of transitions is a *computation* which is halting if it reaches a configuration where no cell can use any program. The result of a halting computation is obtained from the number of copies of objects from F present in the environment in the halting configuration. Because of the non-determinism in choosing the programs, several computations can be obtained from a given initial configuration, hence with a P colony Π we can associate a set of vectors of non-negative integers computed by all possible halting computations of Π.

Initially, the environment contains arbitrarily many copies of the environmental object e, and the cells also contain one or two copies of e inside, depending on the capacity of the P colony.

For a P colony $\Pi = (V, e, F, C_1, \ldots, C_n)$ as above, a configuration can be formally written as an $(n+1)$-tuple

$$(w_1, \ldots, w_n; w_E),$$

where $w_i \in V^*$ represents the multiset of objects from cell C_i, $1 \leq i \leq n$, and $w_E \in (V - \{e\})^*$ represents the multiset of objects from the environment different from the environmental object e. The initial configuration is $(e^i, \ldots, e^i; \varepsilon)$ where $i \in \{1,2\}$ is the capacity of the cells.

A transition from a configuration to another is denoted as

$$(w_1, \ldots, w_n; w_E) \Rightarrow (w'_1, \ldots, w'_n; w'_E)$$

where w'_E and each w'_i is obtained from w_i, $1 \leq i \leq n$, and w_E by executing one of the programs of P_i.

The set of vectors in $\mathbb{N}^m$, $m = |F|$, $F = \{o_1, \ldots, o_m\}$, computed by a P colony Π is defined as

$$N(\Pi) = \{(|v_E|_{o_1}, \ldots, |v_E|_{o_m}) \mid (e^i, \ldots, e^i; \varepsilon) \Rightarrow^* (v_1, \ldots, v_n, v_E)\}$$

where $(e^i, \ldots, e^i, \varepsilon)$, $i \in \{1,2\}$, is the initial configuration, $(v_1, \ldots, v_n, v_E)$ is a halting configuration, and $\Rightarrow^*$ denotes the reflexive and transitive closure of $\Rightarrow$.

Let us denote by $PCOL(i, j, k, check)$ and $PCOL(i, j, k, no\text{-}check)$ the classes of sets of vectors generated by P colonies with at most $j \geq 1$ cells of capacity $i \in \{1,2\}$, having at most $k \geq 1$ programs associated to a cell which contain or do not contain checking rules, respectively. If a numerical parameter is unbounded, we denote it by a $*$.

P colonies can simulate register machines with a rather limited number of programs per cell. In [3], it was shown that

$$PCOL(2, *, 4, check) = PCOL(3, *, 3, check) = \mathbb{N}RE$$

where $\mathbb{N}RE$ denotes the class of recursively enumerable sets of integer vectors. Even one cell is enough, if it may have an arbitrarily large number of programs, that is,

$$PCOL(2, 1, *, check) = \mathbb{N}RE.$$

Similar results were also obtained without the use of checking rules. In this case we have

$$PCOL(2, *, 8, no\text{-}check) = PCOL(3, *, 7, no\text{-}check) = \mathbb{N}RE.$$

3 P colonies with one object

In [1] it was shown that if checking rules are allowed to be used, then all recursively enumerable sets of vectors can even be generated by P colonies with capacity one, that is,

$$PCOL(1, 4, *, check) = \mathbb{N}RE.$$

In the following we show that P colonies with six components generate all vectors even if checking rules are not used.

Theorem 1. $PCOL(1, 6, *, no\text{-}check) = \mathbb{N}RE$.

Proof. We construct a P colony simulating the computations of a register machine. Let us consider an m-register machine $M = (m, H, l_0, l_h, P)$ and represent the content of the register i by the number of copies of a specific object a_i in the environment. We construct the P colony $\Pi = (V, e, F, C_1, \ldots, C_6)$ with:

$$\begin{aligned} V &= \{e, l_i, l_i', l_i'', \bar{l}_i, K_i, L_i, L_i', L_i'', L_i''', E_i, F_i, \$_i \mid \text{for each } l_i \in H\} \cup \\ &\quad \{a_i, a_{i,j} \mid 1 \leq i \leq m,\ 1 \leq j \leq |H|\} \cup \{D, D', T\}, \\ F &= \{a_i \mid \text{register } i \text{ is an output register}\}, \text{ and} \\ C_i &= (e, P_i), \text{ for } 1 \leq i \leq 6. \end{aligned}$$

Because initially there are only copies of e in the environment and inside the cells, we have to initialize the simulation of the computation of M by generating the initial the label l_0, and an arbitrary number of l_i', l_i'' for all $l_i \in H$. These symbols are generated by C_1 and C_2 with the following programs:

$$\begin{aligned} P_1 \supset\ & \{\langle e \to l_r' \rangle, \langle l_r' \longrightarrow e \rangle, \langle e \to l_r'' \rangle, \langle l_r'' \longrightarrow e \rangle \mid l_r \in H\} \cup \\ & \{\langle e \longrightarrow D' \rangle, \langle D' \to l_0 \rangle, \langle l_0 \longrightarrow D \rangle\}, \end{aligned}$$

$$P_2 \supset \{\langle e \to D' \rangle, \langle D' \to D' \rangle, \langle D' \longrightarrow l_1' \rangle, \langle l_1' \to D \rangle, \langle D \longrightarrow l_1'' \rangle\}.$$

With these programs, from the configuration $(e,e,e,e,e,e;\varepsilon)$, we obtain $(D,l_1'',e,e,e,e;l_0w)$ where the environment contains the label of the initial instruction, l_0, and w, a multiset of primed and double primed instruction labels.

To simulate the instruction $l_i:(ADD(r),l_j,l_k)$, cells C_1 and C_3 cooperate to add one copy of object a_r and object l_j or l_k to the environment.

P_1				P_3			
$i_1:$	$\langle D \longrightarrow a_{r,i}\rangle$	$i_6:$	$\langle K_k \to l_k\rangle$	$i_1:$	$\langle e \longrightarrow l_i\rangle$	$i_6:$	$\langle l_i' \to K_k\rangle$
$i_2:$	$\langle a_{r,i} \to a_r\rangle$	$i_7:$	$\langle l_j \longrightarrow D\rangle$	$i_2:$	$\langle l_i \to a_{r,i}\rangle$	$i_7:$	$\langle K_j \longrightarrow e\rangle$
$i_3:$	$\langle a_r \longrightarrow K_j\rangle$	$i_8:$	$\langle l_k \longrightarrow D\rangle$	$i_3:$	$\langle a_{r,i} \longrightarrow l_i'\rangle$	$i_8:$	$\langle K_k \longrightarrow e\rangle$
$i_4:$	$\langle a_r \longrightarrow K_k\rangle$			$i_4:$	$\langle a_{r,i} \to t\rangle$	$i_9:$	$\langle t \to t\rangle$
$i_5:$	$\langle K_j \to l_j\rangle$			$i_5:$	$\langle l_i' \to K_j\rangle$		

It is not difficult to follow how the interplay of these two cells produce the configuration $(D,l_1'',e,e,e,e;l_ja_rw')$ or $(D,l_1'',e,e,e,e;l_ka_rw')$ from a configuration $(D,l_1'',e,e,e,e;l_iw)$ where w,w' are multisets of l_i',l_i'' for $l_i\in H$ and $a_r,\ 1\le r\le m$. If there is no l_i' present in the environment when the program i_3 of cell C_3 should be used, then the programs i_4 and i_9 do not allow the halting of the computation.

For each subtract instruction $l_f:(SUB(r),l_g,l_n)$ there are the following programs in P_1, P_4, P_5 and in P_6:

P_1		P_4				P_5		P_6	
$f_1:$	$\langle D \leftrightarrow L_f\rangle$	$f_1:$	$\langle e \longrightarrow l_f\rangle$	$f_9:$	$\langle L_f \to t\rangle$	$f_1:$	$\langle e \leftrightarrow L_f'\rangle$	$f_1:$	$\langle e \leftrightarrow L_f''\rangle$
$f_2:$	$\langle L_f \to E_f\rangle$	$f_2:$	$\langle l_f \to L_f\rangle$	$f_{10}:$	$\langle L_f' \to t\rangle$	$f_2:$	$\langle L_f' \to l_f'\rangle$	$f_2:$	$\langle L_f'' \to l_f'\rangle$
$f_3:$	$\langle E_f \to F_f\rangle$	$f_3:$	$\langle L_f \leftrightarrow l_f'\rangle$	$f_{11}:$	$\langle t \to t\rangle$	$f_3:$	$\langle l_f' \leftrightarrow a_r\rangle$	$f_3:$	$\langle l_f' \longrightarrow \$_f\rangle$
$f_4:$	$\langle F_f \to \$_f\rangle$	$f_4:$	$\langle l_f' \to L_f'\rangle$			$f_4:$	$\langle l_f' \leftrightarrow \$_f\rangle$	$f_4:$	$\langle \$_f \to l_g\rangle$
$f_5:$	$\langle \$_f \leftrightarrow D\rangle$	$f_5:$	$\langle L_f' \longrightarrow l_f''\rangle$			$f_5:$	$\langle \$_f \to \bar{l}_n\rangle$	$f_5:$	$\langle l_g \leftrightarrow e\rangle$
		$f_6:$	$\langle l_f'' \to L_f'''\rangle$			$f_6:$	$\langle a_r \to e\rangle$	$f_6:$	$\langle l_f' \longrightarrow \bar{l}_n\rangle$
		$f_7:$	$\langle L_f''' \to L_f''\rangle$			$f_7:$	$\langle \bar{l}_n \leftrightarrow e\rangle$	$f_7:$	$\langle \bar{l}_n \to l_n\rangle$
		$f_8:$	$\langle L_f'' \longrightarrow e\rangle$					$f_8:$	$\langle l_n \longrightarrow e\rangle$

In the following table we show how a subtract instruction can be simulated by the programs above. Since C_2 and C_3 cannot apply any of their rules in any step of the following simulation, we omit them from the table. The multiset of objects in the environment is denoted by $[\ldots]$, and for now we assume that it always contains a sufficient amount of l_i',l_i'' objects for any $l_i\in H$.

First we consider the case when there is at least one object a_r in the environment, that is, if the

simulation starts in a configuration $(D, l_1'', e, e, e, e; l_f a_r[\ldots])$.

	configuration of Π					programs to be applied			
	C_1	C_4	C_5	C_6	Env	P_1	P_4	P_5	P_6
1.	D	e	e	e	$l_f a_r[\ldots]$	$-$	f_1	$-$	$-$
2.	D	l_f	e	e	$a_r[\ldots]$	$-$	f_2	$-$	$-$
3.	D	L_f	e	e	$a_r[\ldots]$	$-$	f_3	$-$	$-$
4.	D	l_f'	e	e	$L_f a_r[\ldots]$	f_1	f_4	$-$	$-$
5.	L_f	L_f'	e	e	$Da_r[\ldots]$	f_2	f_5	$-$	$-$
6.	E_f	l_f''	e	e	$L_f' Da_r[\ldots]$	f_3	f_6	f_1	$-$
7.	F_f	L_f'''	L_f'	e	$Da_r[\ldots]$	f_4	f_7	f_2	$-$
8.	$\$_f$	L_f''	l_f'	e	$Da_r[\ldots]$	f_5	f_8	f_3	$-$
9.	D	e	a_r	e	$\$_f L_f''[\ldots]$	$-$	$-$	f_6	f_1
10.	D	e	e	L_f''	$\$_f[\ldots]$	$-$	$-$	$-$	f_2
11.	D	e	e	l_f'	$\$_f[\ldots]$	$-$	$-$	$-$	f_3
12.	D	e	e	$\$_f$	$[\ldots]$	$-$	$-$	$-$	f_4
13.	D	e	e	l_g	$[\ldots]$	$-$	$-$	$-$	f_5
14.	D	e	e	e	$l_g[\ldots]$	$-$	g_1	$-$	$-$

In 13 steps, from a configuration $(D, l_1'', e, e, e, e; l_f a_r[\ldots])$ we obtain $(D, l_1'', e, e, e, e; l_g[\ldots])$ where l_g is the label of the instruction which should follow the successful decrease of the value of the nonempty register r, and the environment contains a multiset of objects l_i', l_i'' for $l_i \in H$.

Now we consider the case when register r, which is the register to be decremented, stores zero, that is, if the simulation starts in a configuration $(D, l_1'', e, e, e, e; l_f[\ldots])$ where the environment does not contain any object a_r.

	configuration of Π					programs to be applied			
	C_1	C_4	C_5	C_6	Env	P_1	P_4	P_5	P_6
1.	D	e	e	e	$l_f[\ldots]$	$-$	f_1	$-$	$-$
2.	D	l_f	e	e	$[\ldots]$	$-$	f_2	$-$	$-$
3.	D	L_f	e	e	$[\ldots]$	$-$	f_3	$-$	$-$
4.	D	l_f'	e	e	$L_f[\ldots]$	f_1	f_4	$-$	$-$
5.	L_f	L_f'	e	e	$D[\ldots]$	f_2	f_5	$-$	$-$
6.	E_f	l_f''	e	e	$L_f' D[\ldots]$	f_3	f_6	f_1	$-$
7.	F_f	L_f'''	L_f'	e	$D[\ldots]$	f_4	f_7	f_2	$-$
8.	$\$_f$	L_f''	l_f'	e	$D[\ldots]$	f_5	f_8	$-$	$-$
9.	D	e	l_f'	e	$\$_f L_f''[\ldots]$	$-$	$-$	f_4	f_1
10.	D	e	$\$_f$	L_f''	$[\ldots]$	$-$	$-$	f_5	f_2
11.	D	e	$\bar{l}_n$	l_f'	$[\ldots]$	$-$	$-$	f_7	$-$
12.	D	e	e	l_f'	$\bar{l}_n[\ldots]$	$-$	$-$	$-$	f_6
13.	D	e	e	$\bar{l}_n$	$[\ldots]$	$-$	$-$	$-$	f_7
14.	D	e	e	l_n	$[\ldots]$	$-$	$-$	$-$	f_8
15.	D	e	e	e	$l_n[\ldots]$	$-$	n_1	$-$	$-$

Similarly to the previous case, in 14 steps we obtain a configuration $(D, l_1'', e, e, e, e; l_n[\ldots])$ where l_n is the label of the instruction which should follow l_f if register r is empty, that is, if the decrease of its value is not possible.

Consider now what happens if there is an insufficient amount of objects l_i', l_i'' for $l_i \in H$ is present in the environment. Notice that such symbols are needed in step 3 and 5 by cell C_4. If there is no more

available (not enough of them were produced in the initial phase by C_1 and C_2), then the programs f_9, f_{10}, and f_{11} do not allow the halting of the computation.

From these considerations we can see that after the initialization phase, all instructions of the register machine M can be simulated by the P colony. If the label of the halt instruction, l_h is produced, the computation halts since there is no program for processing the object l_h. The reader can immediately see that Π computes the same set of vectors as M. □

4 P colonies with senders and consumers

Now we continue with the investigation of two object P colonies with insertion-deletion programs. It is not too difficult to see that if we allow a cell to contain both types of programs, then we can simulate the other types of programs in two steps, thus, it is more interesting to consider P colonies having cells which contain either insertion or deletion programs, but not both types at the same time. We call these systems P colonies with senders and consumers. A sender is a cell with only insertion programs, a consumer is a cell with only deletion programs.

Let us denote by $PCOL(i, j, \text{s-c})$ the class of sets of numbers generated by P colonies with senders and consumers having at most $i \geq 1$ cells with at most $j \geq 1$ program each.

Example 2. *(a) A P colony with one sender cell can generate the Parikh set of a regular language $L \subseteq T^*$. Let $G = (N, T, P, S)$ be a regular grammar such that $L(G) = L$.*

For generating the Parikh vectors of the words in L, we use, for each $S \to aB$ of P, the programs $\langle e, out; e \to eS\rangle$, $\langle e, out; S \to aB\rangle$ and then $\langle x, out; A \to aB\rangle, x \in T$ for every $A \to aB$ in P. Finally, for every rule of the form $A \to a$ we need $\langle x, out; A \to aF\rangle, x \in T$, $\langle a, out; F \to FF\rangle$, where $F \notin T \cup N$.

(b) A P colony with one consumer cell can "consume" the Parikh set of a regular language L. To see this, let $M = (Q, T, \delta, q_0, F)$ be a deterministic finite automaton such that $L(M) = L$.

We need the program $\langle e, in; ee \to q_0\rangle$, and to every transition $\delta(q_i, a) = q_j$ in M, we introduce $\langle a, in; xq_i \to q_j\rangle, x \in T \cup \{e\}$. If $q_j \in F$ in $\delta(q_i, a) = q_j$ we have to add the programs $\langle a, in; xq_i \to F\rangle, x \in T$, where $F \notin Q \cup T$.

Now we show that three cells, one sender and two consumers are sufficient to generate all recursively enumerable sets of integer vectors.

Theorem 3. $PCOL(3, *, s\text{-}c) = \mathbb{N}RE$.

Proof. Consider an m-register machine $M = (m, H, l_0, l_h, P)$, $m \geq 1$. We simulate M by representing the content of the register i by the number of copies of a specific object a_i in the environment. We construct the P colony $\Pi = (V, e, F, C_1, C_2, C_3)$ with:

$$\begin{aligned} V &= \{e, l, l', l'', l''', l^{iv}, l^v, \bar{l}, \bar{\bar{l}} \mid l \in H\} \cup \{a_i \mid 1 \leq i \leq m\} \cup \{K, T_1, T_2, T_3, T_4, T_5\}, \\ F &= \{a_i \mid \text{register } i \text{ is an output register}\}, \text{ and} \\ C_i &= (ee, P_i) \text{ for } 1 \leq i \leq 3. \end{aligned}$$

The P colony Π starts its computation in the initial configuration $(ee, ee, ee; \varepsilon)$. We initialize the computation by generating the initial label l_0 with a program from P_1, $\langle e, out; e \to l_0 l_0\rangle \in P_1$ obtaining $(l_0 l_0, ee, ee; \varepsilon)$.

The simulation of an instruction with label l_i starts from a configuration $(l_i l_i, ee, ee; w)$ where $w \in V^*$, the multiset of objects in the environment, represents the counter contents of M.

To simulate an *ADD* instruction, we use the programs of P_1 and P_3. For each $l_i, l_j, l_k \in H$ with l_i being

the label of an instruction $l_i : (ADD(r), l_j, l_k)$, we have the following programs:

P_1		P_3	
$i_1:$	$\langle l_i, out; l_i \to a_r l_j \rangle$	$i_1:$	$\langle l_i, in; ee \to T_1 \rangle$
$i_2:$	$\langle l_i, out; l_i \to a_r l_k \rangle$	$i_2:$	$\langle e, in; l_i T_1 \to e \rangle$
$i_3:$	$\langle a_r, out; l_j \to l_j l_j \rangle$	$i_3:$	$\langle l_i, in; \bar{\bar{l}}_i T_5 \to T_1 \rangle$
$i_4:$	$\langle a_r, out; l_k \to l_k l_k \rangle$		

Using these programs, we obtain a sequence of configurations

$$(l_i l_i, ee, ee; w) \Rightarrow (a_r l, ee, ee; l_i w) \Rightarrow (ll, ee, l_i T_1; a_r w)$$

where l is the label of the next instruction, that is, we either have $(l_j l_j, ee, l_i T_1; a_r w)$, or the configuration $(l_k l_k, ee, l_i T_1; a_r w)$. The contents of cell C_3, $l_i T_1$, will change in the next step to ee independently of the several ways of the continuation of the computation, as we shall see later.

The program labeled with i_3 is used if the instruction simulated before l_i was a SUB instruction (see below). In this case, the configuration in which the simulation of l_i starts is $(l_i l_i, ee, \bar{l}_i T_4; \bar{\bar{l}}_i w)$ and we need the steps $(l_i l_i, ee, \bar{l}_i T_4; \bar{\bar{l}}_i w) \Rightarrow (a_r l, ee, \bar{\bar{l}}_i T_5; l_i w) \Rightarrow (ll, ee, l_i T_1; a_r w)$ and program i_3 to obtain the same configuration as before.

Now we show how to simulate a SUB instruction. For each $l_j, l_k, l_l \in H$ with l_j being the label of an instruction $l_j : (SUB(r), l_k, l_l)$, and for all labels $l_s \in H$, we have the following programs.

P_1	P_2		P_3	
$j_1 : \langle l_j, out; l_j \to l'_j l'_j \rangle$	$j_1:$	$\langle l_j, in; ee \to e \rangle$	$j_1:$	$\langle l'_j, in; ee \to T_1 \rangle$
$j_2 : \langle l'_j, out; l'_j \to l''_j l''_j \rangle$	$j_2:$	$\langle a_r, in; e l_j \to e \rangle$	$j_2:$	$\langle e, in; l'_j T_1 \to T_2 \rangle$
$j_3 : \langle l''_j, out; l''_j \to l'''_j l^{iv}_j \rangle$	$j_3:$	$\langle l''_j, in; e l_j \to e \rangle$	$j_3:$	$\langle l''_j, in; eT_2 \to T_3 \rangle$
$j_4 : \langle l'''_j, out; l^{iv}_j \to \bar{l}_k \bar{l}_k \rangle$	$j_4:$	$\langle l'''_j, in; a_r e \to e \rangle$	$j_{4,s}:$	$\langle \bar{l}_s, in; l''_j T_3 \to T_4 \rangle$
$j_5 : \langle l^{iv}_j, out; l'''_j \to \bar{l}_l \bar{l}_l \rangle$	$j_5:$	$\langle e, in; l'''_j e \to e \rangle$	$j_{5,s}:$	$\langle \bar{l}_s, in; eT_2 \to T_4 \rangle$
$j_6 : \langle \bar{l}_k, out; \bar{l}_k \to \bar{\bar{l}}_k \bar{\bar{l}}_k \rangle$	$j_6:$	$\langle l^{iv}_j, in; a_r e \to K \rangle$	$j_{6,s}:$	$\langle \bar{\bar{l}}_s, in; \bar{l}_s T_4 \to T_5 \rangle$
$j_7 : \langle \bar{\bar{l}}_k, out; \bar{\bar{l}}_k \to l_k l_k \rangle$	$j_7:$	$\langle e, in; l^{iv}_j K \to K \rangle$	$j_{7,s}:$	$\langle e, in; \bar{\bar{l}}_s T_5 \to e \rangle$
$j_8 : \langle \bar{l}_l, out; \bar{l}_l \to \bar{\bar{l}}_l \bar{\bar{l}}_l \rangle$	$j_8:$	$\langle e, in; eK \to K \rangle$		
$j_9 : \langle \bar{\bar{l}}_l, out; \bar{\bar{l}}_k \to l_l l_l \rangle$	$j_9:$	$\langle l'''_j, in; l''_j e \to K \rangle$		
	$j_{10}:$	$\langle e, in; l'''_j K \to K \rangle$		
	$j_{11}:$	$\langle l^{iv}_j, in; l''_j e \to e \rangle$		
	$j_{12}:$	$\langle e, in; l^{iv}_j e \to e \rangle$		

In the following tables we show how the programs above simulate the execution of the instruction $l_j : (SUB(r), l_k, l_l)$. To save space, we use the sign "/" to separate the different possible multisets which might appear in the same row of the table.

First we consider the case when register r is not empty, that is, when there is at least one object a_r present in the environment. We see that starting with a configuration where C_1 contains the objects $l_j l_j$ and the environment contains a_r, in six steps we obtain a configuration where the object a_r is removed from the environment, and C_1 either contains the label of the next instruction l_k, or because of the presence of program j_8, in P_2, the computation will never be able to halt.

	configuration of Π				programs to be applied		
	C_1	C_2	C_3	Env	P_1	P_2	P_3
1.	$l_j l_j$	ee	?	$a_r w'$	j_1	$-$	?
2.	$l'_j l'_j$	ee	?	$l_j a_r w''$	j_2	j_1	?
3.	$l''_j l''_j$	$l_j e$	ee	$l'_j a_r w$	j_3	j_2	j_1
4.	$l'''_j l^{iv}_j$	$a_r e$	$l'_j T_1$	$l''_j w$	j_4/j_5	$-$	j_2
5.	$\bar{l}_k \bar{l}_k / \bar{l}_l \bar{l}_l$	$a_r e$	eT_2	$(l'''_j/l^{iv}_j) l''_j w$	j_6/j_8	j_4/j_6	j_3
6.	$\bar{\bar{l}}_k \bar{\bar{l}}_k / \bar{\bar{l}}_l \bar{\bar{l}}_l$	$l'''_j e / l^{iv}_j K$	$l''_j T_3$	$(\bar{l}_k/\bar{l}_l) w$	j_7/j_9	j_5/j_7	$j_{4,k}/j_{4,l}$
7.	$l_k l_k / l_l l_l$	ee/eK	$(\bar{l}_k/\bar{l}_l) T_4$	$(\bar{\bar{l}}_k/\bar{\bar{l}}_l) w$	k_1/l_1	$-/j_8$	$j_{6,k}/j_{6,l}$
8.	$l'_k l'_k / l'_l l'_l$	ee/eK	$(\bar{\bar{l}}_k/\bar{\bar{l}}_l) T_5$	$(l_k/l_l) w$	k_2/l_2	k_1/j_8	$j_{7,k}/j_{7,l}$
9.	$l''_k l''_k / l''_l l''_l$	$(l_k/l_l)e/eK$	ee	$(l'_k/l'_l) w$	k_3/l_3	k_2/j_8	j_1

Now we show the simulation of the $l_j : (SUB(r), l_k, l_l)$ instruction when there is no object a_r is present in the environment, that is, when register r is empty. In this case, similarly to the previous one, we either get the objects $l_k l_k$ in the cell C_1, or the computation will not be able to halt.

	configuration of Π				rules to be applied		
	C_1	C_2	C_3	Env	P_1	P_2	P_3
1.	$l_j l_j$	ee	?	w	j_1	$-$	?
2.	$l'_j l'_j$	ee	?	$l_j w$	j_2	j_1	?
3.	$l''_j l''_j$	$l_j e$	ee	$l'_j w$	j_3	$-$	j_1
4.	$l'''_j l^{iv}_j$	$l_j e$	$l'_j T_1$	$l''_j w$	j_4/j_5	j_3	j_2
5.	$\bar{l}_k \bar{l}_k / \bar{l}_l \bar{l}_l$	$l''_j e$	eT_2	$(l'''_j/l^{iv}_j) w$	j_6/j_8	j_9/j_{11}	$-$
6.	$\bar{\bar{l}}_k \bar{\bar{l}}_k / \bar{\bar{l}}_l \bar{\bar{l}}_l$	$l'''_j K / l^{iv}_j e$	eT_2	$(\bar{l}_k/\bar{l}_l) w$	j_7/j_9	j_{10}/j_{12}	$j_{5,k}/j_{5,l}$
7.	$l_k l_k / l_l l_l$	eK/ee	$(\bar{l}_k/\bar{l}_l) T_4$	$(\bar{\bar{l}}_k/\bar{\bar{l}}_l) w$	k_1/l_1	$j_8/-$	$j_{6,k}/j_{6,l}$
8.	$l'_k l'_k / l'_l l'_l$	eK/ee	$(\bar{\bar{l}}_k/\bar{\bar{l}}_l) T_5$	$(l_k/l_l) w$	k_2/l_2	j_8/k_1	$j_{7,k}/j_{7,l}$
9.	$l''_k l''_k / l''_l l''_l$	$eK/(l_k/l_l)e$	ee	$(l'_k/l'_l) w$	k_3/l_3	j_8/k_2	j_1

The rules to be applied and the objects contained by the cell C_3 in row 1. and row 2. of the tables above depend on the instruction l_i which was simulated before l_j. If l_i is an ADD instruction, then we have $l_i T_1$ in the first row, and applying the program i_2 from P_3 we get ee in the second row, where no program is applied until the next step. Also, $w = w' = w''$ in this case.

If l_i is a SUB instruction, then (as we can also see from row 7. and row 8.) the contents of the cell C_3 is $\bar{l}_j T_4$ and $\bar{\bar{l}}_j T_5$ in the first two rows where the programs $i_{6,j}$ and $i_{7,j}$ are applied. In this case $w'' = \bar{\bar{l}}_j w$, and $w' = w$.

As we have seen above, the P colony successfully simulates each instruction of M and since there is no program to process l_h, the label of the halt instruction, it also halts when the computation of M is finished. It is also easy to see that M and Π compute the same set of vectors of non-negative integers. □

5 Conclusion

We have examined extremely simplified variants of P colonies: P colonies of capacity one with no checking rules, and P colonies with capacity two, but only with senders and consumers. We have shown that even these very simple variants are able to simulate arbitrary register machines, that is, to compute all Turing computable sets of vectors.

Bibliography

[1] L. Cienciala, L. Ciencialová, A. Kelemenová. On the number of agents in P colonies. In: *Membrane Computing. 8th International Workshop, WMC 2007. Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers.* Edited by G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa. Volume 4860 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin-Heidelberg, 2007, 193-208.

[2] E. Csuhaj-Varjú, J. Dassow, J. Kelemen, Gh. Păun. *Grammar Systems – A Grammatical Approach to Distribution and Cooperation*. Gordon and Breach, London, 1994.

[3] E. Csuhaj-Varjú, J. Kelemen, A. Kelemenová, Gh. Păun, Gy. Vaszil. Computing with cells in environment: P colonies. *Journal of Multi-Valued Logic and Soft Computing* 12:201-215, 2006.

[4] J. Kelemen, A. Kelemenová. A grammar-theoretic treatment of multi-agent systems. *Cybernetics and Systems* 23:621-633, 1992.

[5] J. Kelemen, A. Kelemenová, Gh. Păun. Preview of P colonies: A biochemically inspired computing model. In: *Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX)*. Edited by M. Bedau et al. Boston Mass., 2004, 82-86.

[6] M. Minsky. *Computation – Finite and Infinite Machines.* Prentice Hall, Englewood Cliffs, NJ, 1967.

[7] Gh. Păun. *Membrane Computing – An Introduction.* Springer-Verlag, Berlin, 2002.

Lucie Ciencialová has received her PhD. in 2008 and she works as an assistant professor at the Institute of Computer Science, Silesian University in Opava. Her interests include theoretical informatics and natural computing.

Erzsébet Csuhaj-Varjú D.Sc, dr. Habil., is head of the Theoretical Computer Science Research Group at the Computer and Automaton Research Institute of the Hungarian Academy of Sciences. She has also been affiliated with the Department of Algorithms and Applications of the Eötvös Loránd University, Budapest, Hungary, as science advisor. Her main research interests are formal languages, distributed systems, and nature-motivated computing. In these areas she has more than 150 papers, a monograph, and eleven edited volumes published in international publication forums.

Alica Kelemenová is an associated professor at the Institute of Computer Science, Silesian University in Opava, Czech republic and at the Department of Computer Science Catholic University in Ružomberok, Slovakia. Her main research interest is theoretical computer science, especially formal language theory and biologically motivated generative devices like L systems and P systems.

György Vaszil, PhD. is a senior research fellow at the Theoretical Computer Science Research Group of the Computer and Automation Research Institute of the Hungarian Academy of Sciences. His research interests are formal language and automata theory, nature motivated computational models.

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 234-243

P–Lingua 2.0: A software framework for cell–like P systems

Manuel García-Quismondo, Rosa Gutiérrez-Escudero, Miguel A Martínez-del-Amor
Enrique Orejuela-Pinedo, I. Pérez-Hurtado

Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: mangarfer2@alum.us.es, {rgutierrez, mdelamor, orejuela, perezh}@us.es

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: P-Lingua is a programming language for membrane computing. It was first presented in Edinburgh, during the Ninth Workshop on Membrane Computing (WMC9). In this paper, the models, simulators and formats included in P-Lingua in version 2.0 are explained. We focus on the stochastic model, associated simulators and updated features. Finally, we present one of the first applications based on P-Lingua: a tool for describing and simulating ecosystems.
Keywords: Programming languages, software development, P systems, Membrane Computing, P-Lingua

1 Introduction

Membrane computing (or cellular computing) is a branch of Natural Computing that was introduced by Gh. Păun [10]. The main idea is to consider biochemical processes taking place inside living cells from a computational point of view. The initial definition of this computing paradigm is very flexible and many different models have been defined.

Each model displays characteristic semantic constraints that determine the way in which rules are applied. Hence, the need for software simulators capable of taking into account different scenarios when simulating P system computations comes to the fore. An initial approach could be defining inputs for each simulator specifically. Nevertheless, this approach involves defining new input formats for each simulator, so designing simulators would take a great effort. A second approach could be standardizing the simulator input, so all simulators need to process inputs specified in the same format. These two approaches raise up a trade-off: On the one hand, specific simulator inputs could be defined in a more straightforward way, as the used format is closer to the P system features to simulate. On the other hand, although the latter approach involves analyzing different P systems and models to develop a standard format, there is no need to develop a new simulator every time a new P system should be simulated, as it is possible to specify it in the standard input format. Moreover, researches would not have to devise a new input format every time they specify a P system and would not need to change the way to specify P systems which need to be simulated every time they move on to another model, as they would keep on using the standard input format.

This second approach is the one considered in P-Lingua project, a programming language whose first version, presented in [3], is able to define P systems within the active membrane P system model with division rules. The authors also provide software tools for compilation, simulation and debug tasks. From now on, we will call P-Lingua 1.0 this version of the language and its associated tools.

As P-Lingua is intended to become a standard for P systems definition, it should also consider other models. In this paper, we present P-Lingua 2.0 as a framework to define cell-like P system models, including several algorithms to simulate P system computations for the supported models (from now on,

simulators), as well as different formats to represent P systems with associated parsers to translate from each other.

This paper is structured as follows. In Section 2 the supported models at this stage are enumerated. The next section introduces some algorithms used to simulate P systems, focusing on the stochastic and the probabilistic P system models. In Section 4 different file formats to representate cell-like P systems are presented, for example, P-Lingua 2.0 programming language. Model definitions, simulators and parsers have been encoded in a JAVA library, pLinguaCore ©, presented in Section 6, this library is free software and it can be easily expanded. Command-line tools to compile files and simulate P systems have been slightly modified - in Section 5 these changes are presented. The next section introduces one of the first applications of P-Lingua, a software tool to describe and simulate ecosystems. Finally, some conclusions and future work are enumerated in Section 8.

2 Supported P system models

The supported models developed so far are enumerated below, but a standard mechanism for defining new cell-like models has been included on the P-Lingua 2.0 framework. Each model displays characteristic semantic constraints entailing the rules applied, such as the number of objects specified on the left-hand side, membrane creation, polarization, and so on. It is possible to define additional models by including the corresponding semantic constraints within the plinguaCore JAVA library. This mechanism has been used on all the existent models.

The supported P system models in P-Lingua 2.0 are Transition P Systems, Symport/Antiport P Systems, P Systems with active membranes, with membrane division and membrane creation rules, Probabilistic P Systems and Stochastic P Systems. More details on those models can be found in [13], except for Stochastic P Systems, which are described in [10].

3 Simulators

In P-Lingua 1.0, only one simulator was supported, since there was only one P system model definition. However, as new models have been included, new simulators have been developed, providing at least one simulator for each supported model.

All simulators in P-Lingua 2.0 can step backwards (as well as the simulator in P-Lingua 1.0), but this option should be set before the simulation starts.

P-Lingua 2.0 also takes into account the existence of different simulation algorithms for the same model and provides a means for selecting a simulator among the ones which are suitable to simulate the P system, by checking its model. So far, only the stochastic P system model provides several simulation algorithms to choose, but the plugin-oriented architecture of the pLinguaCore JAVA library allows easily to encode new simulators.

3.1 Simulators for Stochastic P Systems

In the original definitions P systems evolve in a non-deterministic and maximally parallel manner (that is, all the objects in every membrane that can evolve by a rule must do it [10]). When trying to simulate biological phenomena, like living cells, the classical non-deterministic and maximally parallel approach is not valid anymore. First, biochemical reactions, which are modelled by rules, occur at a specific rate (determined by the propensity of the rule), therefore they cannot be selected in an arbitrary and non-deterministic way. Second, in the classical approach all time steps are equal and this does not represent the time evolution of a real cell system.

The strategies to replace the original approach are based on Gillespie's Theory of Stochastic Kinetics

[6]. A constant c is associated to each rule, which provides P systems with a stochastic extension. The constant c depends on the physical properties of the molecules involved in the reaction modeled by the rule and other physical parameters of the system. Besides, it represents the probability per time unit at which the reaction takes place. Also, it is used to calculate the propensity of each rule which determines the probability and time needed to apply the rule.

Two different algorithms based on the principles stated above have been implemented and integrated in pLinguaCore.

Multicompartimental Gillespie Algorithm

The Gillespie [6] algorithm or SSA (Stochastic Simulation Algorithm) was developed for a single, well-mixed and fixed volume/compartment. P systems generally contain several compartments or membranes. For that reason, an adaptation of this algorithm was presented in [10] and it can be applied in the different regions defined by the compartmentalised structure of a P system model. The next rule to be applied in each compartment and the waiting time for this application is computed using a *local* Gillespie algorithm. The Multicompartimental Gillespie Algorithm can be broadly summarized as follows:

Repeat until a prefixed simulation time is reached:

1. Calculate for each membrane $i, 1 \leq i \leq m$, and for each rule $r_j \in R_{l_i}$ the propensity, a_j, by multiplying the stochastic constant $c_j^{l_i}$ associated to r_j by the number of distinct possible combinations of the objects and substrings present of the left-side of the rule with respect to the current contents of membranes involved in the rule.

2. Compute the sum of all propensities

$$a_0 = \sum_{i=1}^{m} \sum_{r_j \in R_{l_i}} a_j$$

3. Generate two random numbers r_1 and r_2 from the uniform distribution in the unit interval and select τ_i and j_i according to

$$\tau_i = \frac{1}{a_0} \ln(\frac{1}{r_1})$$

$$j_i = \text{the smallest integer satisfying } \sum_{j=1}^{j_i} a_j > r_2 a_0$$

   In this way, we choose τ_i according to an exponential distribution with parameter a_0.

4. The next rule to be applied is r_{j_i} and the waiting time for this rule is τ_i. As a result of the application of this rule, the state of one or two compartments may be changed and has to be updated.

The Multicompartimental Next Reaction Method

The Gillespie Algorithm is an exact numerical simulation method appropriate for systems with a small number of reactions, since it takes a time proportional to the number of reactions (i.e., the number of rules). An exact algorithm which is also efficient is presented in [5], the Next Reaction Method. It uses only a single random number per simulation event (instead of two) and takes a time proportional to the logarithm of the number of reactions. We have adapted this algorithm to make it compartimental.

The idea of this method is to be extremely sensitive in recalculating a_j and t_i, trying to recalculate them only if they change. In order to do that, a data structure called *dependency graph* [5] is introduced.

Let $r : u[v]_l \xrightarrow{c} u'[v']_l$ be a given rule with propensity a_r and let the parent membrane of l be labelled with l'. We define the following sets:

- DependsOn$(a_r) = \{(b,t) \mid b$ is an object or string whose quantity affect the value.

  a_r, $t = l$ if $b \in v$ and $t = l'$ if $b \in u\}$.

  Generally, DependsOn$(a_r) = \{(b,l) \mid b \in v\} \cup \{(b,l') \mid b \in u\}$

- Affects$(r) = \{(b,t) \mid b$ is an object or string whose quantity is changed when the rule.

  r is excuted, $t = l$ if $b \in v \vee b \in v'$ and $t = l'$ if $b \in u \vee b \in u'\}$.

  Generally, Affects$(r) = \{(b,l) \mid b \in v \vee b \in v'\} \cup \{(b,l') : b \in u \vee b \in u'\}$.

Definition 1. Given a set of rules $R = R_{l_1} \cup \cdots \cup R_{l_m}$, the dependency graph is the directed graph $G = (V,E)$, with vertex set $V = R$ and edge set $E = \{(v_i, v_j) \mid \text{Affects}(v_i) \cap \text{DependsOn}(a_{v_j}) \neq \emptyset\}$

In this way, if there exists an edge $(v_i, v_j) \in E$ and v_i is executed, as some objects affected by this execution are involved in the calculation of a_{v_j}, this propensity would have to be recalculated. The dependency graph depends only on the rules of the system and is static, so it is built only once.

The times τ_i, that represent the waiting time for each rule to be applied, are stored in an *indexed priority queue*. This data structure, discussed in detail in [5], has nice properties: finding the minimum element takes constant time, the number of nodes is the number of rules $|R|$, because of the indexing scheme it is possible to find any arbitrary reaction in constant time and finally, the operation of updating a node (only when τ_i is changed, which we can detect using to the dependency graph) takes $\log|R|$ operations.

The Multicompartimental Next Reaction Method can be broadly summarized as follows:

1. Build the dependency graph, calculate the propensity a_r for every rule $r \in R$ and generate τ_i for every rule according to an exponential distribution with parameter a_r. All the values τ_r are stored in a priority queue. Set $t \leftarrow 0$ (this is the global time of the system).

2. Get the minimum τ_μ from the priority queue, $t \leftarrow t + \tau_\mu$. Execute the rule r_μ (this is the next rule scheduled to be executed, because its waiting time is least).

3. For each edge (μ, α) in the dependency graph recalculate and update the propensity a_α and

   - if $\alpha \neq \mu$, set
   $$\tau_\alpha \leftarrow \frac{a_{\alpha,old}(\tau_\alpha - \tau_\mu)}{a_{\alpha,new}} + \tau_\mu$$
   - if $\alpha = \mu$, generate a random number r_1, according to an exponential distribution with parameter a_μ and set $\tau_\mu \leftarrow \tau_\mu + r_1$

   Update the node in the indexed priority queue that holds τ_α.

4. Go to 2 and repeat until a prefixed simulation time is reached.

Both Multicompartimental Gillespie Algorithm and Multicompartimental Next Reaction Method are the core of the Direct Stochastic Simulator and Efficient Stochatic Simulator, respectively. One of them, which can be chosen in runtime, will be executed when compiling and simulating a P-Lingua file that starts with `@model<stochastic>`.

3.2 A Simulator for Probabilistic P Systems

Next, we describe how the simulator for probabilistic P systems implements the applicability of the rules to a given configuration.

(a) Rules are classified into sets so that all the rules belonging to the same set have the same left–hand side.

(b) Let $\{r_1, \ldots, r_t\}$ be one of the sets of rules. Let us suppose that the common left-hand side is $u\ [v]_i^\alpha$ and their respective probabilistic constants are $c_{r_1}, \ldots, c_{r_t}$. In order to determine how these rules are applied to a given configuration, we proceed as follows:

- One computes the greatest number N so that u^N appears in the parent membrane of i and v^N appears in membrane i.
- N random numbers x such that $0 \leq x < 1$ are generated.
- For each k $(1 \leq k \leq t)$ let n_k be the amount of numbers generated belonging to interval $[\sum_{j=0}^{k-1} c_{r_j}\ ,\ \sum_{j=0}^{k} c_{r_j})$ (assuming that $c_{r_0} = 0$).
- For each k $(1 \leq k \leq t)$, rule r_k is applied n_k times.

4 File formats to define P systems

Together with models and simulators, new formats have been included in P-Lingua 2.0. P-Lingua 1.0 provided a programming language to define P systems and an XML file format [3]. Both have been upgraded to allow representations of P systems which have a cell-like structure. It also supports backwards compatibility, so any file which defines a P system by using P-Lingua 1.0 is also recognized by P-Lingua 2.0 tools. A detailed description of the syntax of P-Lingua programming language, including the new extensions added in order to support the new models, can be found in [4].

A new format has been included as well, the binary format, whose purpose is to use less disk space than the XML format.

At this point, the concepts *input format* and *output format* should be introduced. An input format is a file format which, if a P system is specified in a file by following that format, the P system specified can be processed by the pLinguaCore JAVA library. An output format is a file format which, if a P system is specified in a file by following that format, that file can be generated by the library. These concepts are similar to the source code and object code concepts [3].

For P-Lingua 2.0 framework, P-Lingua programming language is an input format, the binary format is an output format and, eventually, XML is both an input and an output format. This means that P-Lingua programs can be processed by pLinguaCore, binary files can be generated by pLinguaCore and XML files can be both processed and generated by the library.

5 Command-line tools

P-Lingua 1.0 provided command-line tools for simulating P systems and compiling files which specify P systems [3]. In P-Lingua 2.0, the command-line tool general syntax has changed but, as it provides backwards compatibility, all valid actions in P-Lingua 1.0 are still valid in P-Lingua 2.0, as well.

5.1 The compilation command-line tool

The command-line tool general syntax for compiling input files is defined as follows:

```
plingua [-input_format] input_file [-output_format]
output_file [-v verbosity_level] [-h]
```

The command header `plingua` requests the system to compile the P system specified on a file to a file specified on another, whereas the file `input_file` contains the programme that we want to be compiled, and `output_file` is the name of the file that is generated [3]. Optional arguments are in square brackets:

- The option `-input_format` defines the format followed by `input_file`, which should be an input format.
- At this stage, valid input formats are `P-Lingua` and `XML`.
- If no input format is set, the P-Lingua format is assumed.
- The option `-output_format` defines the format followed by `output_file`, which should be an output format.
- At this stage, valid output formats are `XML` and `bin`.
- If no input format is set, the XML format is assumed by default.
- The option `-v` verbosity level is a number between 0 and 5 indicating the level of detail of the messages shown during the compilation process [3].
- The option `-h` displays some help information [3].

5.2 The simulation command-line tool

The simulations are launched from the command line as follows:

```
plingua_sim input_file -o output_file [-v verbosity level] [-h] [-to timeout]
[-st steps] [-mode simulatorID] [-a] [-b]
```

The command header `plingua_sim` requests the system to simulate the P system specified on a file, whereas `input_xml` is an XML document where a P system is formatted on, and output file is the name of the file where the report about the simulated computation will be saved [3]. Optional arguments are in brackets:

- The option `-v` verbosity level is a number between 0 and 5 indicating the level of detail of the messages shown during the compilation process [3]. If no value is specified, it is 3 by default.
- The option `-h` displays some help information [3].
- The option `-to` sets a timeout for the simulation defined in timeout (in milliseconds), so when the time out has elapsed the simulation is halted. If the simulation has reached a halting configuration before the time out has elapsed this option has no effect.
- The option `-st` sets a maximum number of steps the simulation can take (defined in steps), so when the time out has elapsed the simulation comes to a halt. If the simulation has reached a halting configuration or the time out has elapsed (in case the option `-to` is set) before the specified number of steps have been taken this option has no effect.

- The option `-mode` sets the specific simulator to simulate the P system (defined in simulatorID). This option reports an error in case the simulator defined by simulatorID is not a valid simulator for the P system model.
- The option `-a` defines if the simulation can take alternative steps. This option reports an error if the simulator does not support alternative steps.
- The option `-b` defines if the simulation can step backwards. As every simulator supports stepping backwards, this option does not report errors.

6 The pLinguaCore JAVA library

pLinguaCore © is a JAVA library which performs all functions supported by P-Lingua 2.0, that is, models definition, simulators and formats. This library reports the rules and membrane structure read from a file where a P system is defined, detects errors in the file, reports them. If the P system is defined in P-Lingua programming language, it locates the error in the file. This library performs simulations by using the simulators implemented as well as taking into account all options defined. It reports the simulation process, by displaying the current configuration as text and reporting the elapsed time. Eventually, this library translates files that define a P system between formats, for instance, from P-Lingua language format to binary format. This library is free software published under LGPL license [12], so everyone who is interested can upgrade, change and distribute it respecting the license restrictions.

7 A tool for simulating ecosystems based on P-Lingua

The Bearded Vulture (Gypaetus barbatus) is an endangered species in Europe that feeds almost exclusively on bone remains of wild and domestic ungulates. In [1], it is presented a first model of an ecosystem related to the Bearded Vulture in the Pyrenees (NE Spain), by using probabilistic P systems where the inherent stochasticity and uncertainty in ecosystems are captured by using probabilistic strategies. In order to validate experimentally the designed P system, the authors have developed a simulator that allows them to analyze the evolution of the ecosystem under different initial conditions. That software application is focused on a particular P system, specifically, the initial model of the ecosystem presented in [1]. With the aim of improving the model, the authors are adding ingredients to it, as new species and more complex behaviour for the animals. The improved model, together with results of virtual experiments made with this software application, is exhaustively described in [2].

A new GPL [11] licensed JAVA application with a friendly user-interface sitting on the pLinguaCore JAVA library has been developed. This application provides a flexible way to check, validate and improve computational models of ecosystem based on P systems instead of designing new software tools each time new ingredients are added to the models. Furthermore, it is possible to change the initial parameters of the modelled ecosystem in order to make the virtual experiments suggested by experts. These experiments will provide results that can be interpreted in terms of hypotheses. Finally, some of these hyphoteses will be selected by the experts in order to be checked in real experiments.

8 Conclusions and future work

Creating a programming language to specify P systems is an important task in order to facilitate the development of software applications for membrane computing.

In [3] P-Lingua was presented as a programming language to define active membrane P systems with division rules. The present paper extends that language to other models: transition P systems,

symport/antiport P systems, active membranes P systems with division or creation rules, probabilistic P systems and stochastic P systems.

We have developed a JAVA library which recognizes the models, implements several simulators for each model and defines different formats to codify P systems, like the P-Lingua one or a new binary format. This library can be expanded to define new models, simulators and formats.

It is possible to select different algorithms to simulate a P system, for example, there are two different algorithms for stochastic P systems. The library can be used inside other software applications, in this sense, we present a tool for virtual experimentation of ecosystems.

An internet website [14], still under construction, will be available to download the applications, libraries, source-code and technical reports, as well as provide information about the progress of the P-Lingua project. In addition, this site aims to be a meeting point for users and developers through the use of web-tools as forums.

The syntax of the P-Lingua programming language is sufficiently standard for specifying different models of cell-like P systems. However, a new version of the language is necessary in order to specify tissue-like P systems but this will be the aim of a future work.

Although P-Lingua 2.0 provides a way to simulate and compile P-systems, command-line tools are usually not user-friendly. It means it is not easy and intuitive for people to use them. For this purpose, P-Lingua 1.0 provided an Integrated Development Environment (IDE) [3], which eased the way people could use P-Lingua 1.0. For P-Lingua 2.0, a new IDE, called pLinguaPlugin, is being developed. Such an application is integrated into the Eclipse platform [13], so it makes the most of Eclipse's capabilities to provide a framework for translating, developing and testing P systems. It aims to be user-friendly and useful for P system researchers.

8.1 Acknowledgement

The authors acknowledge the valuable assistance given by Mario J. Perez-Jimenez whose vast experience and human quality was essential for us in taking our first steps in scientific research. The authors also acknowledge the support of the project TIN2006–13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

Bibliography

[1] M. Cardona, M.A. Colomer, M.J. Pérez–Jiménez, D. Sanuy, A. Margalida. Modeling Ecosystems Using P Systems: The Bearded Vulture, a Case Study. *Lecture Notes in Computer Science*, 5391, 137–156, 2009

[2] M. Cardona, M.A. Colomer, A. Margalida, I. Pérez–Hurtado, M.J. Pérez–Jiménez, D. Sanuy. P System Based Model of an Ecosystem of the Scavenger Birds, *Proceedings of the 7th Brainstorming Week on Membrane Computing*, Vol. I, 65–80, *in press*.

[3] D. Díaz–Pernil, I. Pérez–Hurtado, M.J. Pérez–Jiménez, A. Riscos–Núñez. A P-Lingua programming environment for Membrane Computing, *Proceedings of the 9th Workshop on Membrane Computing*, 155–172, 2008.

[4] M. García–Quismondo, R. Gutiérrez–Escudero, I. Pérez–Hurtado, M.J. Pérez–Jiménez. P-Lingua 2.0: New Features and First Applications, *Proceedings of the 7th Brainstorming Week on Membrane Computing*, Vol. I, 141–168, *in press*.

[5] M.A. Gibson and J. Bruck. Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels, *J. Phys. Chem.*, 104, 1876–1889, 2000.

[6] D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions, *J. Phys. Chem.*, 81, 2340–2361, 1977.

[7] Gh. Păun. Membrane computing. An introduction, Springer–Verlag, Berlin, 2002.

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

[9] Gh. Păun. P systems with active membranes. *Journal of Automata, Languages and Combinatorics*, 6, 1, 75–90, 2001.

[10] F.J. Romero–Campero. P Systems, a Computational Modelling Framework for Systems Biology, Doctoral Thesis, University of Seville, Department of Computer Science and Artificial Intelligence, 2008.

[11] The GNU General Public License: `http://www.gnu.org/copyleft/gpl.html`

[12] The GNU Lesser General Public License: `http://www.gnu.org/copyleft/lgpl.html`

[13] The Eclipse Project: `http://www.eclipse.org`

[14] The P-Lingua website: `http://www.p-lingua.org`

Manuel García-Quismondo Fernández was born in June 11, 1985. He got his degree in *Ingeniería Técnica en Informática de Sistemas* in the University of Sevilla in June 2007. Currently, he is about to get another degree, this time in *Ingeniería en Informática*, at the same university. Since September 2008, he has been a granted student at the Department of Computer Science and Artificial Intelligence. He has developed a software application called *pLinguaPlugin* and co-developed another one called *pLinguaCore*, directed by Agustín Riscos-Nuñez and Ignacio Pérez-Hurtado.

Rosa Gutiérrez-Escudero was born in August 16, 1984. She received her degree in Computer Science in June 2008 from the University of Sevilla. Since September 2008, she has been a PhD student at the Department of Computer Science and Artificial Intelligence of the University of Sevilla (Spain). She is also a member of the Research Group on Natural Computing in the same University. Her main research interests within the Membrane Computing area are computer simulation and Complexity Theory.

Miguel A. Martínez-del-Amor was born in July 10, 1984. He received his degree in Computer Science from the University of Murcia (Spain) in June 2008. Currently, he is a PhD student at the Department of Computer Science and Artificial Intelligence in the University of Sevilla (Spain). He is also a member of the Research Group on Natural Computing at the same University, and his main research interest is to joint Membrane Computing and High Performance Computing by using efficient computer simulations.

Enrique Orejuela-Pinedo was born in June 7, 1979. He received his degree in Biology in 2005, from the University of Sevilla (Spain). He has cooperated as internal student at the Departament of Ecology and Vegetal Biology in the University of Sevilla. Currently, he is a PhD student at the Department of Computer Science and Artificial Intelligence in the University of Sevilla. He is also a member of the Research Group on Natural Computing at the same University, and his main research interests are Natural Computing and Membrane Computing, specially computational models of ecosystems.

Ignacio Pérez-Hurtado was born in September 21, 1977. He received his degree in Computer Science in October 2003. He was systems analyst in a company for three years. Since September 2006, he has been a PhD student at the Department of Computer Science and Artificial Intelligence in the University of Sevilla (Spain). He is an associate professor at the same department. He is also

a member of the Research Group on Natural Computing at the said University, and his main research interests within the membrane computing are computer simulation, models for biological processes and Complexity Theory.

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 244-252

First Steps Towards a CPU Made of Spiking Neural P Systems

Miguel A. Gutiérrez-Naranjo, Alberto Leporati

Miguel A. Gutiérrez-Naranjo
University of Sevilla, Department of Computer Science and Artificial Intelligence
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
E-mail: magutier@us.es

Alberto Leporati
Università degli Studi di Milano – Bicocca, Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336/14, 20126 Milano, Italy
E-mail: alberto.leporati@unimib.it

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: We consider spiking neural P systems as devices which can be used to perform some basic arithmetic operations, namely addition, subtraction, comparison and multiplication by a fixed factor. The input to these systems are natural numbers expressed in binary form, encoded as appropriate sequences of spikes. A single system accepts as inputs numbers of any size. The present work may be considered as a first step towards the design of a CPU based on the working of spiking neural P systems.
Keywords: Spiking neural P systems, Arithmetic operations, Membrane Computing

1 Introduction

Spiking neural P systems (SN P systems, for short) have been introduced in [3] as a new class of distributed and parallel computing devices. They were inspired by *membrane systems* (also known as *P systems*) [12, 13, 7] and are based on the neurophysiological behavior of neurons sending electrical impulses to other neurons. In SN P systems the processing elements are called *neurons* and 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, namely the *spike*. Each neuron may also contain rules which allow to remove a given number of spikes from it, or to send spikes (possibly with a delay) to other neurons. The application of every rule is determined by checking the contents of the neuron against a regular set associated with the rule.

Formally, an SN P system of degree $m \geq 1$, as defined in [4], is a construct of the form $\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out)$, where $O = \{a\}$ is the singleton alphabet (a is called *spike*); $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, with $1 \leq i \leq m$, where: $n_i \geq 0$ is the *initial number of spikes* contained in σ_i; R_i is a finite set of *rules* of the following two forms:

(1) $E/a^c \rightarrow a;d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$ are integer numbers. If $E = a^c$, then it is usually written in the simplified form $a^c \rightarrow a;d$; similarly, if a rule $E/a^c \rightarrow a;d$ has $d = 0$, then we can simply write it as $E/a^c \rightarrow a$. Hence, if a rule $E/a^c \rightarrow a;d$ has $E = a^c$ and $d = 0$, then we can write $a^c \rightarrow a$;

(2) $a^s \rightarrow \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \rightarrow a;d$ of type (1) from R_i, we have $a^s \notin L(E)$ (where $L(E)$ denotes the regular language defined by E);

$syn \subseteq \{1,2,\ldots,m\} \times \{1,2,\ldots,m\}$, with $(i,i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons; $in, out \in \{1,2,\ldots,m\}$ indicate the *input* and *output* neurons of Π.

The rules of type (1) are called *firing* (also *spiking*) *rules*, and are applied as follows. If the neuron σ_i contains $k \geq c$ spikes, and $a^k \in L(E)$, then the rule $E/a^c \rightarrow a;d \in R_i$ can be applied. The execution of this rule removes c spikes from σ_i (thus leaving $k-c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i,j) \in syn$. If $d = 0$, then the spike is immediately emitted, otherwise it is emitted after d computation steps of the system. (Observe that, as usually happens in membrane computing, a global clock is assumed, marking the time for the whole system, hence the functioning of the system is synchronized.) If the rule is used in step t and $d \geq 1$, then in steps $t, t+1, t+2, \ldots, t+d-1$ the neuron is *closed*, so that it cannot receive new spikes (if a neuron has a

synapse to a closed neuron and tries to send a spike along it, then that particular spike is lost), and cannot fire new rules. In the step $t+d$, the neuron spikes and becomes open again, so that it can receive spikes (which can be used starting with the step $t+d+1$) and select rules to be fired.

Rules of type (2) are called *forgetting* rules, and are applied as follows: if the neuron σ_i contains *exactly* s spikes, then the rule $a^s \to \lambda$ from R_i can be used, meaning that all s spikes are removed from σ_i. In what follows we will use an *extended version* of forgetting rules, written in the form $E/a^s \to \lambda;d$. The application of these rules is analogous to that of firing rules. With respect to their basic version, extended forgetting rules are controlled by a regular expression, and may compete against firing rules for their application. It is possible to prove that the use of extended forgetting rules does not modify the computational power of SN P systems.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. In case two or more rules can be applied in a neuron at a given computation step, 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.

The *initial configuration* of the system is described by the numbers $n_1, n_2, \ldots, n_m$ of spikes present in each neuron, with all neurons being open. During the computation, a configuration is described by both the number of spikes present in each neuron and by the number of steps to wait until it becomes open (this number is zero if the neuron is already open). A *computation* in a system as above starts in the initial configuration. A positive integer number is given as input to a specified *input neuron*. Usually, the number is specified as the time elapsed between the arrival of two spikes. However, as discussed in [4], other possibilities exist: for example, we can consider the number of spikes initially contained in the input neuron, or the number of spikes read in a given interval of time. All these possibilities are equivalent from the point of view of computational power. To pass from a configuration to another one, for each neuron a rule is chosen among the set of applicable rules, and is executed. Generally, a computation may not halt. However, in any case the output of the system is usually considered to be the time elapsed between the arrival of two spikes in a designated *output cell*. Defined in this way, SN P systems compute (partial) functions of the kind $f : \mathbb{N} \to \mathbb{N}$; they can also indirectly compute functions of the kind $f : \mathbb{N}^k \to \mathbb{N}$ by using a bijection from $\mathbb{N}^k$ to $\mathbb{N}$. It is not difficult to show that SN P systems can simulate register machines [4], and hence are universal.

If we do not specify an input neuron (hence no input is taken from the environment) then we use SN P systems in the *generative* mode; we start from the initial configuration, and we look at the output produced by the system. Note that generative SN P systems are inherently nondeterministic, otherwise they would always reproduce the same sequence of computation steps, and hence the same output. Dually, we can neglect the output neuron and use SN P systems in the *accepting* mode; for $k \geq 1$, the natural numbers $n_1, n_2, \ldots, n_k$ are read in input and, if the computation halts, then the numbers are accepted. Also in these cases, SN P systems are universal computation devices [3, 4].

In this paper we consider SN P systems in a different way. We will use them to build the components of a restricted Arithmetic Logic Unit in which one or several natural numbers are provided in binary form, some arithmetic operation is performed and the result is sent out also in binary form. The arithmetic operations we will consider are addition, subtraction and multiplication among natural numbers. Each number will be provided to the system as a sequence of spikes: at each time step, zero or one spikes will be supplied to an input neuron, depending upon whether the corresponding bit of the number is `0` or `1`. Also the output neuron will emit the computed number to the environment in binary form, encoded as a spike train.

The paper is organised as follows. In section 2 we present an SN P system which can be used to add two natural numbers expressed in binary form, of any length (that is, composed of any number of bits). In section 3 we present an analogous SN P system, that computes the difference (subtraction) among two natural numbers. Section 4 contains the description of a very simple system that can be used to compare two natural numbers. Section 5 first extends the system presented in section 2 to perform the addition of any given set of natural numbers, and then describes a spiking neural P system that performs the multiplication of any natural number, given as input, by a fixed factor embedded into the system. Finally, section 6 concludes the paper and suggests some possible directions for future research.

2 Addition

In this section we describe a simple SN P system that performs the addition of two natural numbers. We call such a system the *SN P system for 2-addition*. It is composed of three neurons (see Figure 1): two *input* neurons and an *addition* neuron, which is also the output neuron. Both input neurons have a synapse to the addition neuron. Each input neuron receives one of the numbers to be added as a sequence of spikes, that encodes the number in

Time step	$Input_1$	$Input_2$	Add	Output
$t=0$	0	0	0	0
$t=1$	**0**	**1**	0	0
$t=2$	**0**	**0**	1	0
$t=3$	**1**	**1**	0	**1**
$t=4$	**1**	**0**	2	**0**
$t=5$	**1**	**1**	2	**0**
$t=6$	0	0	3	**0**
$t=7$	0	0	1	**1**
$t=8$	0	0	0	**1**

Table 1: Number of spikes in each neuron of Π_{Add}, and number of spikes sent to the environment, at each time step during the computation of the addition $11100_2 + 10101_2 = 110001_2$

binary form. As explained above, no spike in the sequence at a given time instant means 0 in the corresponding position of the binary expression, whereas one spike means 1. Note that the numbers provided as input to the system may be arbitrarily long. The input neurons have only one rule, $a \to a$, which is used to forward the spikes to the addition neuron as soon as they arrive. The addition neuron has three rules: $a \to a$, $a^2/a \to \lambda$ and $a^3/a^2 \to a$, which are used to compute the result.

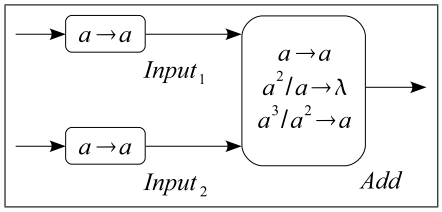


Figure 1: An SN P system that performs the addition among two natural numbers

Theorem 1. *The SN P system for 2-addition outputs the addition in binary form of two non-negative integers, provided to the neurons σ_{Input_1} and σ_{Input_2} in binary form.*

Proof. At the beginning of the computation, the system does not contain any spike. During the computation, neuron σ_{Add} may contain 0, 1, 2 or 3 spikes. We can thus divide the behavior of σ_{Add} in three cases:

• If there are no spikes, no rules are activated and in the next step 0 spikes are sent to the environment. This encodes the operation $0+0=0$.

• If there is 1 spike, then the rule $a \to a$ is triggered. The spike is consumed and one spike is sent out. This encodes $0+1=1+0=1$.

• If there are 2 spikes, then the rule $a^2/a \to \lambda$ is triggered. No spike is sent out and one spike (the carry) remains in the neuron for the next step.

• If there are 3 spikes, then the rule $a^3/a^2 \to a$ is applied. One spike is sent to the environment, two of them are consumed and one remains for the next step.

From this behavior, it is easily seen that the output is computed correctly. At the third computation step, the first of the spikes in the spike train that encodes the output in binary form is emitted by σ_{Add}. □

As an example, let us consider the addition $28+21=49$, that in binary form can be written as $11100_2 + 10101_2 = 110001_2$. Table 1 reports the number of spikes contained in each neuron of Π_{Add}, as well as the number of spikes sent to the environment, at each time step during the computation. The input and the output sequences are written in bold.

3 Subtraction

The *Subtraction SN P system*, illustrated in Figure 2, consists of ten neurons. The first input number, the *minuend*, is provided to neuron σ_{Input_1} in binary form, encoded as a spike train as described above. Similarly, the

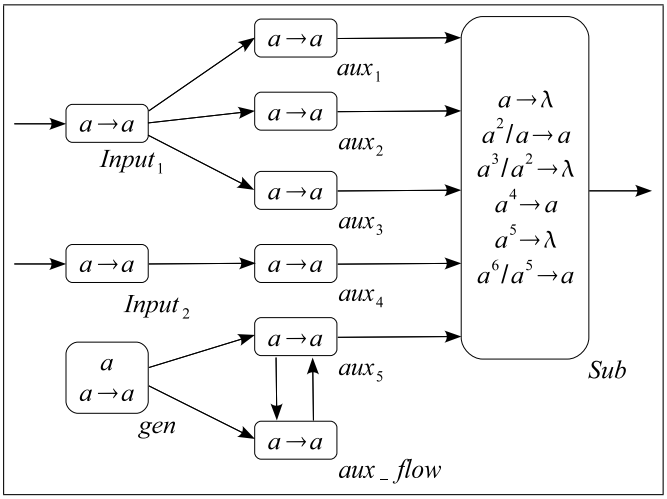


Figure 2: An SN P system that performs the subtraction among two natural numbers

Time step	$Input_1$	$Input_2$	aux_1	aux_2	aux_3	aux_4	aux_5	Sub	Output
$t=0$	0	0	0	0	0	0	0	0	0
$t=1$	**0**	**1**	0	0	0	0	1	0	0
$t=2$	**0**	**1**	0	0	0	1	1	1	0
$t=3$	**1**	**0**	0	0	0	1	1	2	0
$t=4$	**1**	**0**	1	1	1	0	1	3	**1**
$t=5$	**0**	**1**	1	1	1	0	1	5	**0**
$t=6$	**1**	**1**	0	0	0	1	1	4	**0**
$t=7$	**1**	0	1	1	1	1	1	2	**1**
$t=8$	0	0	1	1	1	0	1	6	**1**
$t=9$	0	0	0	0	0	0	1	5	**1**
$t=10$	0	0	0	0	0	0	1	1	**0**

Table 2: Number of spikes in each neuron of Π_{Sub}, and number of spikes sent to the environment, at each time step during the computation of the subtraction $1101100_2 - 110011_2 = 111001_2$

second input number (the *subtrahend*) is supplied in binary form to neuron σ_{Input_2}. The set of neurons σ_{aux_1}, σ_{aux_2} and σ_{aux_3} act as a multiplier of the minuend: they multiply by 3 the number of spikes provided by neuron σ_{Input_1}. The system contains also a subsystem composed of neurons σ_{gen}, σ_{aux_flow} and σ_{aux_5}, whose target is to provide a constant flow of spikes to σ_{Sub}. All the neurons mentioned up to now have only one rule: $a \to a$. The neurons σ_{aux_i}, for $1 \leq i \leq 5$, are connected with neuron σ_{Sub}; this is both the output neuron and the neuron in which the result of the subtraction is computed, by means of six rules: $a \to \lambda$, $a^2/a \to a$, $a^3/a^2 \to \lambda$, $a^4 \to a$, $a^5 \to \lambda$ and $a^6/a^5 \to a$. At the beginning of the computation all neurons are empty except σ_{gen}, which contains one spike.

Theorem 2. *The subtraction SN P system outputs the subtraction, in binary form, of two non-negative integer numbers, provided in binary form to neurons σ_{Input_1} (the minuend) and σ_{Input_2} (the subtrahend).*

The result can be easily checked by direct inspection of all possible cases. A detailed proof of this theorem — not given here, due to the lack of space — can be found in [2].

As an example let us calculate $108 - 51 = 57$, that in binary form can be written as $1101100_2 - 110011_2 = 111001_2$. Table 2 reports the number of spikes that occur in each neuron of Π_{Sub}, at each time step during the computation. Note that at each step only one rule is active in the subtraction neuron, and thus the computation is deterministic. The first time step in which the output starts to be emitted by the system is $t = 4$.

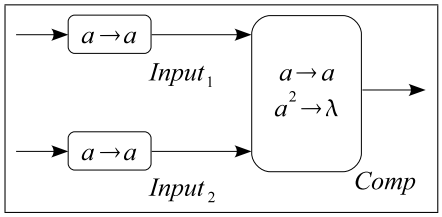


Figure 3: An SN P system that compares two natural numbers of any length, expressed in binary form

4 Checking Equality

Checking the equality of two numbers is a different task with respect to computing addition or subtraction. When comparing two numbers the output should be a binary mark, which indicates whether they are equal or not. Since an SN P system produces a spike train, we will encode the output as follows: starting from an appropriate instant of time, at each computation step the system will emit a spike if and only if the two corresponding input bits (that were inserted into the system some time steps before) are different. So doing, the system will emit no spike to the environment if the input numbers are equal, and at least one spike if they are different. Stated otherwise, if we compare two n-bit numbers then the output will also be an n-bit number: if such an output number is 0, then the input numbers are equal, otherwise they are different.

Bearing in mind these marks for equality and inequality, the design of the SN P system is trivial. It consists of three neurons: two input neurons, having $a \to a$ as the single rule, linked to a third neuron, the *checking neuron*. This checking neuron is also the output neuron, and it has only two rules: $a^2 \to \lambda$ and $a \to a$. The system is illustrated in Figure 3.

5 Multiplication

In this section we present a first approach to the problem of computing the multiplication of two binary numbers by means of SN P systems. The main difference between multiplication and the addition or subtraction operations presented in the previous sections is that in addition and subtraction the n-th digit in the binary representation of the inputs is used exactly once, to compute the n-th digit of the output, and then it can be discarded. On the contrary, in the usual algorithm for multiplication the different digits of the inputs are reused several times; hence the design of a device that executes this algorithm needs some kind of memory. Other algorithms for multiplication, such as Booth's algorithm (see, for example, [1]) also need some kind of memory, to store the intermediate results.

We propose a family of SN P systems for performing the multiplication of two non-negative integer numbers. In these systems only one number, the multiplicand, is provided as input; the other number, the multiplier, is instead encoded in the structure of the system. The family thus contains one SN P system for each possible multiplier.

In the design of our systems, we exploit the following basic fact concerning multiplication by one binary digit: any number remains the same if multiplied by 1, whereas it produces a 0 if multiplied by zero. Bearing this fact in mind, an SN P system associated to a fixed multiplier only needs to add different copies of the multiplicand, by feeding such copies to an addition device with the appropriate delay. Before presenting this design, we extend the 2-addition SN P system from section 2 to an n-addition SN P system.

5.1 Adding n numbers

In this section we present a family $\{\Pi_{Add}(n)\}_{n\in\mathbb{N}}$ of SN P systems which allows to add numbers expressed in binary form. Precisely, for any integer $n \geq 2$ the system $\Pi_{Add}(n)$ computes the sum of n natural numbers. In what follows we will call $\Pi_{Add}(n)$ the SN P system for n-addition. For $n = 2$ we will obtain the SN P system for 2-addition that we have described in section 2.

The system $\Pi_{Add}(n)$ consists of $n+1$ neurons: n *input* neurons and one *addition neuron*, which is also the output neuron. Each input neuron has only one rule, $a \to a$, and is linked to the addition neuron. This latter neuron computes the result of the computation by means of n rules r_i, $i \in \{1,\ldots,n\}$, which are defined as follows: $r_i \equiv a^i/a^{k+1} \to a$ if i is odd and $i = 2k+1$, whereas $r_i \equiv a^i/a^k \to \lambda$ if i is even and $i = 2k$.

As an example, Figure 4 shows $\Pi_{Add}(5)$, the SN P system for 5-addition.

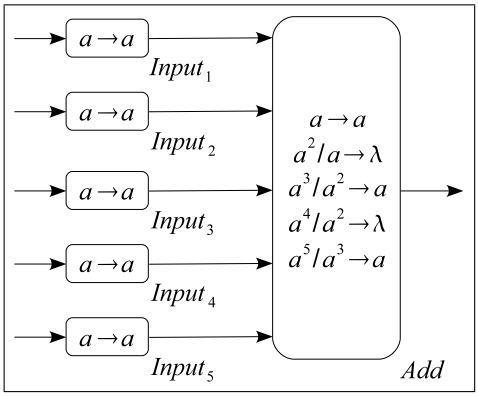


Figure 4: An SN P system that performs the addition among five natural numbers

Theorem 3. *The SN P system for n-addition outputs the addition in binary form of n non-negative integer numbers, provided to the neurons $\sigma_{Input_1}, \ldots, \sigma_{Input_n}$ in binary form.*

Proof. Let $A_1, \ldots, A_n$ be the n numbers to be added, and let $a_i^p a_i^{p-1} \ldots a_i^0$ be the binary expression of A_i, $1 \leq i \leq n$, padded with zeros on the left to obtain $(p+1)$-digit numbers (where $p+1$ is the maximum number of digits among the binary representations of $A_1, \ldots, A_n$). Hence we can write $A_i = \sum_{k=0}^{p} a_i^k 2^k$ for all $i \in \{1, 2, \ldots, n\}$.

For each $i \in \{1, \ldots, n\}$, let A_i' be the number with binary expression $a_i^p \ldots a_i^1$, i.e., $A_i' = \sum_{k=1}^{p} a_i^k 2^{k-1}$. Moreover, let $U = \sum_{i=1}^{n} a_i^0$ and let $k \in \mathbb{N}$ and $\alpha \in \{0, 1\}$ such that $U = 2k + \alpha$ ($\alpha = 1$ if U is odd and $\alpha = 0$ if U is even). The addition of $A_1, \ldots, A_n$ can be written as:

$$\sum_{i=1}^{n} A_i = \sum_{i=1}^{n} \sum_{k=0}^{p} a_i^k 2^k = \left(\sum_{i=1}^{n} \sum_{k=1}^{p} a_i^k 2^k \right) + \sum_{i=1}^{n} a_i^0 = 2 \left(\sum_{i=1}^{n} A_i' + k \right) + \alpha$$

According to this formula, if $b_r \ldots b_0$ is the binary expression of $\sum_{i=1}^{n} A_i$, then $b_0 = \alpha$ and $b_r \ldots b_1$ is the binary expression of $\sum_{i=1}^{n} A_i' + k$.

Let us assume now that at the time instant t there are i spikes in neuron σ_{Add}. These spikes can come from the input neurons, or they may have remained from the previous computation step. Let us compute b_t, the t-th digit of the output, dividing the problem in the following cases.

• Let us assume that i is odd and $i = 2k + 1$. Then, according to the previous formula, $b_t = 1$ and k units should be added to the computation of the next digit. This operation is performed by the rule $a^i/a^{k+1} \to a$. By applying this rule, one spike is sent to the environment ($b_t = 1$) and $k+1$ spikes are consumed, so that $i - (k+1) = 2k + 1 - (k+1) = k$ spikes remain.

• Let us assume that i is even and $i = 2k$. Then, according to the previous formula, $b_t = 0$ and k units should be added to the computation of the next digit. This operation is performed by the rule $a^i/a^k \to \lambda$. By applying this rule, no spike is sent to the environment ($b_t = 0$) and k spikes are consumed, so that $i - k = 2k - k = k$ spikes remain for the next step. □

As an example, let us consider the addition of the numbers 3, 4, 2, 7 and 1, whose binary representations are 11_2, 100_2, 10_2, 111_2 and 1_2, respectively. Table 3 shows the evolution of the number of spikes in the neurons of the SN P system $\Pi_{Add}(5)$ (illustrated in Figure 4), as well as the number of spikes sent to the environment at each computation step, when performing such an addition. The input and the output sequences are written in bold. According with the computation, the result of the addition is $17 = 10001_2$.

5.2 Multiplication by a fixed multiplier

We now describe a family $\{\Pi_{Mult}(n)\}_{n \in \mathbb{N}}$ of SN P systems, one for each natural number n, that operate as multiplier devices. Precisely, the system $\Pi_{Mult}(n)$ takes as input a number in binary form, and outputs the input multiplied by n. The output is also expressed in binary form.

Given a natural number n, the SN P system $\Pi_{Mult}(n)$ is defined as follows. It consists of one *input* neuron, σ_{Input}, linked to k neurons $\sigma_{aux_{1_1}}, \ldots, \sigma_{aux_{k_1}}$, where k is the number of occurrences of the digit 1 in the binary

Time step	$Input_1$	$Input_2$	$Input_3$	$Input_4$	$Input_5$	Add	Output
$t=1$	**1**	**0**	**0**	**1**	**1**	0	0
$t=2$	**1**	**0**	**1**	**1**	0	3	0
$t=3$	0	**1**	0	**1**	0	4	**1**
$t=4$	0	0	0	0	0	4	**0**
$t=5$	0	0	0	0	0	2	**0**
$t=6$	0	0	0	0	0	1	**0**
$t=7$	0	0	0	0	0	0	**1**

Table 3: Number of spikes in each neuron of $\Pi_{Add}(5)$ (the system illustrated in Figure 4) and number of spikes sent to the environment, at each time step during the computation of the addition $11_2+100_2+10_2+111_2+1_2=10001_2$

representation of n. For each $i\in\{1,\ldots,k\}$, neuron $\sigma_{aux_{i_1}}$ is connected with a new neuron $\sigma_{aux_{i_2}}$, which is connected with $\sigma_{aux_{i_3}}$, etc. This sequence of neurons is a path of linked neurons that extends until reaching $\sigma_{aux_{ij_i}}$, where j_i is the number of order of the corresponding digit in the binary representation of n, where the first digit corresponds to 2^0, the second one corresponds to 2^1, and so on. All the last neurons of the k sequences are connected with a final neuron σ_{Add}, which is the same as the output neuron of the k-addition SN P system $\Pi_{Add}(k)$ described above. This neuron has the rules for the addition of k natural numbers. All the other neurons have only the rule $a\to a$.

For example, let us consider $n=26$, whose binary representation is 11010_2. Such a representation has three digits equal to 1, at the positions 2, 4 and 5. The system $\Pi_{Mult}(26)$, illustrated in Figure 5, has 13 neurons: σ_{Input}, σ_{Add}, and three sequences of neurons associated with the three digits equal to 1: $\sigma_{aux_{11}}$ and $\sigma_{aux_{12}}$, corresponding to the 1 in the second position (corresponding to the power 2^1); $\sigma_{aux_{21}}$, $\sigma_{aux_{22}}$, $\sigma_{aux_{23}}$ and $\sigma_{aux_{24}}$, corresponding to the 1 in the fourth position (corresponding to the power 2^3); $\sigma_{aux_{31}}$, $\sigma_{aux_{32}}$, $\sigma_{aux_{33}}$, $\sigma_{aux_{34}}$ and $\sigma_{aux_{35}}$, corresponding to the 1 in the fifth position (corresponding to the power 2^4).

The last neurons of these sequences, namely $\sigma_{aux_{12}}$, $\sigma_{aux_{24}}$ and $\sigma_{aux_{35}}$, are linked to neuron σ_{Add}, which is also the output neuron. The rules of this neuron are $a\to a$, $a^2/a\to\lambda$ and $a^3/a^2\to a$, which are the same as in the addition neuron of the 3-addition SN P system $\Pi_{Add}(3)$ described in the previous section.

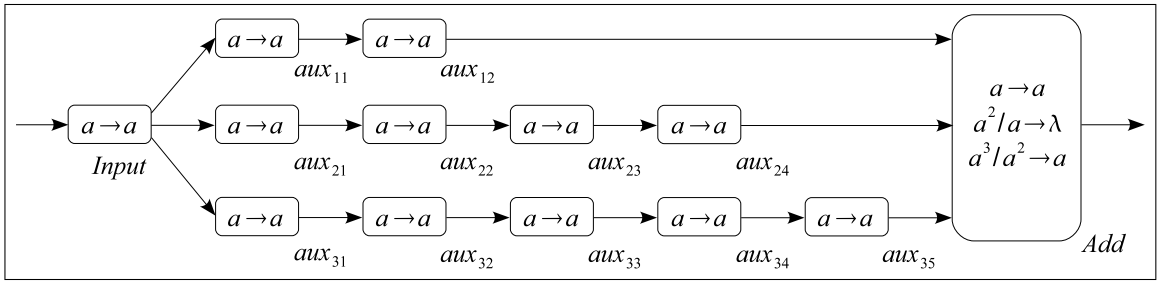


Figure 5: An SN P system that computes the product among the natural number given as input (in binary form) and the fixed multiplier $26=11010_2$, encoded in the structure of the system

Theorem 4. *The SN P system $\Pi_{Mult}(n)$ built as above takes as input a number m in binary form and outputs the result of the multiplication $m\cdot n$ in binary form.*

Proof. Since we already proved that the neuron σ_{Add} performs the addition of several numbers in binary form, it only remains to transform the multiplication $m\cdot n$ (where n is a fixed parameter) into an appropriate addition. To this aim, let $n=\sum_{j=0}^{q}n_j2^j$. Then we can write

$$m\cdot n=m\cdot\left(\sum_{j=0}^{q}n_j2^j\right)=\sum_{j=0}^{q}\left(m\cdot 2^j\right)n_j=\sum_{0\le j\le q\,\wedge\,n_j=1}\left(m\cdot 2^j\right)$$

According to this expression, $m\cdot n$ can be calculated as the addition of as many copies of m as the number of digits n_j equal to 1 that appear in the binary representation of n. Such copies have to be padded with j zeros

Time step	*Input*	aux_{12}	aux_{24}	aux_{35}	*Add*	*Out*
$t=1$	**1**	0	0	0	0	0
$t=2$	**0**	0	0	0	0	0
$t=3$	**1**	1	0	0	0	0
$t=4$	**1**	0	0	0	1	**0**
$t=5$	**1**	1	1	0	0	**1**
$t=6$	0	1	0	1	2	**0**
$t=7$	0	1	1	0	3	**0**
$t=8$	0	0	1	1	3	**1**
$t=9$	0	0	1	1	3	**1**
$t=10$	0	0	0	1	3	**1**
$t=11$	0	0	0	0	2	**1**
$t=12$	0	0	0	0	1	**0**
$t=13$	0	0	0	0	0	**1**

Table 4: Number of spikes in neurons $\sigma_{aux_{12}}$, $\sigma_{aux_{24}}$, $\sigma_{aux_{35}}$ and σ_{Add} of $\Pi_{Mult}(26)$ (the system illustrated in Figure 5) and number of spikes sent to the environment, at each time step during the computation of the multiplication $11101_2 \cdot 11010_2 = 1011110010_2$

on the right (that is, they have to be multiplied by 2^j), to take into account the correct weight of n_j. Hence, if $k = \sum_{j=0}^{q} n_j$ then to compute $m \cdot n$ it suffices to provide k copies of m — each shifted in time of a number of steps that corresponds to the weight of a bit n_j equal to 1 — to a neuron that computes the addition of k natural numbers. □

6 Conclusion and Future Work

In this paper we have presented some simple SN P systems that perform the following operations: addition, multiple addition, comparison, and multiplication by a fixed factor. All the numbers given as inputs to these systems are expressed in binary form, encoded as a spike train in which at each time instant the presence of a spike denotes 1, and the absence of a spike denotes 0. The outputs of the computations are also expelled to the environment in the same form.

The motivation for this work lies in the fact that we would like to implement a CPU using only spiking neural P systems. To this aim, the first step is to design the Arithmetic Logic Unit of the CPU, and hence to study a compact way to perform arithmetical and logical operations by means of spiking neural P systems. Ours is certainly not the unique possible way to approach the problem; other two possibilities are: (1) implementing the CPU as a network composed of AND/OR/NOT Boolean gates, and (2) simulating the CPU by means of register machines. In both cases, using techniques widely known in the literature, one could design an SN P system that simulates the Boolean network (resp., the register machine), thus implementing the CPU.

In any case, an interesting extension to the present work is to try to design an SN P system for the multiplication, where both the numbers m and n to be multiplied are supplied as inputs. And, of course, we would also need a system to compute the integer division between two natural numbers; probably, this last system is the most difficult to design.

Acknowledgement

The first author wishes to acknowledge the support of the project TIN2006–13425 of Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200. The second author was partially supported by the MIUR project “Mathematical aspects and emerging applications of automata and formal languages" (2007).

Bibliography

[1] M.J. Flynn. *Advanced computer arithmetic design*. John Wiley Publisher, 2001.

[2] M.A. Gutiérrez-Naranjo and A. Leporati. Performing arithmetic operations with spiking neural P systems. In *Proc. of the Seventh Brainstorming Week on Membrane Computing*, Vol. I, Fénix Editora, Seville, Spain, 2009, 181–198. Availble at `http://www.gcn.us.es`.

[3] M. Ionescu, Gh. Păun and T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, **71**(2-3):279–308, 2006.

[4] M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez. Computing with spiking neural P systems: Traces and small universal systems. In *DNA Computing,* 12^{th} *International Meeting on DNA Computing (DNA12)*, Revised Selected Papers, LNCS 4287, Springer, 2006, 1–16.

[5] Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**:108–143, 2000. See also Turku Centre for Computer Science — TUCS Report No. 208, 1998.

[6] Gh. Păun. *Membrane computing. An introduction*. Springer–Verlag, 2002.

[7] The P systems web page: `http://ppage.psystems.eu/`

Miguel A. Gutiérrez-Naranjo is an associate professor at the Department of Computer Science and Artificial Intelligence of the University of Seville in Spain. He obtained his doctoral degree in Mathematics in 2002. His main research area is Natural Computing, with a special interest in Membrane Computing.

Alberto Leporati obtained a Ph.D. in Computer Science from the University of Milano (Italy) in 2002. Since 2004, he is assistant professor at the University of Milano – Bicocca. His research interests are in Membrane Computing, Theoretical Computer Science and Computational Complexity.

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 253-262

Mutation Based Testing of P Systems

Florentin Ipate, Marian Gheorghe

Florentin Ipate
The University of Pitesti
Department of Computer Science, Faculty of Mathematics and Computer Science
Str Targu din Vale 1, 110040 Pitesti
E-mail: florentin.ipate@ifsoft.ro

Marian Gheorghe
The University of Sheffield
Department of Computer Science
Regent Court, Portobello Street, Sheffield S1 4DP, UK
E-mail: M.Gheorghe@dcs.shef.ac.uk

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: Although testing is an essential part of software development, until recently, P system testing has been completely neglected. Mutation testing (mutation analysis) is a structural software testing method which involves modifying the program in small ways. In this paper, we provide a formal way of generating mutants for systems specified by context-free grammars. Furthermore, the paper shows how the proposed method can be used to construct mutants for a P system specification.
Keywords: mutation testing, P systems, Kripke structures, context-free grammars

1 Introduction

Membrane computing, the research field initiated by Gheorghe Păun in 1998 [12], aims to define computational models, called P systems, which are inspired by the behaviour and structure of the living cell. Since its introduction in 1998, the P system model has been intensively studied and developed: many variants of membrane systems have been proposed, a research monograph [13] has been published and regular collective volumes are annually edited. Furthermore, a comprehensive bibliography of P systems can be found at [16]. Of the many variants of P systems that have been defined, in this paper we consider cell-like P systems without priority and membrane dissolving rules [13].

Testing is an essential part of software development and all software applications, irrespective of their use and purpose, are tested before being released. Testing is not a replacement for a formal verification procedure, when the former is also present, but rather a complementary mechanism to increase the confidence in software correctness [5]. Although formal verification has been applied to different models based on P systems [1], until recently testing has been completely neglected in this context.

The main testing strategies involve either (1) knowing the specific function or behaviour a product is meant to deliver (functional or black-box testing) or (2) knowing the internal structure of the product (structural or white-box testing). In black-box testing, the test generation is based on a formal specification or model, in which case the process could be automated. A number of recent papers devise black-box testing strategies for P systems based on rule coverage [4], finite state machine [8] and stream X-machine [7] conformance techniques. In this paper, we propose an approach to P system testing based on mutation analysis.

Mutation testing (mutation analysis) is a structural software testing method which involves modifying the program in small ways [14], [9]. The modified versions of the program are called *mutants*.

Consider, for example, the following fragment of a Java program:

if $(x \geq 0 \&\& a)$ $y = y+1$; else $y = y+2$;

Then mutants for this code fragment can be obtained by substituting: (i) && with another logic operator, e.g., ||; (ii) $\geq$ with another comparison operator, e.g., $>$, $=$; (iii) $+$ with another arithmetic operators, e.g., $-$; (iv) substituting one variable (e.g., x) with another one, e.g., y (we assume that the two variables have the same type).

Some (not all) mutants of the above code fragment are given below.

if $(x \geq 0 || a)$ $y = y+1$; else $y = y+2$;
if $(x > 0 \&\& a)$ $y = y+1$; else $y = y+2$;
if $(x \geq 0 \&\& a)$ $y = y-1$; else $y = y+2$;
if $(x \geq 0 \&\& a)$ $y = y+1$; else $y = y-2$;
if $(x \geq 0 \&\& a)$ $x = y+1$; else $y = y+2$;
if $(x \geq 0 \&\& a)$ $y = y+1$; else $x = y+2$;

A variety of *mutation operators* (ways of introducing errors into the correct code) for imperative languages are defined in the literature [9], [10] (a few examples are given above). These are called traditional mutation operators. Beside these, there are mutation operators for specialised programming environments, such as object-oriented languages [10]. A popular tool for generating mutants for Java programs is MuJava [15], [10].

The underlying idea behind mutation testing is that, in practice, an erroneous program either differs only in a small way from the correct program or, alternatively, a bigger fault can be expressed as the summation of smaller (basic) faults and so, in order to detect the fault, the appropriate mutants need to be generated. If the test suite is able to detect the fault (i.e., one of the tests fails), then the mutant is said to be killed. Two kinds of mutation have been defined in the literature: *weak mutation* requires the test input to cause different program states for the mutant and the original program; *strong mutation* requires the same condition but also the erroneous state to be propagated at the end of the program.

Mutation analysis has been largely used in white-box testing, but only a few tentative attempts to use this idea in black-box testing have been reported in the literature [11]. Offutt et al. propose a general strategy for developing mutation operators for a grammar based software artefact, but the ideas that outline the proposed strategy for mutation operator development are rather vague and general and no formalisation is provided.

In this paper we provide a formal way of generating mutants for systems specified by context-free grammars. Given such a specification, a derivation (or parse) tree can be associated with it. Based on the tree, we formally describe the process of generating the mutants for the given specification. Furthermore, the paper shows how the proposed method can be used to construct mutants for a P system specification.

2 Preliminaries

For an alphabet $V = \{a_1, ..., a_p\}$, V^* denotes the set of all strings over V; λ denotes the empty string. For a string $u \in V^*$, $|u|_{a_i}$ denotes the number of a_i occurrences in u. Each string u has an associated vector of non-negative integers $(|u|_{a_1}, ..., |u|_{a_p})$. This is denoted by $\Psi_V(u)$.

The concept of context-free grammar is assumed to be known, for details we refer to a classical textbook [6]. Only proper context-free grammar, i.e., with no useless symbols and no λ or renaming productions, will be used in this paper. For any derivation from the start symbol to a string of terminal symbols, w, a derivation (or parse) tree with the yield, the string of terminals obtained by concatening the leaves from left to right, w, is associated. The set of terminal strings derived from the start symbol

is called the language generated by the language. A grammar is said to be *ambiguous* if there exists a string and in any leftmost derivation (always the leftmost nonterminal is rewritten) this can be generated by more than one derivation (parse) tree. In the sequel possibly ambiguous grammars will be considered.

2.1 P systems

A basic cell-like P system is defined as a hierarchical arrangement of membranes identifying corresponding regions of the system. With each region there are associated a finite multiset of objects and a finite set of rules; both may be empty. A multiset is either denoted by a string $u \in V^*$, where the order is not considered, or by $\Psi_V(u)$. The following definition refers to one of the many variants of P systems, namely cell-like P systems, which uses non-cooperative transformation and communication rules [13]. We will call these processing rules. Since now onwards we will refer to this model as simply P system.

Definition 1. A *P system* is a tuple $\Pi = (V, \mu, w_1, ..., w_n, R_1, ..., R_n)$, where V is a finite set, called *alphabet*; μ defines the membrane structure, which is a hierarchical arrangement of n compartments called *regions* delimited by *membranes* - these membranes and regions are identified by integers 1 to n; w_i, $1 \leq i \leq n$, represents the initial multiset occurring in region i; R_i, $1 \leq i \leq n$, denotes the set of processing rules applied in region i.

The membrane structure, μ, is denoted by a string of left and right brackets ([, and]), each with the label of the membrane it points to; μ also describes the position of each membrane in the hierarchy. The rules in each region have the form $u \rightarrow (a_1, t_1)...(a_m, t_m)$, where u is a multiset of symbols from V, $a_i \in V$, $t_i \in \{in, out, here\}$, $1 \leq i \leq m$. When such a rule is applied to a multiset u in the current region, u is replaced by the symbols a_i with $t_i = here$; symbols a_i with $t_i = out$ are sent to the outer region or outside the system when the current region is the external compartment and symbols a_i with $t_i = in$ are sent into one of the regions contained in the current one, arbitrarily chosen. In the following definitions and examples all the symbols $(a_i, here)$ are used as a_i. The rules are applied in maximally parallel mode which means that they are used in all the regions in the same time and in each region all the symbols that may be processed, must be.

A configuration of the P system Π, is a tuple $c = (u_1, ..., u_n)$, where $u_i \in V^*$, is the multiset associated with region i, $1 \leq i \leq n$. A derivation of a configuration c_1 to c_2 using the maximal parallelism mode is denoted by $c_1 \Longrightarrow c_2$. In the set of all configurations we will distinguish terminal configurations; $c = (u_1, ..., u_n)$ is a terminal configuration if there is no region i such that u_i can be further derived.

For the type of P systems we investigate in this paper, multi-membranes can be equivalently collapsed into one membrane through properly renaming symbols in a membrane. Thus, for the sake of convenience, subsequently we will only focus on P systems with only one membrane.

2.2 Kripke structures

Definition 2. A Kripke structure over a set of atomic propositions AP is a four tuple $M = (S, H, I, L)$, where S is a finite set of states; $I \subseteq S$ is a set of initial states; $H \subseteq S \times S$ is a transition relation that must be left-total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $(s, s') \in H$; $L : S \longrightarrow 2^{AP}$ is an interpretation function, that labels each state with the set of atomic propositions true in that state.

Usually, the Kripke structure representation of a system results by giving values to every variable in each configuration of the system. Suppose $var_1, \ldots, var_n$ are the system variables, Val_i denotes the set of values for var_i and val_i is a value from Val_i, $1 \leq i \leq n$. Then the states of the system are $S = \{(val_1, \ldots, val_n) \mid val_1 \in Val_1, \ldots, val_n \in Val_n\}$, and the set of atomic predicates are $AP = \{(var_i = val_i) \mid 1 \leq i \leq n, val \in Val_i\}$. Naturally, L will map each state (given by the values of variables) onto the corresponding set of atomic propositions. Additionally, a halt (sink) state is needed when H is not left-total and an extra atomic proposition, that indicates that the system has reached this state, is added to AP. For convenience, in the sequel AP and L will be omitted from the definition of a Kripke structure.

3 Mutation testing from a context-free grammar

In this section we provide a way of constructing mutants for systems specified by context-free grammars. Given the system specification, in the form of a parse tree, we formally describe the generation of mutants for the given specification.

Consider a context-free grammar $G = (V, T, P, S)$ and $L(G)$ the language defined by G. We assume that, for every production rule $p : A \longrightarrow X_1 \dots X_k$, we have defined a set $Mut(p)$, called the *set of mutants* of p. A mutant p' of p is a production rule of the form $A \longrightarrow X_1' \dots X_n'$ such that each symbol $X_1', \dots, X_n'$ is either a terminal or is found among $X_1, \dots, X_k$. Furthermore, p' is either a production rule of G itself or has the form $A \longrightarrow A$, $A \in V$; this condition ensures that the yield of the mutated tree is syntactically correct.

Among the mutants of p, the following types of mutants can be distinguished:

- A *terminal replacement mutant* is a production rule of the form $A \longrightarrow X_1' \dots X_k'$ if there exists j, $1 \leq j \leq k$, such that $X_j, X_j' \in T$, $X_j \neq X_j'$ and $X_i' = X_i$, $1 \leq i \leq n$, $i \neq j$.
- A *terminal insertion mutant* is a production rule of the form $A \longrightarrow w$ where w is obtained by inserting one terminal into the string $X_1 \dots X_k$ (at any position).
- A *string deletion mutant* is a production rule of the form $A \longrightarrow w$ where w is obtained by removing one or more symbols from $X_1 \dots X_k$.
- A *string reordering mutant* is a production rule of the form $A \longrightarrow w$ where w is obtained by reordering the string $X_1 \dots X_k$.

Given any parse tree Tr for G, the set of mutants of Tr is defined as follows:

- A one-node tree has no mutants.
- Let Tr be the tree with root A and subtrees $Tr_1, \dots, Tr_k$ having roots, nodes $X_1, \dots, X_k$, respectively and $p \in P$ the corresponding production rule of G, of the form $A \longrightarrow X_1 \dots X_k$. This is denoted by $Tr = MakeTree(A, Tr_1, \dots, Tr_k)$. Let Tr' denote a mutant of Tr. Then either
  - (**subtree mutation**) $Tr' = MakeTree(A, Tr_1', \dots, Tr_k')$, where there exists j, $1 \leq j \leq k$, such that Tr_j' is mutant of Tr_j and $Tr_i' = Tr_i$, $1 \leq i \leq k$, $i \neq j$, or
  - (**rule mutation**) $Tr' = MakeTree(A, Tr_1', \dots, Tr_n')$, where there exists a mutant p' of p of the form $A \longrightarrow X_1' \dots X_n'$ such that for every i, $1 \leq i \leq n$, there exists j_i, $1 \leq j_i \leq k$, such that $Tr_i' = Tr_{j_i}$.

According to [11] these operations can be made such as to keep the result produced by them in the same language or in a larger one. In the first case a much simpler approach can be considered whereby each rule having a certain nonterminal in the left hand side is replaced by another different rule having the same nonterminal as left hand side. However the above set of operations provide a two stage method which generates mutants by considering first the rule level and then the derivation (parse) tree. If these operations are restricted to produce strings in the same language then we have the following result.

Lemma 3. *Every mutant of a parse tree from G is also a parse tree from G.*

Proof. Follows by induction on the depth of the tree. □

Thus, the yield of any mutant constructed as above belongs to the language described by G and so only *syntactically correct* mutants will be generated. Syntactically incorrect mutants are useless (they do not produce test data) and so the complexity of the testing process is reduced by making sure that these are ruled out from the outset.

Let us consider the grammar $G = (V,T,P,S)$ where $V = \{S\}$; $T = \{0,\ldots,N\} \cup \{+,-\}$, with N a fixed upper bound; $P = \{p_1, p_2\} \cup \{p_3^i \mid 0 \leq i \leq N\}$, with $p_1 : S \longrightarrow S+S$, $p_2 : S \longrightarrow S-S$, $p_3^i : S \longrightarrow i$, $0 \leq i \leq N$.

Suppose we have the following rule mutants:

- for $p_1 : S \longrightarrow S-S$ (terminal replacement), $S \longrightarrow S$ (string deletion)
- for $p_2 : S \longrightarrow S+S$ (terminal replacement), $S \longrightarrow S$ (string deletion)
- for $p_3^i : S \longrightarrow i-1$ and $S \longrightarrow i+1$ if $1 < i < N$, $S \longrightarrow 1$ if $i = 0$ and $S \longrightarrow N-1$ if $i = N$. The mutants of p_3^i are of terminal replacement type and are based on a technique widely used in software testing practice, called boundary value analysis. According to practical experience, many errors tend to lurk close to boundaries; thus, an efficient way to uncover faults is to look at the neighbouring values

Consider the string $1+2-3$ and a parse tree for this string as represented in Figure 1 (leaf nodes are in bold). The construction of mutants for the given parse tree is illustrated in Figures 2, 3 and 4. Thus, the mutated strings are $0+2-3$, $2+2-3$, $1+1-3$, $1+3-3$, $1-3$, $2-3$, $1+2-2$, $1+2-4$, $1+2+3$, $1-2-3$, $1+2$, 3. Some of these produce the same result as the original string; these are called *equivalent mutants*. Since no input value can distinguish these mutants from the correct string, they will not affect the test suite when strong mutation is considered.

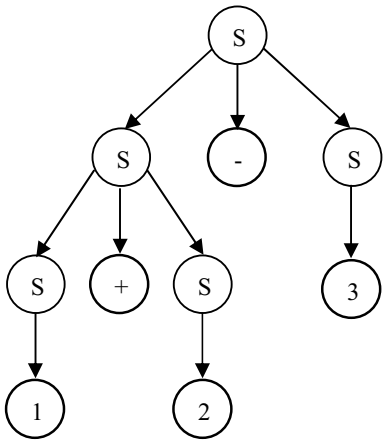


Figure 1: Example parse tree

4 P system mutation testing

Consider a 1-membrane P system $\Pi = (V, \mu, w, R)$, where $R = \{r_1, \ldots, r_m\}$; each rule r_i, $1 \leq i \leq m$, is of the form $u_i \longrightarrow v_i$, where u_i and v_i are multisets over the alphabet V. In the sequel, we treat the multisets as vectors of non-negative integers, that is each multiset u is replaced by $\Psi_V(u) \in \boldsymbol{N}^k$, where k denotes the number of symbols in V. In order to keep the number of configurations finite we will assume that each component of a configuration u cannot exceed an established upper bound denoted Max. We denote $u \leq Max$ if $u_i \leq Max$ for every $1 \leq i \leq k$ and $N_{Max}^k = \{u \in \boldsymbol{N}^k \mid u \leq Max\}$. Analogously to [3], the system is assumed to crash whenever $u \leq Max$ does not hold (this is different from the normal termination, which occurs when $u \leq Max$ and no rule can be applied). Under these conditions, the 1-membrane P system Π can be described by a Kripke structure. In order to define the Kripke structure equivalent of Π we use two predicates, $MaxParal$ and $Apply$, defined by:

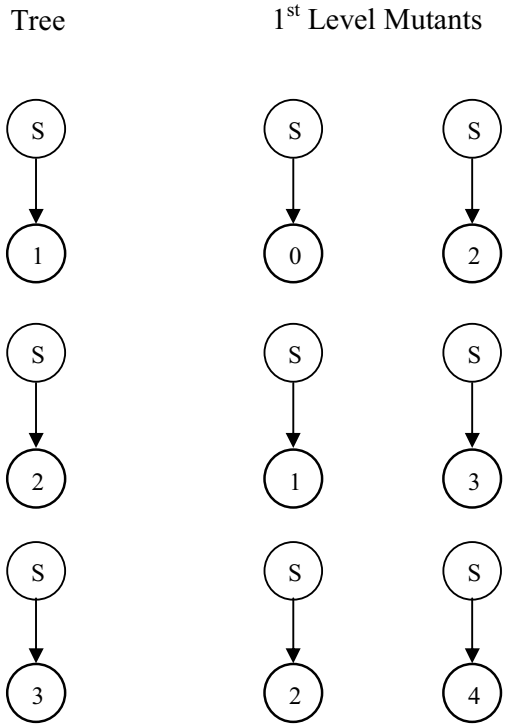


Figure 2: 1st level mutants

$MaxParal(u,u_1,v_1,n_1,\ldots,u_m,v_m,n_m)$, $u \in N^k_{Max}$, $n_1,\ldots,n_m \in \boldsymbol{N}$ signifies that a derivation of the configuration u in maximally parallel mode is obtained by applying rules $r_1 : u_1 \longrightarrow v_1,\ldots,r_m : u_m \longrightarrow v_m$, $n_1,\ldots,n_m$ times, respectively; $Apply(u,v,u_1,v_1,n_1,\ldots,u_m,v_m,n_m)$, $u \in N^k_{Max}$, $n_1,\ldots,n_m \in \boldsymbol{N}$, denotes that v is the result of applying rules $r_1,\ldots,r_m$, $n_1,\ldots,n_m$ times, respectively.

Then the Kripke structure equivalent $M = (S,H,I,L)$ of Π is defined as follows: $S = N^k_{Max} \cup \{Halt, Crash\}$ with $Halt, Crash \notin N^k_{Max}$, $Halt \neq Crash$; $I = w$; H is defined by:

- $(u,v) \in H$, $u,v \in N^k_{Max}$, if $\exists n_1,\ldots,n_m \in \boldsymbol{N} \cdot MaxParal(u,u_1,v_1,n_1,\ldots,u_m,v_m,n_m) \wedge$ $Apply(u,v,u_1,v_1,n_1,\ldots,u_m,c_m,n_m)$;
- $(u,Halt) \in H$, $u \in N^k_{Max}$, if $\neg\exists v \in N^k_{Max}, n_1,\ldots,n_m \in \boldsymbol{N} \cdot$ $Apply(u,v,u_1,v_1,n_1,\ldots,u_m,v_m,n_m)$;
- $(u,Crash) \in H$ if $\neg\exists v \in N^k_{Max} \cup \{Halt\} \cdot (u,v) \in H$;
- $(Halt,Halt) \in H$, $(Crash,Crash) \in H$.

It can be observed that the relation H is left-total.

In order to use mutation analysis in P system testing we first have to describe an appropriate context-free grammar, such that the P system specification can be written as a string accepted by this grammar. The parse tree for the string is then generated and the procedure presented in the previous section is used for mutant construction.

The grammar definition will depend on the level at which testing is intended to be performed. At a high level (for instance in integration testing) the predicates *MaxParal* and *Apply* will normally be assumed to be correctly implemented and so they will be presented as terminals in the grammar; obviously, they can be themselves described by context-free grammars and appropriate mutants will be generated in a similar fashion. On the other hand, it is possible to incorporate the definitions of the two predicates

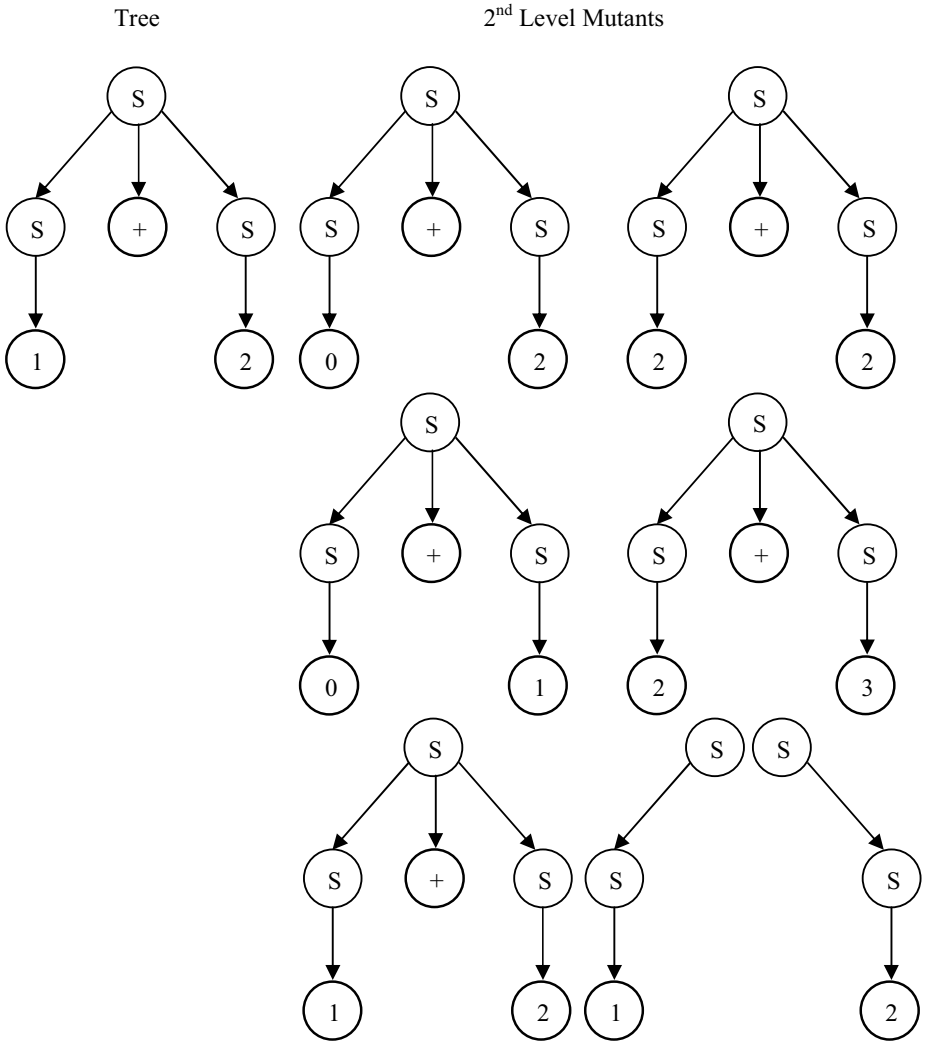


Figure 3: 2nd level mutants

into the definition of the transition relation H; in this case the corresponding grammar will be much more complex and system testing will be performed in one single step.

The following (simplified) example illustrates the above strategy for high-level testing of P systems.

Example 4. *Consider a 1-membrane P systems with 2 rules $r_1 : u_1 \longrightarrow v_1$, $r_2 : u_2 \longrightarrow v_2$. Then the transition of the Kripke structure representation of Π is given by the formulae:*

- $(u,v) \in H$, $u,v \in N^2_{Max}$, *if* $\exists n_1,n_2 \in N \cdot MaxParal(u,u_1,v_1,n_1,u_2,v_2,n_2) \wedge$ $Apply(u,v,u_1,v_1,n_1,u_2,c_2,n_2)$;
- $(u,Halt) \in H$, $u \in N^2_{Max}$, *if* $\neg\exists v \in N^2_{Max}, n_1,n_2 \in \boldsymbol{N} \cdot Apply(u,v,u_1,v_1,n_1,u_2,v_2,n_2)$;
- $(u,Crash) \in H$ *if* $\neg\exists v \in N^2_{Max} \cup \{Halt\} \cdot (u,v) \in H$;
- $(Halt,Halt) \in H$, $(Crash,Crash) \in H$;

Then such a system can be described by a context-free grammar $G = (V,T,P,S)$ where $V = \{S,S_1,S_2,U,V,U_1,V_1,U_2,V_2\}$; T contains (bounded) vectors from $\boldsymbol{N}^2$, the additional states $Halt$ and

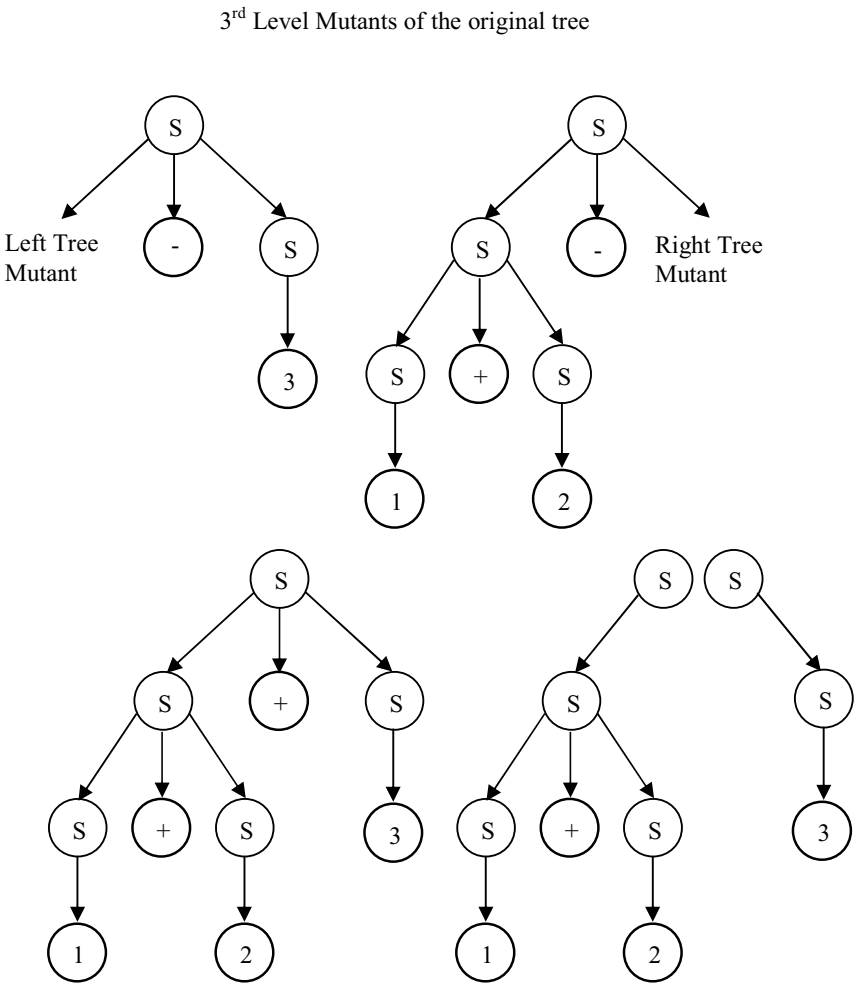


Figure 4: 3rd level mutants

Crash, predicates MaxParal and Apply, the "true" logical value, logical operators, quantifiers and other symbols, i.e.,
$T = N^2_{Max} \cup \{Halt, Crash, MaxParal, Apply, true, \wedge, , \vee, \neg, \exists, \forall, n_1, n_2, \cdot, (,)\}$.
The set of production rules consists of: $p_1 : S \longrightarrow \neg S$; $p_2 : S \longrightarrow S \wedge S$; $p_3 : S \longrightarrow S \vee S$; $p_4 : S \longrightarrow true$; $p_5 : S \longrightarrow \exists n_1 \cdot S_1$; $p_6 : S_1 \longrightarrow \exists n_2 \cdot S_2$; $p_7 : S_2 \longrightarrow S_2 \wedge S_2$; $p_8 : S_2 \longrightarrow Apply(U, V, U_1, V_1, n_1, U_2, V_2, n_2)$; $p_9 : S_2 \longrightarrow MaxParal(U, U_1, V_1, n_1, U_2, V_2, n_2)$; *rules that transform nonterminals* U, U_1, V_1, U_2, V_2 *into vectors from* $\boldsymbol{N}^2$.

The following mutants can be defined for the rules p_1 *to* p_7: $p'_1 : S \longrightarrow S$; $p'_2 : S \longrightarrow S \vee S$, $p''_2 : S \longrightarrow S$; $p'_3 : S \longrightarrow S \wedge S$, $p''_3 : S \longrightarrow S$; $p'_4 : S \longrightarrow \neg true$; $p'_5 : S \longrightarrow \forall n_1 \cdot S_1$; $p'_6 : S_1 \longrightarrow \forall n_2 \cdot S_2$. $p'_7 : S_1 \longrightarrow S \vee S_1$, $p''_7 : S_1 \longrightarrow S_1$. *For* p_8 *mutants can be defined by negating de predicate, changing parameters such that the obtained formula is syntactically correct, e.g., switch u and* u_1. *Similarly, mutants for* p_9 *are obtained by negating de predicate, changing parameters such that the obtained formula is syntactically correct. For the remaining rules mutants are generated by adding* `1` *to or subtracting* `1` *from each integer value.*

5 Conclusions

In many applications based on formal specification methods the test sets are generated directly from the formal models. The same applies to formal models based on grammars. However the approach presented in [11], although novel and with many practical consequences, lacks a rigorous method of defining the process of generating the mutants. In this paper a formal method is introduced to rigorously define operations with rules and subtrees of derivation trees for context-free grammar formalisms. This is then extended to P systems and some examples are provided to illustrate the approach. In this paper, the mutation operators are applied to the Kripke structure equivalent of the P system rather than to the P system itself. The advantage of this approach is that test values can be simply generated using a model checking tool (these are the counterexamples returned by the tool). Future work may investigate the application of the mutation operators directly to the P system and the associated test generation process.

Acknowledgment. This work is supported by the CNCSIS grant IDEI 643/2009 (EvoMT). The authors are grateful to reviewers for their comments.

Bibliography

[1] F. Bernardini, M. Gheorghe, J. J. Romero-Campero, N. Walkinshaw, A Hybrid Approach to Modelling Biological Systems, Workshop on Membrane Computing 2007, *Lecture Notes in Computer Science,* Vol. 4860, pp. 138–159, 2007.

[2] G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez (eds.), *Applications od Membrane Computing,* Springer, 2006.

[3] Z. Dang, O. H. Ibarra, C. Li, G. Xie, On the Decidability of Model-Checking for P Systems, *Journal of Automata, Languages and Combinatorics,* Vol. 11, pp. 279–298, 2006.

[4] M. Gheorghe, F. Ipate, On Testing P Systems, Workshop on Membrane Computing, *Lecture Notes in Computer Science,* Vol. 5391, pp. 204–216, 2008.

[5] M. Holcombe, F. Ipate, *Correct Systems: Building a Business Process Solution,* Springer, 1998.

[6] J. E. Hopcroft, R. Motwani, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation (2nd Edition),* Addison-Wesley, 2001.

[7] F. Ipate, M. Gheorghe, Testing Non-determinstic Stream X-machine Model and P Systems, *Electronic Notes in Theoretical Computer Science,* Vol. 227, pp. 113–126, 2009.

[8] F. Ipate, M. Gheorghe, Finite State based Testing of P Systems, *Natural Computing,* to appear, 2009.

[9] J. Offutt, A Practical System for Mutation Testing: Help for the Common Programmer, *International Test Conference,* pp. 824–830, 1994.

[10] Y.-S. Ma, J. Offutt, Y. R. Kwon, MuJava: An Automated Class Mutation System, *Software Testing, Verification and Reliability,* Vol. 15, pp. 97–133, 2005.

[11] J. Offutt, P. Ammann, G. Mason, L. (Ling) Liu, Mutation Testing implements Grammar-Based Testing, *Proceedings of the Second Workshop on Mutation Analysis,* 2006.

[12] Gh. Păun, Computing with Membranes, *Journal of Computer and System Sciences,* Vol. 61, pp. 108–143, 2000.

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

[14] http://en.wikipedia.org/wiki/Mutation_testing

[15] http://cs.gmu.edu/ offutt/mujava/

[16] http://ppage.psystems.eu

Florentin Ipate was born on 4th December 1967 in Constanta. FI holds a PhD and MSc degrees with the University of Sheffield and a BSc with Politehnica University of Bucharest, all in Computer Science. He is now a professor of Computer Science and PhD supervisor with the University of Pitesti. He has been awarded In Hoc Signo Vinces Prize for research and publications, by the National Research Council for Higher Education, Romania, 2002 and COPYRO Publishing Prize for Computer Science, Romania, 2000. FI's research interests are in specification and model based testing, formal specification languages for software systems, agile modelling and testing, modelling and testing biology-inspired computing systems. His main research results have been published in a research monograph with Springer and in high profile journals.

Marian Gheorghe was born on 2nd February 1953 in Bucharest. MG holds a PhD and a BSc with the University of Bucharest. He is now Senior Lecturer with the University of Sheffield and head of the Verification and Testing Group. MG's research interests are in formal computational models, verification and testing, modelling biological systems, agent technologies, artificial life, empirical software engineering. He has published in important international journals and is featured in the main computer science publications database, DBLP, with around 60 items.

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 263-272

An Algorithm for Initial Fluxes of Metabolic P Systems

Roberto Pagliarini, Giuditta Franco, Vincenzo Manca

University of Verona, Italy
Computer Science Department
Strada Le Grazie 15, 37134 Verona, Italy
E-mail: {roberto.pagliarini, giuditta.franco, vincenzo.manca}@univr.it

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: A central issue in systems biology is the study of efficient methods inferring fluxes of biological reactions by starting from experimental data. Among the different techniques proposed in the last years, the theory of *Metabolic P systems*, which is based on the *Log-Gain* principle, proved to be helpful for deducing biological fluxes from temporal series of observed dynamics. According to this approach, the algebraic systems provided by the Log-Gain principle determine the reaction fluxes underlying a system dynamics when initial fluxes are known. Here we propose a heuristic algorithm for estimating the initial fluxes, that is tested in two case studies.

Keywords: Biological modeling, P systems, MP systems, Metabolic flux estimation, Heuristic algorithms.

1 Introduction

In the last years, the problem of reverse-engineering of biological phenomena from experimental data has spurred increasing interest in scientific communities. For these reasons, many computational models inspired from biology have been proposed. Among these models, the *Metabolic P systems* [11, 12], shortly *MP systems*, proved to be relevant in the analysis of dynamics of biochemical processes, that is, structures where matter of different type is transformed by reactions. By means of MP systems models of several interesting phenomena were provided, among which we mention: the Lotka-Volterra dynamics [2, 3, 15], a Susceptible-Infected-Recovered epidemic [2], the Leukocyte Selective Recruitment in the immune response [2], the Protein Kinase C Activation [3], the Mitotic Cycle [14], the Pseudomonas Quorum Sensing [4] and the Non-Photochemical Quenching phenomenon [16].

The importance of MP systems is their potential applicability to the reverse-engineering problem of biological phenomena. In fact, in the framework of MP systems, a theory called *Log-Gain* [10, 11, 12] has been introduced, specifically devoted to the deduction of reaction fluxes, that is, the amount of reactants transformed by the reactions at any step of the system.

As we will show, a key point for achieving this task consists in the discovery of the fluxes associated to the passage of a metabolic system from the state at the initial observation instant to the next one. In this paper a heuristic algorithm is proposed for estimating the initial fluxes vector from few steps of observation. In few words, the algorithm first roughly computes the initial fluxes by assuming they have a form recalling the mass action principle, and then solves a system of equations to deduce the corresponding fluxes at the next step. From these values, the algorithm evaluates how much of each substance is necessary to activate the first evolution step, and finally the actual initial fluxes are computed by solving a minimization problem.

The present paper is organized as follows. Section 2 is devoted to the definition of Metabolic P Systems, while in Section 3 Log-Gain theory is briefly recalled. In Section 4 we describe the algorithm

which solves the initial fluxes problem. Section 5 reports the simulations of a couple of systems obtained by starting with initial fluxes computed by our algorithm. Further remarks and some directions for future research 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 [13]), based on *P systems* [5, 18, 19], 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 equation. In an MP system the variation of the whole system is considered in a macroscopic time interval. In this manner, the evolution law of the system includes the knowledge of the contribution of each reaction in the evolution from one state to the next one. Therefore, dynamics is given at discrete steps, and in 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*, and it defines the transformations of object populations, rather than single objects, according to a suitable generalization of chemical laws [11].

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

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

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

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 $\alpha_r \to \beta_r$, 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 $r = (\mathbb{A}_{x,r} \mid x \in X)$ of $\mathbb{R}^n$. 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. This set represents entities which affect the dynamics but are not transformed by reactions;
- $H = \{h_v \mid v \in V\}$ is a set of parameters evolution functions. The function $h_v : \mathbb{N} \to \mathbb{R}$ states the value of parameter v, and $H[i] = (h_v(i) \mid v \in V)$;
- $\Phi = \{\varphi_r \mid r \in R\}$ is the set of flux regulation maps, where, for each $r \in R$, $\varphi_r : \mathbb{R}^{n+k} \to \mathbb{R}$. Let $q \in \mathbb{R}^n$ be the vector of substance values and $s \in \mathbb{R}^k$ be the vector of parameter 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), constituted by the state q and by the parameters state 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 to some measure units).

- τ is the temporal interval between two consecutive observation steps;

Let $X[i] = (x_1[i], x_2[i], \ldots, x_n[i])$, for each $i \in \mathbb{N}$, be the vector of substances values at the step i, and let $X[0]$ be the initial values of substances. The dynamics of an MP 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), which allows us to obtain the vector $X[i+1]$ from vectors $X[i]$ and $U(X[i], X[i])$.

If in an MP system the elements ν, μ, and τ are omitted, then the result is called *MP grammar*. It is a multiset rewrite grammar where rules are regulated by specific functions. Such a grammar is completely specified by: *i)* reactions, *ii)* flux regulation functions, *iii)* parameter evolution functions, *iv)* substances, which are the elements occurring in the reactions, and their initial values and *v)* parameters, which are the arguments of flux regulation functions different from substances. Parameter evolution maps and/or initial values of substances may be omitted when only the MP grammar structure is specified.

3 Log-Gain Theory: a brief recall

The starting point of the Log-Gain theory [10, 11, 12] for MP systems is the *Allometry Law* [1, 7], which has many possible formulations [10], but, in the case here discussed, it can be expressed in a simple way. Namely, a proportion can be assumed, at each step, between the relative variations of the flux of a reaction and the sum of relative variations of its reactants, with a possible gap, called *offset*.

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

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

Assuming to know the vectors $X[i]$ and $X[i+1]$, the equation (1) can be rewritten in the following form, which we call $ADA[i]$ (Avogadro and Dalton Action [12]):

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

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 assumption is not restrictive. In fact, if it does not hold, the rows which are linearly dependent on other rows can be removed, by keeping the notations $\mathbb{A}$, $X[i+1]$ and $X[i]$ for the stoichiometric matrix and the vectors of concentration of substances, respectively. We assume thus that $\mathbb{A}$ has maximum rank, which we newly call n. Then there exist n linearly independent reactions of R, and we call R_0 such a subset of reactions. From a metabolic point of view, this means that fluxes of each reaction of R can be obtained as linear combination of fluxes of the reactions of R_0.

Formally, $ADA[i]$ is essentially the 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 a system comprised of both the equations in $ADA[i]$ and further equations, dictated by the following Log-Gain principle, to state the reaction regulation level, as we will see by formula (6).

Indeed, since the number of reactions is realistically assumed greater than the number of substances, then system (3) has more than one solution. Therefore, fluxes cannot be univocally deduced by means of $ADA[i]$. The Log-Gain principle allows us to add more equations in order to get a univocally solvable system which could provide the flux vector.

The two following definitions state the Log-Gain principle. For the detailed motivations of this principle we refer to papers on MP systems theory [10, 11, 12]. Further developments providing theoretical and experimental evidences of this principle will be matter of forthcoming papers.

Definition 2 (Discrete Log-Gain)**.** Let $(z[i]\,|i \in \mathbb{N})$ be a real valued sequence. Then, the discrete log-gain of z, for each step i, 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 $U[i]$ be the vector of fluxes at step i, for $i \geq 0$, and let $R_0 \subset R$ be a set of n linearly independent vectors of $\mathbb{R}^n$. Then, the Log-Gain regulation can be expressed in terms of matrix and vector operations:

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

where:

- $\mathbb{B} = (p_{r,x}\,|r \in R, x \in X)$ where $p_{r,x} \in \{0,1\}$ with $p_{r,x} = 1$ if x is a reactant of r and $p_{r,x} = 0$ otherwise;
- $Lg(X[i]) = (Lg(x[i])\,|x \in X)$ is the column vector of log-gains of substances;
- $C = (c_r\,|r \in R)$, where $c_r = 1$ if $r \in R_0$, while $c_r = 0$;
- $P[i+1]$ is a column vector of values associated with the reactions and called (Log-Gain) offsets at step $i+1$;
- $\times$ denotes the usual matrix product;
- $+, -, /, \otimes$ denote the component-wise sum, subtraction, division and product of vectors.

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 $OLGA[i]$, where the number of variables (reported in bold font) is equal to the number of equations:

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

Given the vector $Lg(X[i])$, for $i = 0,1,\ldots,l$, where $l \in \mathbb{N}$, it is possible to prove that $OLGA[i]$, for $i = 0,1,\ldots,l-1$, univocally provides $U[i]$ for $i = 1,2,\ldots,l-1$.

4 An algorithm to estimate initial metabolic fluxes

The iteration of the OLGA module, introduced in the previous section in order to deduce the fluxes of reactions, assumes the knowledge of the initial values of fluxes. This leads to the formulation of the following problem.

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

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

where $\approxeq$ means that we are searching for the vector $U[0]$ providing the minimum value of the stoichiometric error, defined as ($\|\cdot\|$ represents the Euclidean norm)

$$\|\mathbb{A} \times U[0] - (X[1]-X[0])\|.$$

The algorithm given below solves the Initial Fluxes Problem by using the knowledge about the dynamics in the first evolution steps in order to evaluate the amount of each substance which is necessary to activate the first evolution step.

4.1 The proposed algorithm

Our algorithm consists of three phases, some of which include different computational steps. The first phase consists in the approximation of initial fluxes by assuming that fluxes are proportional to the reactant quantities product. In the second phase an OLGA module is employed to approximate the amount of substances which needs as a fuel for the first evolution step. In the third phase an optimization problem is solved, which is based on the ADA system (3). The details of the algorithm work-flow are described in the following.

Phase 1. The goal here is to roughly evaluate the initial reaction fluxes by assuming that they are proportional to the reactants for certain initial evolution steps i. This could appear restrictive, but at this stage we require only an initial approximation. Therefore, at a given step i, for all $r \in R$, we set:

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

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 $y_r[i] = 1$, and we set

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

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

For any $x \in X$, let us consider the following system, called *Local-Stoichiometric Module* at the step i, where $\mathbb{A}$ is the stoichiometric matrix:

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

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), in at most $m-n$ steps either we obtain a square linear system of dimension m having maximum rank or the algorithm ends without an output. In fact, under the assumption that the rank of Local-Stoichiometric Module is n (that is, the number of equations) and that the number of variables is m, with $n < m$, then the system is completely determined if we add other $m-n$ equations. Assuming to gain at least one new significant equation at each step i, then in at most $m-n$ steps we obtain a system of $(m-n)n+n$ equations with m variables and rank equals to m. In this way, we can obtain a square linear system having unique solution.

In the example reported in Table 1, we have a Local-Stoichiometric Module of 3 equations having rank 3 which initially has 4 variables. At the second iteration of this module, starting from the step 0, we

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 corresponding flux regulation maps of the Local-Stoichiometric Module.

get other 3 equations finally giving the following system:

$$\begin{aligned}
\boldsymbol{a}[1]-\boldsymbol{a}[0] &= -\boldsymbol{k}_1\boldsymbol{a}[0]+\boldsymbol{k}_2\boldsymbol{b}[0]+\boldsymbol{k}_3\boldsymbol{c}[0] \\
b[1]-b[0] &= k_1a[0]-k_2b[0]+k_3c[0] \\
\boldsymbol{c}[1]-\boldsymbol{c}[0] &= \boldsymbol{k}_1\boldsymbol{a}[0]-\boldsymbol{k}_3\boldsymbol{c}[0]+\boldsymbol{k}_4\boldsymbol{c}[0] \\
a[2]-a[1] &= -k_1a[1]+k_2b[1]+k_3c[1] \\
\boldsymbol{b}[2]-\boldsymbol{b}[1] &= \boldsymbol{k}_1\boldsymbol{a}[1]-\boldsymbol{k}_2\boldsymbol{b}[1]+\boldsymbol{k}_3\boldsymbol{c}[1] \\
\boldsymbol{c}[2]-\boldsymbol{c}[1] &= \boldsymbol{k}_1\boldsymbol{a}[1]-\boldsymbol{k}_3\boldsymbol{c}[1]+\boldsymbol{k}_4\boldsymbol{c}[1]
\end{aligned}$$

where $a[0]=100$, $b[0]=100$, $c[0]=1$, $a[1]=101.87$, $b[1]=96.17$, and $c[1]=2.85$. This system has rank 4, and 4 linearly independent equations are reported in bold font. Thus, we can obtain a system of equations having unique solution. In general, if we start with the Local-Stoichiometric Module at the step 0 then we can compute the vector $\hat{U}[0]=(\hat{u}_r[0]\,|r\in R)$ by applying the Local-Stoichiometric module for a suitable number of steps. The algorithm stops with no output if after $m-n$ iterations of the above technique, the number of equations linearly independent is less than m.

Phase 2. The aim of this step is to estimate the amount of substance necessary to start the first system evolution step. We describe this step along with two sub-phases.

In the first sub-phase we solve *OLGA*[0] module, with $U[0]=\hat{U}[0]$, where $\hat{U}[0]$ is the vector of fluxes computed in the previous step. Let us call $U^*=(u_r^*\mid r\in R)$ the solution of this system. However, if some elements of this vector have a negative value, then we choose a different set of n linearly independent reactions in *OLGA* and newly apply the above procedure. The algorithm stops with no answer if a positive solution is not found after a number of attempts equal to the number of such different sets. However, general methods are under investigation which systematically and efficiently search for an unique positive solution U^*.

In the second sub-phase we compute, for each $x\in X$, the amount of substance $\bar{x}$ occurring for the application of the reactions in the first evolution step. If $\mathbb{A}^-$ is the *activation matrix* defined by $\mathbb{A}^-_{x,r}=|\alpha_r|_x$, for $x\in X$, $r\in R$, then the searched values are obtained by computing the vector $\bar{X}=\mathbb{A}^-\times U^*$.

Phase 3. In the last step we obtain the actual vector of fluxes U° by solving a norm minimization problem [9] such that U° provides the minimum of the following (Euclidean) norm

$$\|\mathbb{A}\times\xi-(X[2]-X[1])\| \tag{10}$$

over all the positive vectors $\xi=(\xi_r\mid r\in R)$ of $\mathbb{R}^m$ such that

$$\mathbb{A}^-\times\xi=\bar{X}, \tag{11}$$

where $\bar{X}$ is the vector computed at the previous step.

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 [8, 20, 21].

5.1 A synthetic metabolic system

Let us consider the synthetic non-cooperative metabolic system without parameters called Sirius [11] and given by Table 2. Firstly, we compute $U[1]=(\varphi_1(X[1],\varphi_2(X[2]),\ldots,\varphi_5(X[1]))$. Then, we use our algorithm to approximate the vector of fluxes U°. The two vectors are essentially the same.

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: The Sirius MP grammar.

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 system represents a well-known example of biochemical oscillatory phenomenon, in fact this is the first evidence of a chemical 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 [17] proposed a simplified formulation of the dynamics of the BZ reaction, called *Brusselator*, whose oscillating behaviour is represented by only two substances, x and y respectively, and it is governed by the following system of differential equations:

$$\begin{aligned} \frac{dx}{dt} &= k_1 - k_2 x + k_3 x^2 y - k_4 x \\ \frac{dy}{dt} &= k_2 x - k_3 x^2 y \end{aligned} \tag{12}$$

where $k_1 = 100, k_2 = 3, k_3 = 10^{-4}$ and $k_4 = 1$ represent constant rates. We use the oscillatory dynamics obtained by solving the system (12), with initial conditions $x = 1$ and $y = 10$, as experimental data on which testing our algorithm. MP formulation of the Brusselator is expressed by the set of rewriting rules reported in Table 3, where, according to the literature, the fluxes of each rule r depend on the concentrations of the reactants of r. 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 mapped into suitable stoichiometry by following the strategy described in [6].

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

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 deduced by OLGA module (Figure 1), starting from the initial fluxes inferred by means of our algorithm.

6 Conclusions

The study of efficient methods for defining MP systems from experimental data is of crucial importance for systematic applications of MP systems to complex dynamics. An essential component of

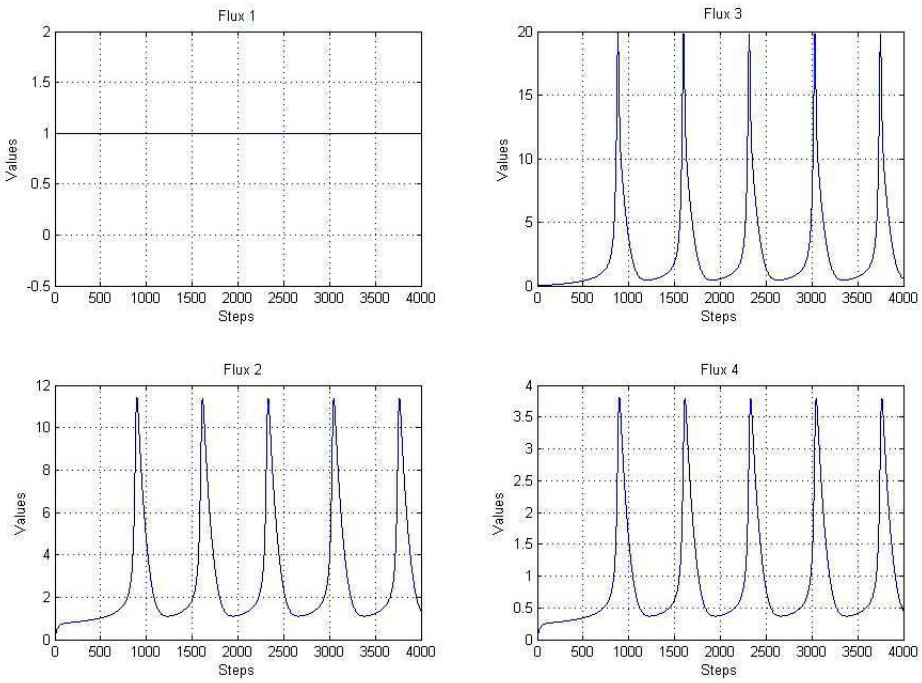


Figure 1: The BZ reaction fluxes calculated by solving the system (6) with initial vector of fluxes inferred by our algorithm.

the regulation level of an MP system can be deduced by applying the Log-Gain theory to data that can be collected from observations of the system. A crucial task to perform in this context is the reliable determination of the initial vector of fluxes.

In this paper we have devised an algorithm to infer the initial reaction fluxes of a biological network. The proposed algorithm has been validated on test cases of a synthetic metabolic oscillator and the Brusselator phenomenon. Future investigations will be developed with the aim *i)* to develop the computational features of this algorithm and *ii)* to show the applicability of our method to complex biological cases.

Bibliography

[1] L. von Bertalanffy. *General Systems Theory: Foundations, Developments, Applications*. George Braziller Inc, New York, NY, 1967.

[2] L. Bianco, F. Fontana, G. Franco, and V. Manca. P systems for biological dynamics. In *[5]*, 81–126, 2006.

[3] L. Bianco, F. Fontana, and V. Manca. P systems with reaction maps. *International Journal of Foundations of Computer Science*, 17(1):27–48, February 2006.

[4] L. Bianco, D. Pescini, P. Siepmann, N. Krasnogor, F.J. Romero-Campero, and M. Gheorghe. Towards a P Systems Pseudomonas Quorum Sensing Model. *Lecture Notes in Computer Science*, 4361:197–214, 2007.

[5] G. Ciobanu, M. J. Pérez-Jiménez, and G. Păun (Eds.). *Applications of Membrane Computing (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] F. Fontana and V. Manca. Discrete solution to differential equations by metabolic P systems. *Theoretical Computer Science*, 372:165–182, 2007.

[7] J. S. Huxley. *Problems of Relative Growth*. 2nd Ed., Dover, New York, 1972.

[8] D. S. Jones and B. D. Sleeman. *Differential Equations and Mathematical Biology*. Chapman & Hall/CRC, February 2003.

[9] D. G. Luenberger. *Optimization by Vector Space Methods*, John Wiley & Sons, Inc., 1969.

[10] V. Manca. Log-Gain Principles for Metabolic P Systems, In A. Condon et al. (Eds.), *Algorithmic Bioprocesses*, CHAPTER 28, *Natural Computing Series*, Springer-Verlag, Berlin Heidelberg, 2009.

[11] V. Manca. The Metabolic Algorithm for P systems: Principles and Applications. *Theoretical Computer Science*, 404:142–157, 2008.

[12] 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.

[13] 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.

[14] V. Manca and L. Bianco. Biological networks in metabolic P systems. *BioSystems*, 91(3):489–498, 2008.

[15] V. Manca, L. Bianco, and F. Fontana. Evolution and oscillation in P systems: Applications to biological phenomena. *Lecture Notes in Computer Science*, 3365:63–84, 2005.

[16] V. Manca, R. Pagliarini, and S. Zorzan. A photosynthetic process modelled by a metabolic P system. *Natural Computing*, 2009. DOI 10.1007/s11047-008-9104-x.

[17] G. Nicolis and I. Prigogine. *Exploring Complexity. An Introduction*. Freeman ans Company, San Francisco, CA, 1989.

[18] G. Păun. *Membrane Computing: An Introduction*. Springer, 2002.

[19] G. Păun and G. Rozenberg. A guide to membrane computing. *Theoretical Computer Science*, 287(1):73–100, September 2002.

[20] K. S. Scott. *Chemical Chaos*. Cambridge University Press, Cambridge, UK, 1991.

[21] A. M. Zhabotinsky. Proc. Acc. Sci, USRR. 157:392, 1964.

Roberto Pagliarini (July 23, 1981) earned his M. S. degree in Computer Science at University of Verona, where he is currently a PhD student in Computer Science. His research interests focus on System Biology, Molecular and Membrane Computing. He has collaborated with the biochemistry and vegetal physiology group at Biotechnological Department of Verona University, in order to investigate computational models for crucial events related to photosynthetic organisms. He is co-author of scientific papers on this subject.

Giuditta Franco (August 5, 1974) graduated in Mathematics at the University of Pisa and earned her PhD in Computer Science, with a dissertation titled "Biomolecular Computing — Combinatorial Algorithms and Laboratory Experiments", at University of Verona, where she is currently an assistant professor. Her research interests focus on discrete mathematics and computational models of biological systems, namely on DNA and Membrane Computing. She gave talks in several international workshops and she is co-author of scientific papers published by prestigious sectoral journals. She is an effective member of both the European Molecular Computing Consortium (EMCC) and the International Society for Nanoscale Science, Computing and Engineering (ISNSCE).

Vincenzo Manca (March 9, 1949) is a Full Professor at the Computer Science Department of the University of Verona, where he is also Chair of the Bioinformatics programme. He obtained his degrees from the University of Pisa. His research interests cover a wide class of topics from mathematical logic, discrete mathematics, and theoretical computer science to informational analysis and computational models of biological systems. At present, his investigation is focused on "Natural Computing" (in particular DNA Computing and Membrane Computing, Synthetic Computational Biology). He is author of more than 100 scientific publications, appearing in international journals and scientific series.

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 273-282

Spiking Neural P Systems with Anti-Spikes

Linqiang Pan, Gheorghe Păun

Linqiang Pan
Department of Control Science and Engineering
Huazhong University of Science and Technology
Wuhan 430074, Hubei, China
E-mail: lqpan@mail.hust.edu.cn, lqpan@us.es

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
E-mail: george.paun@imar.ro, gpaun@us.es

Received: April 20, 2009
Accepted: May 20, 2009

Abstract: 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.
Keywords: membrane computing, P system, spiking neural P system, computability

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). We do not claim having a clear biological counterpart of such issues, we only look for an elegant mathematical definition.

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

$$\Pi = (O, \sigma_1, \ldots, \sigma_m, syn, in, out), \text{ where:}$$

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \ldots, \sigma_m$ are *neurons*, of the form

$$\sigma_i = (n_i, R_i), 1 \leq i \leq m, \text{ where:}$$

   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 and that annihilation has priority over spiking and forgetting rules, 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 no priority case also remains to be investigated).

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.

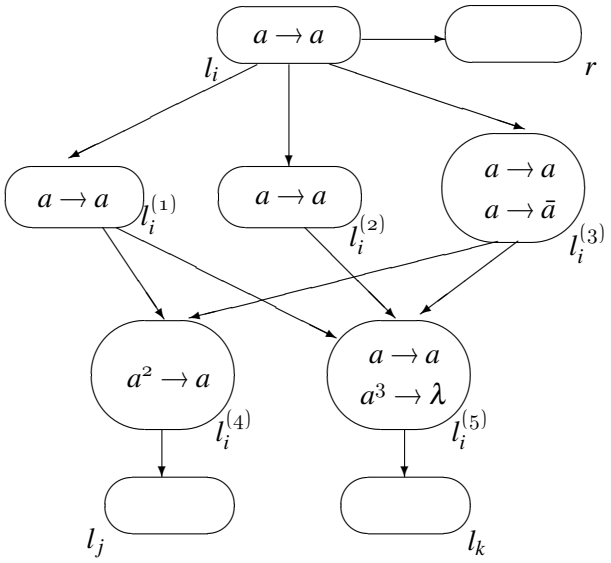


Figure 1: Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

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,

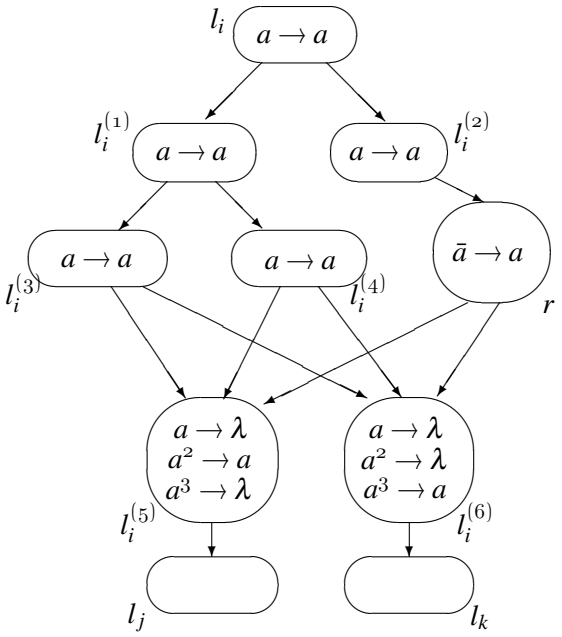


Figure 2: Module SUB, simulating $l_i : (\mathtt{SUB}(r), l_j, l_k)$

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.

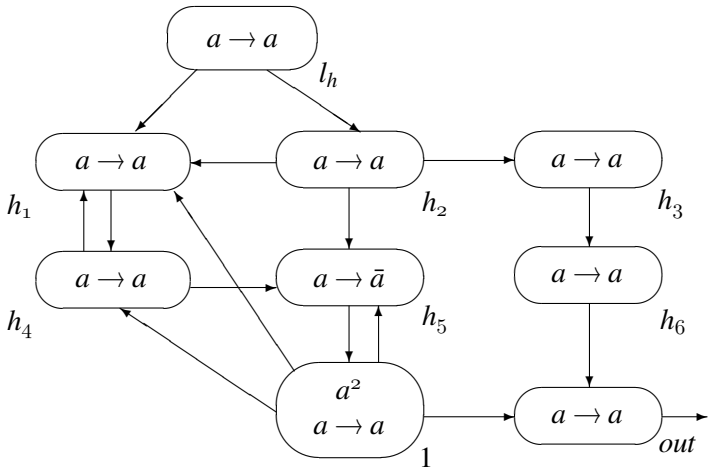


Figure 3: The FIN module

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 $l_i : (\mathtt{SUB}(r), l_j, l_k)$, 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 $a \to \bar{a}$?

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.

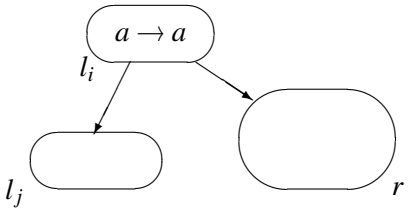


Figure 4: Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j)$

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:

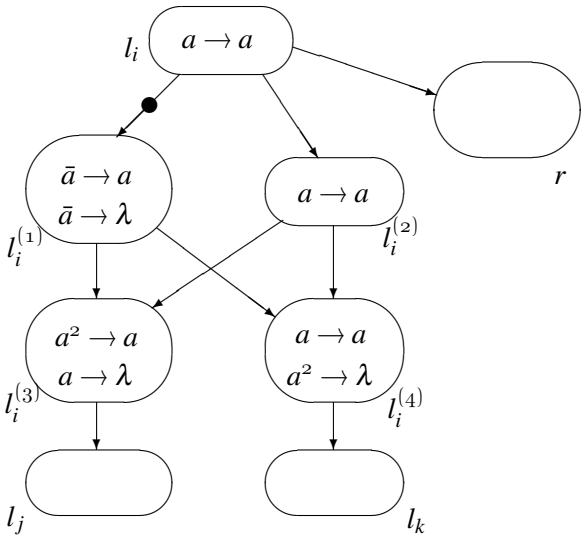


Figure 5: Module ADD, simulating $l_i : (\texttt{ADD}(r), l_j, l_k)$

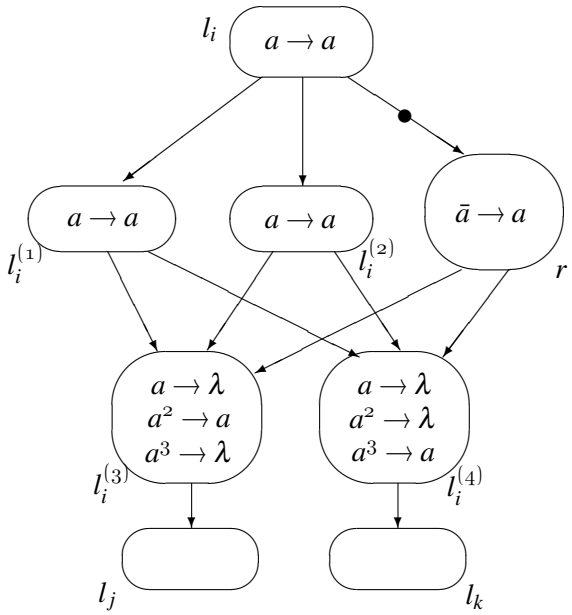


Figure 6: Module SUB, simulating $l_i : (\texttt{SUB}(r), l_j, l_k)$

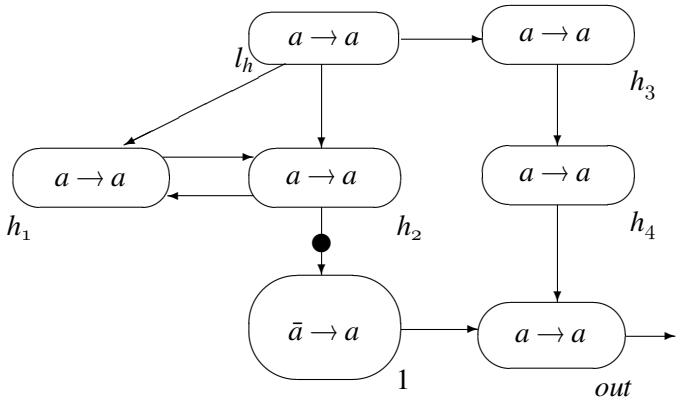


Figure 7: Module FIN

Theorem 2. $NRE = N_2S_aNP_s(prule_2)$.

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 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?). What about cases when the annihilation rule $a\bar{a} \to \lambda$ takes one time unit or/and it has no priority over other rules? By allowing the output neuron to also produce anti-spikes we can get a spike train over a three letter alphabet: no output, producing spikes, and producing anti-spikes, respectively. This can be an interesting way to produce languages (over three letters or perhaps over two, ignoring the no-output steps).

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. Useful remarks by A. Alhazov and M.A. Gutiérrez-Naranjo are gratefully acknowledged.

Bibliography

[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*, Vol. 372, pp. 196–217, 2007.

[4] M. Ionescu, Gh. Păun, T. Yokomori, Spiking Neural P Systems. *Fundamenta Informaticae*, Vol. 71, pp. 279–308, 2006.

[5] J.M. Mingo, Sleep-Awake Switch with Spiking Neural P Systems: A Basic Proposal and New Issues. *Proc. 7th Brainstorming Week on Membrane Computing*, Sevilla, 2009, vol. II, 59–72.

[6] L. Pan, Gh. Păun, New Normal Forms for Spiking Neural P Systems. *Proc. 7th Brainstorming Week on Membrane Computing*, Sevilla, 2009, vol. II, 127–138.

[7] Gh. Păun, G. Rozenberg, A. Salomaa, eds., *Handbook of Membrane Computing*. Oxford University Press, 2010 (in press).

[8] J. Wang, L. Pan, Excitatory and Inhibitory Spiking Neural P Systems. Submitted, 2007.

[9] The P Systems Website, `http://ppage.psystems.eu`.

Linqiang Pan was born in Zhejiang, China on November 22, 1972. He got PhD at Nanjing University in 2000. Since 2004, he is a professor at Huazhong University of Science and Technology, China. His main research fields are graph theory and membrane computing.

Gheorghe Păun graduated the Faculty of Mathematics, University of Bucharest, in 1974 and received his Ph.D. from the same university in 1977. From 1990 he is a senior researcher at the Institute of Mathematics of the Romanian Academy. He (repeatedly) visited numerous universities in Europe, Asia, and North America. His main research areas are formal language theory and its applications, computational linguistics, DNA computing, and membrane computing; this last research area was initiated by him, in 1998, and the respective models are now called P systems, see `http://ppage.psystems.eu`). He has published a large number of research papers, has lectured at many universities, and gave numerous invited talks at recognized international conferences. He has published eleven monographs in mathematics and computer science, has (co)edited over seventy collective volumes and special issues of journals, and also published many popular science books, books on recreational mathematics (games), and fiction books. He is a member of the editorial board of more than a dozen international journals and was/is involved in the program/steering/organizing committees for many recognized conferences and workshops. In 1997 he was elected a member of the Romanian Academy and from 2006 he is a member of Academia Europaea. He also got other honors, in Romania or abroad. He is an ISI Highly Cited Researcher (see `http://isihighlycited.com/`).

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 283-290

On Parallel Graph Rewriting Systems

Dragoş Sburlan

Ovidius University of Constantza, Romania
Faculty of Mathematics and Informatics
124 Mamaia Blvd., 900527 Constantza
E-mail: dsburlan@univ-ovidius.ro

Received: April 5, 2009
Accepted: May 30, 2009

Abstract:
In this paper we introduce a new theoretical paradigm, called PGR systems, which can be used to model in a discrete manner some natural phenomena occurring in-vivo/in-vitro environments. PGR systems make use of graphs to describe the spatial structure of space of individuals, while the system dynamics caused by the movement/interaction of individuals is captured by the parallel applications of some graph rewriting rules. In this frame, an illustrative example is studied and based on it, an eloquent comparison between the abstract rewriting machines and PGR systems is done. Several further ideas to overcome the global computational effort needed for simulations, but still maintaining the overall ability for modeling are finally proposed.
Keywords: parallel multiset processing, abstract rewriting systems, P systems

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 that 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 (see [4], [6], [7], [9]). 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 [10]). In general, P systems make use of multisets to represent the computational support. These multisets are placed inside 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 essentially simplifies the structure of the environment and of the individuals interaction (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 [4], [14], [2]). 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, employing 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 bio-system into the way the rules of the model are selected to be applied and executed step by step (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 [10] 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 reaction 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 $r_i : aA + bB \to cC + dD$, where $a, b, c, d \in \mathbb{N}$, $A, B, C, D \in O$, and a given multiset of objects $M \in O^*$, the probability of rule execution is defined as $Prob(r_i) = \frac{k_i M_A^a M_B^b}{R}$, where k_i is the rate constant (determined experimentally), M_A and M_B are the multiplicities of objects A and B in M, and R is a coefficient for normalizing the probabilities (i.e., $\sum_i Prob(r_i) = 1$). In a straightforward way 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 to the probabilities computed as above. Π is governed by a universal clock that marks equal time units.

A simple example, meaningful for our work, is presented bellow. We ran various tests using an ARM system Π where $O = \{A, B, C, D, X, F\}$, and the set of rules R is given below:

The initial configuration of Π was $w = A^{250} B^{250} C^5 D^5$ 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 and we aimed to test the ARM system ability to simulate such conditions.

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 rewriting rate of objects A, B, C, D, and X.

$$
\begin{array}{ll}
r_1 : AB \xrightarrow{k_1} X & r_8 : F \xrightarrow{k_8} F \\
r_2 : AC \xrightarrow{k_2} X & r_9 : A \xrightarrow{k_9} A \\
r_3 : BD \xrightarrow{k_3} X & r_{10} : B \xrightarrow{k_{10}} B \\
r_4 : CD \xrightarrow{k_4} F & r_{11} : C \xrightarrow{k_{11}} C \\
r_5 : FX \xrightarrow{k_5} FF & r_{12} : D \xrightarrow{k_{12}} D \\
r_6 : FA \xrightarrow{k_6} FF & r_{13} : X \xrightarrow{k_{13}} X \\
r_7 : FB \xrightarrow{k_7} FF &
\end{array}
$$

Since we have assumed the existence in the environment of a patch 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 can interact with each other but which cannot interact initially with the individuals A and B, then there exists a "significant" probability that the rule $CD \xrightarrow{k_4} F$ is executed (assuming that $k_4 > k_{11}$ and $k_4 > k_{12}$). 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 diagrams shown on the left 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 (as it is in the presented charts); if this is the case, then it would make almost impossible the precise identification of the rate constants associated to the rules.

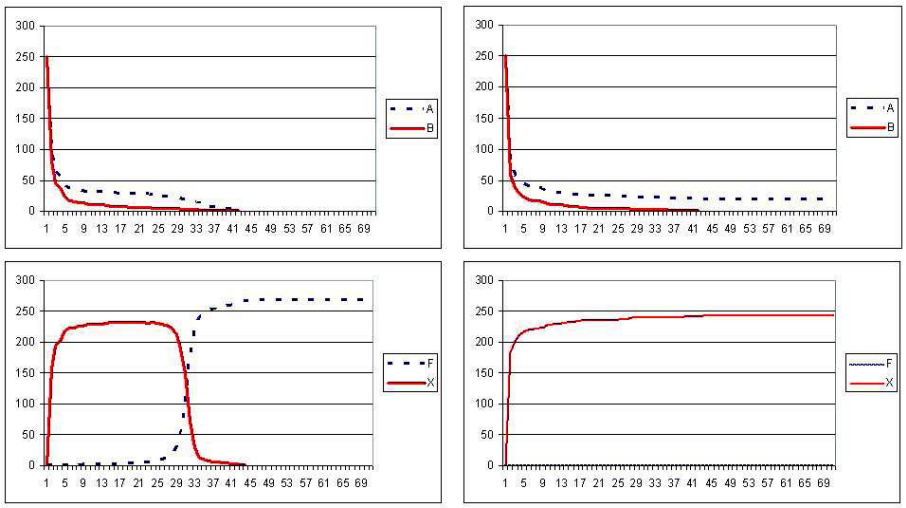


Figure 1: Two runs of system Π. The results are presented on columns and they show the different behaviors of the same system when some minor changes in the circumstances happen.

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 that are composed by interacting individuals of various species in a given environment.

Denote by C the finite 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 $r \in \mathbb{R}$, $r > 0$, be a positive constant.

Based on the above definitions 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 $1 \leq i \leq t$. For example, in Figure 2 it is presented a set of 4 individuals which initially lay on an environment represented as $\mathbb{R}^2$ and the way the corresponding labeled graph is constructed.

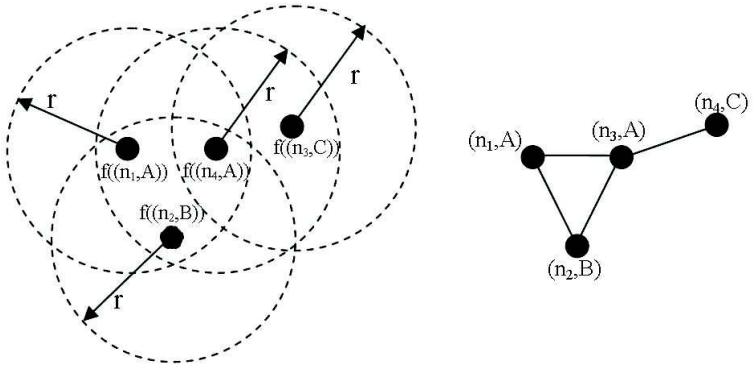


Figure 2: The construction of the labeled graph representing the initial computational support

Motivated by these facts we 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 the 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 assumes 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 one 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).

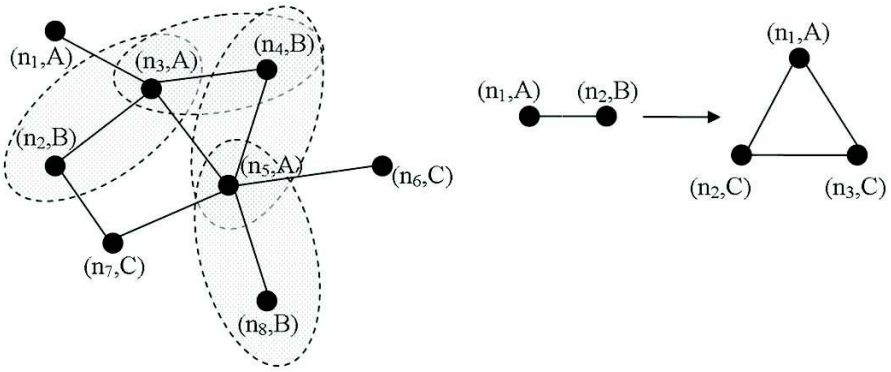


Figure 3: A graph $G=(V,E)$ denoting the computing support and a graph rewriting rule $r=(G_1,G_2)$. The sites where the rule r can be applied in G are explicitly figured. If $G_s=(V_s=\{(n_4,B),(n_5,A)\},E_s=\{((n_4,B),(n_5,A))\})$ then G_1 is label isomorphic with G_s. The neighborhood set of degree $k=1$ of G_s is $B_1=\{(n_3,A),(n_6,C),(n_7,C),(n_8,B)\}$.

The following steps are accomplished when a rule r is applied over G:

- 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 [1].

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 $t_r\in\mathbb{N}^+$ 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.

The problem of 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.

[1] If the studied individuals are particles that perform the Brownian motion, then at each application of a rule a random positive integer k can be generated and correspondingly a new neighborhood set can be defined; hence, the set B_k might be also useful to describe the random particle movement in the environment.

Remark 3.1. 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) belong to the same vicinity, hence the right hand side graph of any rule should be complete.

Remark 3.2. 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 as "similar" to the 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).

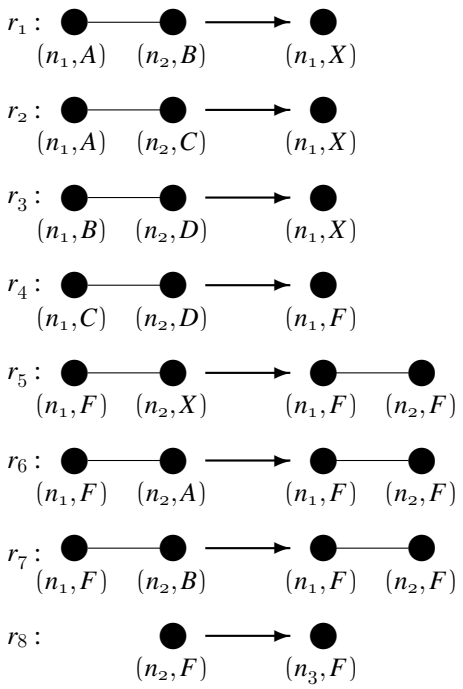
4 PGRS Simulator and a Test Case

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 $\overline{G}$ (a task useful when multiple applications of the same rule are performed). 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, hence in general unfeasible. 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 simulation is viable. Moreover, taking into account that the problem can be easily parallelized one can divide 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 obey some properties. First of all, a random graph G_0' was generated and this graph contains 500 nodes labeled only with A and B (for each test case, the apparition of these

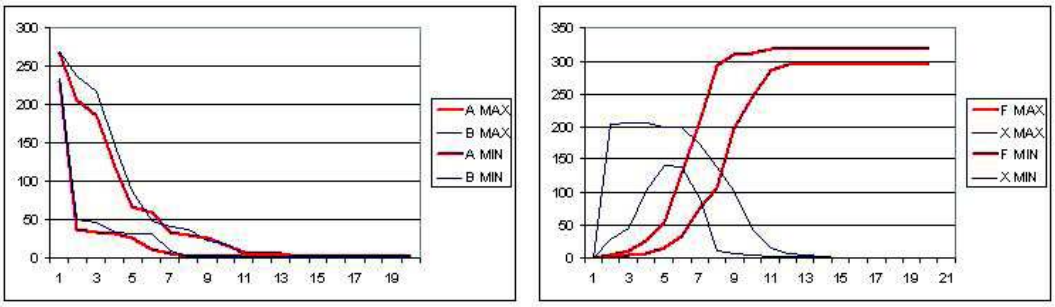


Figure 4: The results of 100 simulations of different GPR systems but having the same properties. The minimal and maximal obtained values are explicitly marked.

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 (also in this case, 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 there 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.

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).

Acknowledgments

The author gratefully acknowledges the support by CNCSIS-IDIEI grant, Romanian Ministry of Education, Research and Innovation, No. 804/2008.

Bibliography

[1] D. Besozzi, G. Mauri, D. Pescini, C. Zandron, Dynamical Probabilistic P Systems. *Int. J. Found. Comput. Sci.*, 17, 1 (2006), pp. 183–204.

[2] D. Besozzi, P. Cazzaniga, G. Mauri, D. Pescini, Modelling Metapopulations with Stochastic Membrane Systems, *BioSystems*, 91 (2008), pp. 499–514.

[3] B. Bollobas, *Modern Graph Theory*, Springer, 1991.

[4] M. Cavaliere, I.I. Ardelean, Modeling Biological Processes by Using a Probabilistic P System Software, *Natural Computing*, 2, 2 (2003), pp. 173–197.

[5] H. Ehrig, Introduction to the Algebraic Theory of Graph Grammars, *Lecture Notes in Computer Science* 73 (1979), pp. 1–69.

[6] P. Frisco, The Conformon-P System: A Molecular and Cell Biology-Inspired Computability Model, *Theoretical Computer Science*, 312, 2-3 (2004), pp. 295–319.

[7] P. Frisco, R.T. Gibson, A Simulator and an Evolution Program for Conformon-P Systems, *Proc. of the* 7^{th} *Int. Symp. on Symbolic and Numeric Algorithms for Scientific Computing*, 2005, pp. 427–430.

[8] D.T. Gillespie, Exact Simulation of Coupled Chemical Reactions, *J. Physical Chemistry*, 81 (1977), pp. 2340–2361.

[9] V. Manca, L. Bianco, Biological Networks in Metabolic P Systems. *Biosystems*, 91, 3 (2008), pp. 489–498.

[10] G. Păun, *Membrane Computing. An Introduction*, Springer, Berlin, 2002.

[11] D. Sburlan, Parallel Graph Rewriting Systems, *Proc. of the* 7^{th} *BWMC*, Seville, Spain, 2009, in print.

[12] Y. Suzuki, H. Tanaka, S. Tsumoto, Analysis of Cycles in Symbolic Chemical System Based on Abstract Rewriting System on Multisets, *Proceedings of International Conference on Artificial Life 5* (Alife 5), 1996, pp. 482–489.

[13] Y. Suzuki, J. Takabayashi, H. Tanaka, Investigation of Tritrophic System in Ecological Systems by Using an Artificial Chemistry, *J. Artif. Life Robot.*, 6 (2002), pp. 129–132.

[14] Y. Suzuki, H. Tanaka, Modelling p53 Signaling Pathways by Using Multiset Processing, *Applications of Membrane Computing* (G. Ciobanu, G. Păun, M. Pérez-Jiménez, Eds.), Springer, Berlin, 2006, pp. 203–214.

Dragoş Sburlan graduated the Faculty of Mathematics and Informatics, the Master Program *Computational Mathematics and Modern Computer Science Technologies* at the Ovidius University of Constanta, Romania. He defended his *European PhD* in computer science at the University of Seville, Spain. He followed several research stages (Leiden Institute of Advanced Computer Science, Holland and STAKI, Budapest, Hungry) and he is involved in several national and international research projects. Currently, he is Senior Lecturer at the Faculty of Mathematics and Informatics, Ovidius University of Constantza. His research interests include formal languages, theory of computing, natural computing, and software engineering.

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 291-300

P Systems Computing the Period of Irreducible Markov Chains

Mónica Cardona-Roca, M. Ángels Colomer-Cugat, Agustín Riscos-Núñez, Miquel Rius-Font

Mónica Cardona-Roca, M. Ángels Colomer-Cugat
Dpt. of Mathematics, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
E-mail: {mcardona, colomer}@matematica.udl.es

Agustín Riscos-Núñez
Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
E-mail: ariscosn@us.es

Miquel Rius-Font
Department of Applied Mathematics IV
Universitat Politécnica de Catalunya
Edifici C3, Despatx 016, Av. del Canal Olímpic, s/n,
08860 Castelldefels, Spain
E-mail: mrius@ma4.upc.edu

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: It is well known that any irreducible and aperiodic Markov chain has exactly one stationary distribution, and for any arbitrary initial distribution, the sequence of distributions at time n converges to the stationary distribution, that is, the Markov chain is approaching equilibrium as $n \to \infty$.
In this paper, a characterization of the aperiodicity in existential terms of some state is given. At the same time, a P system with external output is associated with any irreducible Markov chain. The designed system provides the aperiodicity of that Markov chain and spends a polynomial amount of resources with respect to the size of the input. A comparative analysis with respect to another known solution is described.
Keywords: Markov chain, P Sytems, Membrane Computing

1 Introduction

A discrete-time Markov chain is a stochastic process such that the past time is irrelevant to predict the future, given knowledge of the present time. That is, given the present time, the future does not depend on the past time: the result of each event depends only on the result of the previous event.

In order to study the evolution in time of a Markov chain as well as the existence of the stationary distribution, it is suitable to classify its states. This classification depends on the path structure of the chain.

One of the central issues in Markov Theory is the study of the asymptotic behavior of Markov chains. It is well known that for any irreducible and aperiodic Markov chain: (a) there exists at least one stationary distribution (that is, a probability distribution on the state space which is an invariant for the transition matrix associated with the chain), and (b) for any initial distribution, $\mu^{(0)}$ and for any stationary distribution π for the Markov chain, the sequence $(\mu^{(n)})_{n\in\mathbf{N}}$ converges to π in total variation as $n \to \infty$ (that is, the Markov chain is approaching equilibrium as $n \to \infty$).

In paper [2], a classification of states of a finite and homogeneous Markov chain is provided by using P systems. Moreover, the period was calculated for recurrent classes. The design of the P systems was inspired in properties used in classic algorithms that deal with the problem of the classification. Especially, this solution allows us to decide whether an irreducible Markov chain is aperiodic or not.

The main goal of this paper is to design a P system associated with an irreducible Markov chain which provides an answer to the aperiodicity of the chain. If the answer is negative, then the system provides the period of the chain. The solution presented is based on a characterization of the aperiodicity in existential terms of some state and a natural number, and it is *semi–uniform*, in the sense that for each Markov chain, a P system associated with it is constructed. Besides, the solution spends a polynomial amount of resources in the sense of the computational complexity theory in Membrane Computing.

The solution presented in the paper improves the solution obtained in [2], because less computational resources are used.

The paper is organized as follows. In the following section, we recall some basic notions and results that we use in the paper. In Section 3, a P system associated with an irreducible Markov chain is constructed in order to study the periodicity of that class. In Section 4, the solution presented is compared with another solution given in [2]. Finally some conclusions are presented.

2 Preliminaries

A discrete Markov chain is a sequence $\{X_t \mid t \in \mathbb{N}\}$ of random variables whose values are called *states*, that verifies the following property:

$$P(X_{t+1} = j/X_0 = i_0, X_1 = i_1, \ldots, X_t = i_t) = P(X_{t+1} = j/X_t = i_t).$$

Without loss of generality, we can suppose that the state space is the set of nonnegative integers.

The value of variable X_t is interpreted as the state of the process at instant t. In this paper we work with Markov chains having a finite state space $S = \{s_1, \ldots, s_k\}$.

A discrete Markov chain is characterized by the *transition probability*

$$p_{ij}(t) = P(X_t = s_j/X_{t-1} = s_i),\ \forall t \geq 1,$$

where $p_{ij}(t)$ provides the transition from state s_i to state s_j at time $t-1$.

The matrix of transition probabilities

$$P(t) = (p_{ij}(t))_{1 \leq i,j \leq k},$$

is a stochastic matrix, that is, is nonnegative for all t and the sum of each row is equal to 1, $\sum_{j=1}^{k} p_{ij}(t) = 1$.

We say that the chain is *time homogeneous* or *stationary* if $p_{ij}(t) = p_{ij}$ for each t and it verifies the Kolmogorov-Chapman equation:

$$p_{ij}^{(1)} = p_{ij}, \quad p_{ij}^{(2)} = \sum_{l=1}^{k} p_{il} p_{lj}, \quad \ldots, \quad p_{ij}^{(n)} = \sum_{l=1}^{k} p_{il} p_{lj}^{(n-1)},$$

where $p_{ij}^{(n)}$ is the transition probability of state s_i to state s_j at time n.

We denote the initial distribution by means of the vector

$$\mu^{(0)} = (\mu_1^{(0)}, \ldots, \mu_k^{(0)}) = (P(X_0 = s_1), P(X_0 = s_2), \ldots, P(X_0 = s_k)),$$

and the distribution of the Markov chain at time n is

$$\mu^{(n)} = (\mu_1^{(n)}, \ldots, \mu_k^{(n)}) = (P(X_n = s_1), P(X_n = s_2), \ldots, P(X_n = s_k)).$$

Then, $\mu^{(n)} = \mu^{(0)} \cdot P^{(n)}$, where $P = (p_{ij})$ is the transition matrix of the homogeneous Markov chain.

Next, we introduce some concepts and results related to the states of a homogeneous Markov chain.

We say that a state s_j *communicates* with another state s_i (and we denote it by $s_i \to s_j$), if there exists a natural number $n > 0$ such that $p_{ij}^{(n)} > 0$ (that is, if the chain has a positive probability of ever reaching s_j when we start from s_i. We say that the states s_i and s_j *intercommunicate* (and we denote it by $s_i \leftrightarrow s_j$) if $s_i \to s_j$ and $s_j \to s_i$.

In the finite state space $S = \{s_1, \ldots, s_k\}$ of a Markov chain, the relation $\leftrightarrow$ is an equivalence relation and we can consider the corresponding quotient set $\{s_1, \ldots, s_k\}/\leftrightarrow$ whose elements are the classes of equivalence by $\leftrightarrow$.

A Markov chain with state space $S = \{s_1, \ldots, s_k\}$ is said to be *irreducible* if there exists only equivalence class with respect to $\leftrightarrow$; that is, if for all $s_i, s_j \in E$ we have $s_i \leftrightarrow s_j$. Otherwise, the chain is said to be *reducible*.

We say that a state s_i is *recurrent* or *essential* if for each natural number m and for each state s_j verifying $p_{ij}^{(m)} > 0$ there exists a natural number n such that $p_{ji}^{(n)} > 0$. Otherwise, the state is said to be *transient*. A recurrent class is the equivalence class determined by a recurrent state.

It is easy to prove that from a recurrent state, only recurrent states belonging to the same class are reachable.

A *recurrence time* of s_i is a natural number $n > 0$ such that $p_{ii}^{(n)} > 0$. The *period* of a state s_i is defined as $d(i) = \text{g.c.d. } \{n \geq 1 \mid p_{ii}^{(n)} > 0\}$, that is, it is the greatest common divisor of the recurrence times associated with it. All states belonging to the same class have the same period.

Then, we can define the period of a class of a given Markov chain in a natural manner: it is the period of any state of the class (see [3] and [4] for more details).

Definition 1. A Markov chain is said to be aperiodic if all its states are aperiodic; that is, their periods are equal to 1. Otherwise, the chain is said to be periodic.

Next, we provide a method to compute the period of a recurrent class and a characterization of the periodicity of a class.

Theorem 2. *Let $A = \{s_1, \ldots, s_r\}$ be a recurrent class. The period of A is*

$$d = g.c.d. \{n \mid p_{ii}^{(n)} > 0;\ 1 \leq i, n \leq r\}.$$

Proof. As all states have the same period d, we have

$$d = d(1) = d(2) = \ldots = d(r) = g.c.d. \{n \geq 1 \mid p_{ii}^{(n)} > 0;\ 1 \leq i \leq r\}.$$

Let $d' = g.c.d.\{n|\ p_{ii}^{(n)} > 0;\ 1 \leq i, n \leq r\}$. Let us see that $d = d'$. For that, let $n > r$ be a time of recurrence associated with a state $s_i \in A$, that is, $p_{ii}^{(n)} > 0$. There exists a state s_{i_0} such that $p_{ii}^{(n)} \geq p_{ii_0}^{(n')} \cdot p_{i_0 i_0}^{(n_0)} \cdot p_{i_0 i}^{(n'')} > 0$, where $n = n' + n_0 + n''$. Thus, n_0 and $n' + n''$ are also times of recurrence. If $n_0 > r$ or $n' + n'' > r$, then we repeat the process until we obtain a decomposition

$$p_{ii}^{(n)} \geq p_{ii_0}^{(n')} \cdot p_{i_0 i_0}^{(n_0)} \cdot p_{i_1 i_1}^{(n_1)} \cdots p_{i_t i_t}^{(n_t)} \cdot p_{i_t i}^{(n'')} > 0,$$

with $1 \leq i_1, \ldots, i_t \leq r$, $n = n' + n_1 + \ldots + n_t + n''$ verifying $n' + n'' \leq r$ and $n_1, \ldots, n_t \leq r$.

Finally, let us notice that substituting $p_{ii}^{(n)}$, with $n > r$, by a *suitable* sequence of $p_{ii}^{(m)}$, with $m \leq r$, the g.c.d. is the same. □

Lemma 3. *Let $A = \{a_1, \cdots, a_r\}$ be a set of natural numbers. Let us suppose* g.c.d. $\{a_1, \cdots, a_r\} = 1$. *Let us denote by A^+ the set of all positive linear combinations*

$$\lambda_1 a_1 + \cdots + \lambda_r a_r, \quad with \quad \lambda_i \in Z^+, 1 \leq i \leq r.$$

Then, there exists a natural number N such that $n \in A^+$ for all $n \geq N$.

Proof. See, e.g., the appendix of [1] □

Next, we characterize the aperiodicity of a recurrent class of a finite Markov chain through the existence of a state s_j reachable from each state s_i.

Theorem 4. *Let $\{X_t \mid t \in \mathbb{N}\}$ be a Markov chain with state space $S = \{s_1, \ldots, s_k\}$ and transition matrix $P = (p_{ij})$.*

(1) If $\{X_t \mid t \in \mathbb{N}\}$ is aperiodic, then there exists a natural number N such that $p_{ii}^{(n)} > 0$, for all i $(1 \leq i \leq k)$ and all $n \geq N$.

(2) If $\{X_t \mid t \in \mathbb{N}\}$ is irreducible and aperiodic, then there exists a natural number M such that $p_{ij}^{(n)} > 0$, for all i, j $(1 \leq i, j \leq k)$ and all $n \geq M$.

Proof. See, e.g., Chapter 4 from [3] □

Theorem 5. *Let $A = \{s_1, \ldots, s_r\}$ be a recurrent class of a finite Markov chain. The following are equivalent:*

(1) Class A is aperiodic.

(2) There exists a state $s_j \in A$ and a natural number $m_0 \in \mathbb{N}$ such that $p_{ij}^{(m_0)} > 0$ for all state $s_i \in A$.

Proof. Let us suppose that class A is aperiodic. Then all states in A have the same period $d = 1$. From Theorem 4 there exists a natural number N such that $p_{ii}^{(n)} > 0$, for all i $(1 \leq i \leq r)$ and all $n \geq N$. Given j $(1 \leq j \leq r)$, we define $n_i(j) = min\{n \mid p_{ij}^{(n)} > 0\}$, for each $s_i \in A$, $n(j) = max\{n_1(j), \ldots, n_r(j)\}$, and $m_0 = N + n(j)$. Let us see that $p_{ij}^{(m_0)} > 0$, for each i $(1 \leq i \leq r)$. We have $p_{ij}^{(m_0)} \geq p_{ij}^{(n_i(j))} p_{jj}^{(m_0 - n_i(j))} > 0$ because of $p_{ij}^{(n_i(j))} > 0$ by definition of $n_i(j)$, and $p_{jj}^{(m_0 - n_i(j))} > 0$ by Theorem 4.

Conversely, let us suppose that there exists $m_0 \geq 1$ and a state $s_j \in A$ such that $\forall\ s_i \in A$ we have $p_{ij}^{(m_0)} > 0$. In particular, $p_{jj}^{(m_0)} > 0$ so m_0 is a recurrence time. On the one hand, if d is the period of the class, then m_0 is a multiple of d. On the other hand, if $s_i \in A$ is a state such that $p_{ji} > 0$, then $0 < p_{ij}^{(m_0)} p_{ji} \leq p_{ii}^{(m_0+1)}$, so $m_0 + 1$ is a multiple of d. Hence, $d = 1$. □

3 A P System Associated with an Irreducible Markov Chain

The goal of this paper is to study the aperiodicity of an irreducible Markov chain with state space $S = \{s_1, \ldots, s_k\}$, $k \geq 2$, by using P systems. In the affirmative case, the answer of the system is YES, on the contrary, the system sends an object encoding the period of the class to the environment.

3.1 The Design of the P System

Let $P_k = (p_{ij})_{1 \leq i,j \leq k}$ be a Boolean matrix associated with a class with a finite and homogeneous Markov chain of order k such that $p_{ij} = 1$ if the transition from s_i to s_j is possible, and $p_{ij} = 0$ otherwise; that is, P_k is the adjacency matrix of the directed graph associated with the recurrent class.

The solution presented in this paper is a *semi–uniform* one in the following sense: we give a family $\boldsymbol{\Pi} = \{\Pi(P_k) \mid k \in \boldsymbol{N}\}$, associating with P_k a P system with external output, such that (a) there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(P_k)$ from P_k; and (b) the output of the P system $\Pi(P_k)$ provides the classification of the recurrent class of the Markov chain as well as the period of the states.

We associate with the matrix P_k the P system of degree 4 with external output,

$$\Pi(P_k) = (\Gamma(P_k), \mu(P_k), \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4, R)$$

defined as follows:

- Working alphabet:

$$\begin{aligned}\Gamma(P_k) = \quad & \{s_{ij},\, t_{ij},\, \tau_{ij} \mid 1 \le i,j \le k\} \cup \{s_{ijr} \mid 1 \le i,j,r \le k\} \cup \{T_r \mid 0 \le r \le k\} \cup \\ & \{\beta_l \mid 0 \le l \le k-1\} \cup \{b_i, p_i \mid 1 \le i \le k\} \cup \{c_i,\, d_i \mid 0 \le i \le \alpha\} \cup \{yes, YES, \sigma\}\end{aligned}$$

  where $\alpha = 3k + \lceil \frac{k}{2} \rceil$.

- Membrane structure: $\mu(P_k) = [\,[\,[\,[\,]_1\,]_2\,]_3]_4$.

- Initial multisets:

$$\mathcal{M}_1 = \{t_{ij}^{p_{ij}} \mid 1 \le i,j \le k\} \cup \{\beta_0\};\ \mathcal{M}_2 = \{s_{ii} \mid 1 \le i \le k\}$$
$$\mathcal{M}_3 = \{b_i \mid 1 \le i \le k\} \cup \{d_0\};\ \mathcal{M}_4 = \emptyset$$

- The set R of evolution rules consists of the following rules:

$$r_1 = [t_{ij} \to \tau_{ij} t_{ij}^k]_1,\ \ 1 \le i,j \le k$$
$$r_2 = [\beta_i \to \beta_{i+1}]_1,\ \ 0 \le i \le k-2$$
$$r_3 = [\beta_{k-1}]_1 \to c_0^k$$
$$r_4 = [c_r s_{ij} \tau_{j1}^{p_{j1}} \dots \tau_{jk}^{p_{jk}}]_2 \to [s_{i1}^{p_{j1}} \dots s_{ik}^{p_{jk}} c_{r+1}^{\gamma_j}]_2 s_{i1r+1}^{p_{j1}} \dots s_{ikr+1}^{p_{jk}} T_{r+1}^{p_{ji}},$$
$$1 \le i,j \le k, \quad 0 \le r \le \alpha - 1, \gamma_j = \textstyle\sum_{l=1}^{k} p_{jl}$$
$$r_5 = [\sigma]_2 \to \sigma$$
$$r_6 = [s_{1jr} \dots s_{kjr}]_3 \to [\sigma]_2\, yes,\ \ 1 \le j \le k,\ 1 \le r \le \alpha$$
$$r_7 = [T_r b_r \to p_r]_3,\ \ 1 \le r \le k$$
$$r_8 = [p_i p_{i+l} \to p_i p_l]_3,\ \ 1 \le i \le k,\ 1 \le l \le k-i$$
$$r_9 = [p_i^2 \to p_i]_3,\ \ 1 \le i \le k$$
$$r_{10} = [d_i \to d_{i+1}]_3,\ \ 0 \le i \le \alpha - 1$$
$$r_{11} = [d_\alpha p_r]_3 \to p_r[\,]_3,\ \ 2 \le r \le k$$
$$r_{12} = [d_\alpha p_1]_3 \to yes[\,]_3$$
$$r_{13} = [yes]_4 \to YES[\,]_4$$
$$r_{14} = [p_r]_4 \to p_r[\,]_4,\ \ 1 \le r \le k$$

3.2 An Overview of Computations

Initially, membrane 1 contains objects t_{ij} that codify the elements p_{ij} of the Boolean matrix associated with the transition matrix of the Markov chain, together with the counter β_0. This counter allows us to dissolve membrane 1 at a certain instant. Membrane 2 contains initially objects s_{ii} that codify the states s_i of the chain. Membrane 3 contains objects b_i that will be used in order to avoid that repeated recurrence times smaller than or equal to k appear. The counter d in membrane 2 will be used to trigger the answer at the suitable instant.

The design of the P system $\Pi(P_k)$ implements a process that is structured by stages. The first one consists of k steps which allow the production of sufficiently many new copies τ_{ij} of objects t_{ij}. This is done by applying rules of type r_1 and r_2 in membrane 1 at $k-1$ first steps and applying at step k rule r_3 that dissolves membrane 1.

At the second stage, all paths between states with length at most k, as well as recurrence times smaller than or equal to k, are generated. This stage starts at step $k+1$ and it spends at most k steps. First, rules of type r_4 are applied producing objects s_{ijr} in membrane 3 that codify the existence of a path with length r from state s_i to state s_j, as well as the objects T_r codifying the existence of a recurrence time equal to r. Simultaneously, it is checked if there exists a state s_j and a natural number m_0 such that $p_{ij}^{(m_0)} > 0$, for all states s_i. In that case, an object σ is produced in membrane 2 and the system expels an object YES to the environment.

The third stage is only applied if an object YES has not been expelled to the environment. At this stage, the period of the class is computed and it takes $k + \lceil \frac{k}{2} \rceil$ steps. By applying rules of type r_7, objects p_r encoding recurrence times smaller than or equal to k, are obtained. Such recurrence times are different from each other. By applying rules of types r_8 and r_9, the greatest common divisor of these times is computed. If the period of the class is equal to 1, then the system sends an object YES to the environment, otherwise, the system expels an object p_n that encodes the period of the class to the environment.

4 Results and Discussions

In [2] a P system was constructed which allows us to classify the states of a Markov chain. Thus, that P system can be adapted to characterize the aperiodicity of such a chain. Specifically, if $P_k = (p_{ij})_{1 \leq i,j \leq k}$ is the Boolean matrix associated with the states of a recurrent class of a finite and homogeneous Markov chain of order k, then we define the system

$$\Pi'(P_k) = (\Gamma'(P_k), \mu'(P_k), \mathcal{M}'_1, \mathcal{M}'_2, \mathcal{M}'_3, \mathcal{M}'_4, R', \rho'),$$

as follows:

- Working alphabet:

$$\begin{aligned} \Gamma'(P_k) = \; & \{A,\, R_i,\, t_{ij} \mid 1 \leq i,j \leq k\} \cup \{c_r \mid 0 \leq r \leq 2k+2\} \cup \\ & \{t_{ijur} \mid 1 \leq i,j,u \leq k,\, 0 \leq r \leq k\} \cup \{\beta_i \mid 0 \leq i \leq \gamma+1\} \cup \\ & \{s_{ijr} \mid 1 \leq i,j \leq k,\, 0 \leq r \leq k\} \cup \{d_i \mid 0 \leq i \leq 3(k-1)\} \end{aligned}$$

  where $\gamma = 2k+4+\lceil \lg_2 k \rceil + \frac{(k-1)(k+2)}{2}$.

- Membrane structure: $\mu'(P_k) = [\,[\,[\,[\,]_4\,]_3\,]_2\,]_1$.

- Initial multisets:

$$\mathcal{M}'_1 = \emptyset;\ \mathcal{M}'_2 = \{\beta_0\};\ \mathcal{M}'_3 = \{c_0\};\ \mathcal{M}'_4 = \{s_{ii0} \quad t_{ij}^{p_{ij}(k-1)} \mid 1 \leq i,j \leq k\}.$$

- The set R of evolution rules consists of the following rules:

  - Rules in the skin membrane labeled by 1:
    $r_1 = \{d_p \to (R_p, out) \mid 1 < p \leq k\}$
    $r_2 = \{d_1 \to (A, out)\}$
  - Rules in the membrane labeled by 2:
    $r_3 = \{\beta_i \to \beta_{i+1} \mid 0 \leq i \leq \gamma\} \cup \{\beta_{\gamma+1} \to \lambda\}.$
    $r_4 = \{d_j^2 \to d_j \mid 1 \leq j \leq k\}$
    $r_5 = \{d_j d_{j+l} \to d_j d_l \mid 1 \leq j \leq k,\, 2 \leq j+l \leq k\}$

– Rules in the membrane labeled by 3:

$r_6 = \{t_{ijur} \to (t_{ij}s_{uj(r+1)}, in_4) \mid p_{ij} = 1, u \neq j, 1 \leq i, j, u \leq k, 0 \leq r < 3(k-1)\}$

$r_7 = \{t_{iju(3k-3)} \to (t_{ij}, in_4) \mid p_{ij} = 1, u \neq j, 1 \leq i, j, u \leq k\}$

$r_8 = \{t_{ijjr} \to (t_{ij}, in_4)\ d_{r+1} \mid p_{ij} = 1, 1 \leq i, j \leq k, 0 \leq r < 3(k-1)\}$

$r_9 = \{t_{ijj(3k-3)} \to (t_{ij}, in_4) \mid p_{ij} = 1, 1 \leq i, j \leq k\}$

$r_{10} = \{c_r \to c_{r+1} \mid 0 \leq r \leq 6(k-1)+1\} \cup \{c_{6(k-1)+2} \to \lambda\}$

– Rules in the membrane labeled by 4:

$r_{11} = \{s_{uir}t_{i1}^{p_{i1}} \ldots t_{ik}^{p_{ik}} \to (t_{i1ur}^{p_{i1}} \ldots t_{ikur}^{p_{ik}}, out) \mid 1 \leq u, i \leq k,\ 0 \leq r \leq 3(k-1)\}$.

- There is only a priority relation in the membrane labeled by 2: $\{r_4 > r_5\}$.

In order to study the efficiency of the P system $\Pi(P_k)$ constructed in this work, we will compare the results with those obtained by the P system $\Pi'(P_k)$ described above. For that purpose, a comparative analysis of the computational resources required in both P systems is given first. Secondly, an analysis of the times of execution obtained on designed simulators for both P systems with some case studies is presented.

4.1 Computational Resources Required

The resources required initially to construct the systems $\Pi(P_k)$ and $\Pi'(P_k)$, and the number of steps taken by the systems, are the following:

	$\Pi(P_k)$	$\Pi'(P_k)$
Size of the alphabet	$\Theta(k^3)$	$\Theta(k^4)$
Initial number of membranes	4	4
Sum of the sizes of initial multisets	$\Theta(k^2)$	$\Theta(k^4)$
Number of rules	$\Theta(k^3)$	$\Theta(k^4)$
Maximal length of a rule	$\Theta(k)$	$\Theta(k)$
Number of priority relations	0	$\Theta(k^2)$
Number of steps	$\Theta(k)$	$\Theta(k)$

In the previous table, let us notice that the amount of resources requested by $\Pi(P_k)$ is smaller than the ones requested by $\Pi'(P_k)$. Indeed, the size of the alphabet and the number of rules pass from power 3 to power 4, and the system $\Pi(P_k)$ has no priority relation. The number of steps is of the same asymptotic order.

4.2 Case Studies

We have realized a simulator for each system $\Pi(P_k)$ and $\Pi'(P_k)$. These simulators have been written in C++ language and they have been executed on a Pentium 4 computer with 512 Mb RAM and 3.20 GHz.

In both simulators objects t_{ij} have been represented by means of arrays of dimension 2; objects s_{ij} have been represented by vectors of dimension 2 and recurrent times have been represented by one-dimensional vectors.

The simulator of the system $\Pi(P_k)$ generates the trajectories with a length at most $3k + \lceil k/2 \rceil$ in a sequential way, keeping the times of recurrence smaller than or equal to k. If assertion (2) in Theorem 5 is fulfilled, the simulator halts displaying the time of execution and the aperiodicity of the Markov chain. Otherwise the simulator computes the g.c.d. of the recurrence times obtained where all of them are different.

Similarly, a simulator for the system $\Pi'(P_k)$ has been implemented. The main difference with respect to the previously mentioned one is that it can keep more than a copy of the times of recurrence. All trajectories of the Markov chain with a length smaller than or equal to $3(k-1)$ and their recurrence time are computed. Then the g.c.d. of these times is obtained.

When the Markov chain is aperiodic, the P system $\Pi(P_k)$ can finish before all trajectories with a length $3k+\lceil k/2 \rceil$ are computed. In case it is necessary to calculate the period, bearing in mind that all recurrence times are different, system $\Pi(P_k)$ is faster than $\Pi'(P_k)$ in computing the g.c.d. of these times.

When the Markov chain is periodic the length of the trajectories computed by $\Pi(P_k)$ are longer than those computed by $\Pi'(P_k)$. Nonetheless, in order to compute the period, recurrence times used in $\Pi(P_k)$ are all different.

The simulators designed have been executed on eight recurrent Markov chains with 100 states. Four of these Markov chains are periodic and the others are aperiodic. Table 1 shows the values equal to 1 of the adjacency matrix of the graph associated with the recurrent Markov chains. The execution times are described in Table 2.

Example			
1	$p_{i,i+1}=1$	$1 \le i \le 99$	
	$p_{100,1}=1$		
2	$p_{i,i+1}=1$	$1 \le i \le 99$	
	$p_{i,1}=1$	$1 \le i \le 100$	
3	$p_{10j+i,10j+i+1}=1$	$1 \le i \le 9$	$0 \le j \le 9$
	$p_{10j,10j-9}=1$	$1 \le j \le 10$	
	$p_{10j+1,10j+11}=1$	$0 \le j \le 8$	
	$p_{91,1}=1$		
4	$p_{10j+i,10j+i+1}=1$	$1 \le i \le 9$	$0 \le j \le 9$
	$p_{10j,10j-9}=1$	$1 \le j \le 10$	
	$p_{10j+1,10j+11}=1$	$0 \le j \le 8$	
	$p_{91,1}=1$		
	$p_{1,1}=1$		
5	$p_{10j+i,10j+i+1}=1$	$1 \le i \le 9$	$0 \le j \le 9$
	$p_{10j,10j-9}=1$	$1 \le j \le 10$	
	$p_{10j+1,10j+11}=1$	$0 \le j \le 8$	
	$p_{91,1}=1$		
	$p_{2,2}=1$		
6	$p_{5j+i,5j+i+1}=1$	$1 \le i \le 4$	$0 \le j \le 19$
	$p_{5j,5j-4}=1$	$1 \le j \le 20$	
	$p_{5j+1,5j+6}=1$	$0 \le j \le 18$	
	$p_{96,1}=1$		
7	$p_{i,i+1}=1$	$1 \le i \le 100$	
	$p_{i+1,i}=1$	$1 \le i \le 100$	
	$p_{1+3i,4+3i}=1$	$0 \le i \le 32$	
8	$p_{i,i+1}=1$	$1 \le i \le 100$	
	$p_{i+1,i}=1$	$1 \le i \le 100$	
	$p_{1+3i,4+3i}=1$	$0 \le i \le 32$	
	$p_{1,1}=1$		

Table 1. Adjacency values of the examples

5 Conclusions

Markov chains have applications in different fields such as physics, economics, biology, statistics, social sciences, etc. In these applications it is important to know whether the Markov chain associated with the process is convergent or not. When the Markov chain is aperiodic, the transition matrix converges and the process becomes stable. In other cases, the process does not reach an equilibrium.

In this work, a characterization of the aperiodicity of a Markov chain has been given in terms of the

Example	period	$\Pi'(P_k)$	$\Pi(P_k)$
1	100	0	0
2	1	146	0
3	10	0	0
4	1	122	35
5	1	1	2
6	5	11	20
7	2	381	169
8	1	1101	104

Table 2. Observed run times

existence of a state reachable from any other state. Based on this property, a computational P system has been constructed that allows us to know whether the Markov chain is aperiodic and calculate its period if not.

In [2], every finite and homogeneous Markov chain has associated a P system that provides a classification of its recurrent classes. That P system can be adapted to study the aperiodicity of a Markov chain and then its period can be calculated. The solution presented in this work improves the solution derived from the P system described in [2]. For that purpose, simulators have been constructed for these P systems and the respective times of execution on eight examples have been analyzed.

For the computational study of the aperiodicity of a Markov chain it would be interesting to design new P systems that incorporate additional features such as electrical charges, active membranes, etc. and that improve quantitatively the amount of computational resources used.

Acknowledgement

The authors wish to thank Mario J. Pérez-Jiménez for his advices, suggestions and constant help. Among the numerous virtues of this excellent professor, we would like to highlight his enviable capacity of work and his great human quality. As it is usually said, behind a great man there is a great woman and so we cannot end theses line without expressing our gratitude to Queta for her immense patience with all of us.

The third author acknowledges the support of the project TIN2006–13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

Bibliography

[1] P. Brémaud: *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer, New York, 1998.

[2] M. Cardona, M.A. Colomer, M.J. Pérez–Jiménez, A. Zaragoza: Classifying states of a finite Markov chains with membrane computing. *Lecture Notes in Computer Science. Springer*,Vol 4361, pp. 266-278, 2006.

[3] O. Häggstöm: *Finite Markov Chains and Algorithmic Applications*. London Mathematical Society, Cambridge University Press, Cambridge, 2003.

[4] R. Nelson: *Probability, Stochastic Processes, and Queing Theory*. Springer, New York, 1995.

Mónica Cardona-Roca received his degree in Mathematics in 2000, from the Barcelona University. He is associate professor at the Department of Mathematics of the University of Lleida where he has been teaching for eight years. Here main research areas are natural computing and membrane computing. She has co-authored two books in mathematics. She has co-authored 5 papers on international journals in natural computing. She has co-authored 3 papers in statistics published in proceedings of Spanish conferences.

M. Ángels Colomer-Cugat received his degree in Engineer Agronomist from the UPC (Politecnic Universitiy of Catalonia) and doctor degree in mathematics in 1996 from the UPC. Currently, she is titular professor of Statistics and Operative Investigation where she has been teaching for more than twenty five years. In the current moment she is the head of the emergent Research Group on Models of membrane computation applied to ecosystems. She has published seven books on statistics and quality control. She has published twelve scientifics papers in international journals. Her main research area are models of computation with membranes applied to the ecology and biology processes. Hers first researches were centred on the stocastics models, concretely Markov chain applied to natural processes where she has some works.

Agustín Riscos-Núñez got his Master degree in Mathematics in 2000, and then his PhD degree on 2004. Currently he is an associate professor at the Department of Computer Science and Artificial Intelligence in the University of Sevilla (Spain). He is a member of the Research Group on Natural Computing in the same University, and his main research interests within the membrane computing area are complexity theory, models for biological processes, and computer simulation. He has co-authored about 30 papers on international journals in the last years, and he has also participated in 15 conferences and workshops.

Miquel Rius-Font received his bachelor degree in Mathematics form the Universitat Autónoma de Barcelona and his doctor degree in mathematics from University of Barcelona. He is currently teaching mathematics at Catalonian Techology University and his research area is graph theory and natural computing. His interest involves isoperimetric problems as well as graph labeling problems.

Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. IV (2009), No. 3, pp. 301-310

Introducing a Space Complexity Measure for P Systems

Antonio E. Porreca, Alberto Leporati, Giancarlo Mauri, Claudio Zandron

Università degli Studi di Milano-Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano, Italy
E-mail: {porreca, leporati, mauri, zandron}@disco.unimib.it

Received: April 5, 2009
Accepted: May 30, 2009

Abstract: 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.
Keywords: membrane computing, complexity theory.

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:

(a) *Object evolution rule* of the form $[a \to w]_h^\alpha$

It can be applied inside 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 multiset w).

(b) *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 surrounding region contains an occurrence of the object a; the object a is sent into h becoming b and, simultaneously, the polarization of h is changed to β.

(c) *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 surrounding region becoming b and, simultaneously, the polarization of h is changed to β.

(d) *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.

(e) *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.

(f) *Non-elementary division rule* of the form

$$\left[\left[\,\right]^{+}_{h_1}\cdots\left[\,\right]^{+}_{h_k}\left[\,\right]^{-}_{h_{k+1}}\cdots\left[\,\right]^{-}_{h_n}\right]^{\alpha}_{h}\rightarrow\left[\left[\,\right]^{\delta}_{h_1}\cdots\left[\,\right]^{\delta}_{h_k}\right]^{\beta}_{h}\left[\left[\,\right]^{\varepsilon}_{h_{k+1}}\cdots\left[\,\right]^{\varepsilon}_{h_n}\right]^{\gamma}_{h}$$

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.

Note that here we have used the original definition of division rules introduced in [5]. Afterwards, other papers have proposed several alternatives for non-elementary division rules; nonetheless, in this paper these variants are not considered.

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. Notice that an object or membrane may remain inactive, even if it can trigger a rule, only when its use is inhibited by the application of another rule.
- Each object can be subject to only one rule during each 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 applied according to the above principles, 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 *recogniser* 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 $\boldsymbol{\Pi}=\{\Pi_x \mid x\in\Sigma^{\star}\}\subseteq\mathcal{D}$ be a family of P systems, either confluent or non-confluent. We say that $\boldsymbol{\Pi}$ *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 $\boldsymbol{\Pi}$ and restricting the amount of time available for deciding a language.

Definition 4. Consider a language $L \subseteq \Sigma^\star$, a class of recogniser 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 $\boldsymbol{MC}^\star_{\mathcal{D}}(f)$ if and only if there exists a family of confluent P systems $\boldsymbol{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L such that

- $\boldsymbol{\Pi}$ 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;
- $\boldsymbol{\Pi}$ 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 $\boldsymbol{PMC}^\star_{\mathcal{D}}$ iff there exists a semi-uniform family of confluent P systems $\boldsymbol{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L in polynomial time.

The analogous complexity classes for *non*-confluent P systems are denoted by $\boldsymbol{NMC}^\star_{\mathcal{D}}(f)$ and $\boldsymbol{NPMC}^\star_{\mathcal{D}}$.

Another set of complexity classes is defined in terms of *uniform* families of recogniser P systems:

Definition 5. Consider a language $L \subseteq \Sigma^\star$, a class of recogniser 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 $\boldsymbol{MC}_{\mathcal{D}}(f)$ if and only if there exists a family of confluent P systems $\boldsymbol{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L such that

- $\boldsymbol{\Pi}$ 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 $x \in L$.
- $\boldsymbol{\Pi}$ 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 $\boldsymbol{PMC}_{\mathcal{D}}$ iff there exists a uniform family of confluent P systems $\boldsymbol{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\} \subseteq \mathcal{D}$ deciding L in polynomial time.

The analogous complexity classes for *non*-confluent P systems are denoted by $\boldsymbol{NMC}_{\mathcal{D}}(f)$ and $\boldsymbol{NPMC}_{\mathcal{D}}$.

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 the 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 the 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 recogniser 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) recogniser P system, and let $\vec{\mathcal{C}} = (\mathcal{C}_0, \ldots, \mathcal{C}_m)$ 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 the rules in a maximally parallel way. The *space required by* $\vec{\mathcal{C}}$ is defined as

$$|\vec{\mathcal{C}}| = \max\{|\mathcal{C}_0|, \ldots, |\mathcal{C}_m|\}.$$

The *space required by* Π itself is then

$$|\Pi| = \max\{|\vec{\mathcal{C}}| \text{ such that } \vec{\mathcal{C}} \text{ is a halting computation of } \Pi\}.$$

Definition 9. Let $\boldsymbol{\Pi} = \{\Pi_x \mid x \in \Sigma^\star\}$ be a uniform or semi-uniform family of recogniser 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 recogniser P systems; let $f\colon \mathbb{N} \to \mathbb{N}$ and $L \subseteq \Sigma^\star$. Then $L \in \boldsymbol{MCSPACE}^\star_{\mathcal{D}}(f)$ iff L is decided by a semi-uniform family $\boldsymbol{\Pi} \subseteq \mathcal{D}$ of P systems operating within space bound f.

The corresponding complexity class for uniform families of confluent P systems is denoted by $\boldsymbol{MCSPACE}_{\mathcal{D}}(f)$, while in the non-confluent case we have the classes $\boldsymbol{NMCSPACE}^\star_{\mathcal{D}}(f)$ and $\boldsymbol{NMCSPACE}_{\mathcal{D}}(f)$ 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

$$\boldsymbol{PMCSPACE}^\star_{\mathcal{D}} = \bigcup_{k\in\mathbb{N}} \boldsymbol{MCSPACE}^\star_{\mathcal{D}}(O(n^k))$$
$$\boldsymbol{EXPMCSPACE}^\star_{\mathcal{D}} = \bigcup_{k\in\mathbb{N}} \boldsymbol{MCSPACE}^\star_{\mathcal{D}}\big(2^{O(n^k)}\big).$$

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 12. *The following inclusions hold:*

$$\boldsymbol{PMCSPACE}^{\star}_{\mathcal{D}} \subseteq \boldsymbol{EXPMCSPACE}^{\star}_{\mathcal{D}}$$
$$\boldsymbol{NPMCSPACE}^{\star}_{\mathcal{D}} \subseteq \boldsymbol{NEXPMCSPACE}^{\star}_{\mathcal{D}}.$$

Proposition 13. $\boldsymbol{MCSPACE}^{\star}_{\mathcal{D}}(f) \subseteq \boldsymbol{NMCSPACE}^{\star}_{\mathcal{D}}(f)$ *for each* $f\colon \mathbb{N} \to \mathbb{N}$*, and in particular*

$$\boldsymbol{PMCSPACE}^{\star}_{\mathcal{D}} \subseteq \boldsymbol{NPMCSPACE}^{\star}_{\mathcal{D}}$$
$$\boldsymbol{EXPMCSPACE}^{\star}_{\mathcal{D}} \subseteq \boldsymbol{NEXPMCSPACE}^{\star}_{\mathcal{D}}.$$

The following results mirror those which hold for Turing machines, and they describe closure properties and provide an upper bound for time requirements of P systems operating in bounded space.

Proposition 14. *The classes* $\boldsymbol{PMCSPACE}^{\star}_{\mathcal{D}}$*,* $\boldsymbol{NPMCSPACE}^{\star}_{\mathcal{D}}$*,* $\boldsymbol{EXPMCSPACE}^{\star}_{\mathcal{D}}$*, and* $\boldsymbol{NEXPMCSPACE}^{\star}_{\mathcal{D}}$ *are all closed under polynomial-time reductions.*

Proof. Let $L \in \boldsymbol{PMCSPACE}^{\star}_{\mathcal{D}}$ and let M be the Turing machine constructing the family $\boldsymbol{\Pi}$ that decides L. 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 $|f(x)|$ is bounded by a polynomial, M' operates in polynomial time and $\Pi_{f(x)}$ in polynomial space; but then $\boldsymbol{\Pi}' = \{\Pi_{f(x)} \mid x \in \Sigma^{\star}\}$ is a polynomially semi-uniform family of P systems deciding L' in polynomial space. Thus $L' \in \boldsymbol{PMCSPACE}^{\star}_{\mathcal{D}}$.

The proof for the three other classes is analogous. □

Proposition 15. $\boldsymbol{MCSPACE}^{\star}_{\mathcal{D}}(f)$ *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 16. *For each function* $f\colon \mathbb{N} \to \mathbb{N}$

$$\boldsymbol{MCSPACE}^{\star}_{\mathcal{D}}(f(n)) \subseteq \boldsymbol{MC}^{\star}_{\mathcal{D}}\left(2^{O(f(n)\log f(n))}\right)$$
$$\boldsymbol{NMCSPACE}^{\star}_{\mathcal{D}}(f(n)) \subseteq \boldsymbol{NMC}^{\star}_{\mathcal{D}}\left(2^{O(f(n)\log f(n))}\right).$$

Proof. Let $L \in \boldsymbol{MCSPACE}^{\star}_{\mathcal{D}}(f(n))$ be decided by the semi-uniform family $\boldsymbol{\Pi}$ of recogniser P systems in space f; let $\Pi_x \in \boldsymbol{\Pi}$ 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 $kf(n)\log f(n)$ over a finite alphabet, say with $b \geq 2$ symbols, for some constant k:

- We need $kf(n)$ symbols in order to represent the membrane structure and the objects it contains, as well as the polarizations.
- We also need to encode the labels of the membranes: even if they do not contribute to the space required by the P system, nonetheless each assignment of labels gives rise to a different configuration. Since all labels might be distinct, and there are at most $f(n)$ of them, $kf(n)\log f(n)$ symbols are needed.

Notice that there are less than $b^{kf(n)\log f(n)+1}$ such strings. Since Π_x is a recogniser P system, by definition every computation halts: then it must halt within $b^{kf(n)\log f(n)+1}$ steps in order to avoid repeating a previous configuration (thus entering an infinite loop). This number of steps is $2^{O(f(n)\log f(n))}$.

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 analyse 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 $\boldsymbol{P}$ (the so-called Milano theorem [11]); on the other hand, they can be used to solve *all* problems in $\boldsymbol{P}$ with a minimal amount of space, when a semi-uniform construction is considered:

Proposition 17. $\boldsymbol{P} \subseteq \boldsymbol{MCSPACE}^{\star}_{\mathcal{NAM}}(O(1))$.

Proof. Let $L \in \boldsymbol{P}$. Then there exists a deterministic Turing machine M deciding L in polynomial time. Now consider the family of P systems $\boldsymbol{\Pi} = \{\Pi_{no}, \Pi_{yes}\}$, where Π_{no} (resp. Π_{yes}) is the following trivial P system with active membranes:

- the membrane structure consists of the skin only, labelled by h;
- in the initial configuration, exactly one object a is located inside the skin;
- the only rule is $[a]^{\mathrm{o}}_h \rightarrow [\,]^{\mathrm{o}}_h\, no$ (resp. $[a]^{\mathrm{o}}_h \rightarrow [\,]^{\mathrm{o}}_h\, yes$).

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 $\boldsymbol{\Pi}$ 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 $\boldsymbol{NP}$-complete problems in polynomial (often even linear) time. In terms of space complexity, this can be stated as follows:

Proposition 18. $\boldsymbol{NP} \cup \boldsymbol{coNP} \subseteq \boldsymbol{EXPMCSPACE}^{\star}_{\mathcal{EAM}}$.

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 $\boldsymbol{EXPMCSPACE}^{\star}_{\mathcal{EAM}}$. □

This result can be improved when the use of non-elementary membrane division rules is allowed; indeed, all problems in $\boldsymbol{PSPACE}$ can be solved by such class of P systems with active membranes, denoted by $\mathcal{AM}$.

Proposition 19. $\boldsymbol{PSPACE} \subseteq \boldsymbol{EXPMCSPACE}^{\star}_{\mathcal{AM}}$.

Proof. In [10] a polynomial-time uniform solution to QBF (also known as QSAT), the canonical $\boldsymbol{PSPACE}$-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 $\boldsymbol{PSPACE}$:

Proposition 20. $\boldsymbol{PSPACE} \subseteq \boldsymbol{EXPMCSPACE}_{\mathcal{AM}}$.

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 21. *$\boldsymbol{PSPACE} \subseteq \boldsymbol{EXPMCSPACE}_{\mathcal{AM}^0}$, 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 22. *The three complexity classes $\boldsymbol{PMCSPACE}^{\star}_{\mathcal{NAM}}$, $\boldsymbol{PMCSPACE}^{\star}_{\mathcal{EAM}}$ and $\boldsymbol{PMCSPACE}^{\star}_{\mathcal{AM}}$ coincide.*

An idea which might be useful in proving this conjecture is precomputing 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 behaviour (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 $\boldsymbol{P} \neq \boldsymbol{PSPACE}$ 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 23. *Is $\boldsymbol{PMC}^{\star}_{\mathcal{D}} \neq \boldsymbol{PMCSPACE}^{\star}_{\mathcal{D}}$ 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 $\boldsymbol{PSPACE} = \boldsymbol{NPSPACE}$ 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 24. *Does $\boldsymbol{PMCSPACE}^{\star}_{\mathcal{D}} = \boldsymbol{NPMCSPACE}^{\star}_{\mathcal{D}}$ 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 25. *What are the relations between space complexity classes for P systems and traditional ones, such as* ***P****,* ***NP****,* ***PSPACE****,* ***EXP****,* ***NEXP****, and* ***EXPSPACE****?*

Bibliography

[1] A. Alhazov, C. Martín-Vide, L. Pan, Solving a PSPACE-Complete Problem by Recognizing P Systems with Restricted Active Membranes, *Fundamenta Informaticae*, vol. 58(2), pp. 67–77, 2003.

[2] 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*, 5th International Conference, MCU 2007, Orléans, France, Lecture Notes in Computer Science, vol. 4664, pp. 122–133, Springer, 2007.

[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, in: C. Calude, M. J. Dinneen, G. Păun, M. J. Pérez-Jiménez, G. Rozenberg, eds., *Unconventional Computation*, 4th International Conference, UC 2005, Sevilla, Spain, Lecture Notes in Computer Science, vol. 3699, pp. 105–116, Springer, 2005.

[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] G. Păun, P-Systems with Active Membranes: Attacking NP Complete Problems, in: I. Antoniou, C. Calude, M. J. Dinneen, eds., *Unconventional Models of Computation*, 2nd International Conference, UMC'2K, Brussels, Belgium, Springer, 2001.

[6] G. 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*, vol. 2(3), pp. 265–285, 2003.

[8] M. J. Pérez-Jiménez, Á. Romero-Jiménez, F. Sancho-Caparrini, The P versus NP Problem through Cellular Computing with Membranes, in: N. Jonoska, G. Păun, G. Rozenberg, eds., *Aspects of Molecular Computing*, Lecture Notes in Computer Science, vol. 2950, pp. 338–352, Springer, 2004.

[9] A. E. Porreca, G. Mauri, C. Zandron, Complexity Classes for Membrane Systems, *RAIRO Theoretical Informatics and Applications*, vol. 40(2), pp. 141–162, 2006.

[10] P. Sosík: The Computational Power of Cell Division in P Systems: Beating Down Parallel Computers?, *Natural Computing*, vol. 2(3), pp. 287–298, 2003.

[11] C. Zandron, C. Ferretti, G. Mauri, Solving NP-Complete Problems Using P Systems with Active Membranes, in: I. Antoniou, C. Calude, M. J. Dinneen, eds., *Unconventional Models of Computation*, 2nd International Conference, UMC'2K, Brussel, Belgium, 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*, vol. 87(1), pp. 79–91, 2008.

Antonio E. Porreca was born on September 18, 1982 in Monza, Italy. He got his B.Sc. and M.Sc. in Computer Science, in 2005 and 2008 respectively, from University of Milano-Bicocca, where he is now a Ph.D. student. His current research interests focus on complexity-theoretic aspects of membrane computing.

Alberto Leporati obtained a Ph.D. in Computer Science from the University of Milano (Italy) in 2002. Since 2004, he is assistant professor at the University of Milano-Bicocca. His research interests are in membrane computing, theoretical computer science and computational complexity.

Giancarlo Mauri is full professor of Computer Science at the University of Milano-Bicocca. His research interests are mainly in the area of theoretical computer science, and include: formal languages and automata, computational complexity, bioinformatics and unconventional computing models, in particular membrane systems. On these subjects, he published more than 200 scientific papers in international journals, contributed volumes and conference proceedings.

Claudio Zandron was born in 1972 in Milan, Italy. He received his M.Sc. in Computer Science from the University of Milano in 1996, and a Ph.D. degree in 2002 from the same university. Since 2006 he is associate professor at the University of Milano-Bicocca, Department of Informatics, Systems, and Communication. His research interests concern the areas of formal languages, molecular computing, and computational complexity.

Author index

Description

International Journal of Computers, Communications & Control (IJCCC) is a quarterly peer-reviewed publication started in 2006 by Agora University Editing House - CCC Publications, Oradea, ROMANIA.

Beginning with 2007, EBSCO Publishing is a licensed partner of IJCCC Publisher.

Every issue is published in online format (ISSN 1841-9844) and print format (ISSN 1841-9836).

Now we offer free online access to the full text of all published papers.

The printed version of the journal should be ordered, by subscription, and will be delivered by regular mail.

IJCCC is directed to the international communities of scientific researchers from the universities, research units and industry.

IJCCC publishes original and recent scientific contributions in the following fields:

- Computing & Computational Mathematics
- Information Technology & Communications
- Computer-based Control

To differentiate from other similar journals, the editorial policy of IJCCC encourages especially the publishing of scientific papers that focus on the convergence of the 3 "C" (Computing, Communication, Control).

The articles submitted to IJCCC must be original and previously unpublished in other journals. The submissions will be revised independently by minimum two reviewers and will be published only after end of the editorial workflow.

The peer-review process is single blinded: the reviewers know who the authors of the manuscript are, but the authors do not have access to the information of who the peer-reviewers are.

IJCCC also publishes:

- papers dedicated to the works and life of some remarkable personalities;
- reviews of some recent important published books

Also, IJCCC will publish as supplementary issues the proceedings of some international conferences or symposiums on Computers, Communications and Control, scientific events that have reviewers and program committee.

The authors are kindly asked to observe the rules for typesetting and submitting described in Instructions for Authors.

Thomson Reuters Subject Category of IJCCC:
AUTOMATION & CONTROL SYSTEMS
Category Description: Automation & Control Systems covers resources on the design and development of processes and systems that minimize the necessity of human intervention. Resources in this category cover control theory, control engineering, and laboratory and manufacturing automation.
COMPUTER SCIENCE, INFORMATION SYSTEMS
Category Description: Computer Science, Information Systems covers resources that focus

on the acquisition, processing, storage, management, and dissemination of electronic information that can be read by humans, machines, or both. This category also includes resources for telecommunications systems and discipline-specific subjects such as medical informatics, chemical information processing systems, geographical information systems, and some library science.

Editorial Workflow

The editorial workflow is performed using the online Submission System.

The peer-review process is single blinded: the reviewers know who the authors of the manuscript are, but the authors do not have access to the information of who the peer-reviewers are.

The following is the editorial workflow that every manuscript submitted to the IJCCC during the course of the peer-review process.

Every IJCCC submitted manuscript is inspected by the Editor-in-Chief/Associate Editor-in-Chief. If the Editor-in-Chief/Associate Editor-in-Chief determines that the manuscript is not of sufficient quality to go through the normal review process or if the subject of the manuscript is not appropriate to the journal scope, Editor-in-Chief/Associate Editor-in-Chief *rejects the manuscript with no further processing*.

If the Editor-in-Chief/Associate Editor-in-Chief determines that the submitted manuscript is of sufficient quality and falls within the scope of the journal, he sends the manuscript to the IJCCC Executive Editor/Associate Executive Editor, who manages the peer-review process for the manuscript.

The Executive Editor/Associate Executive Editor can decide, after inspecting the submitted manuscript, that it should be rejected without further processing. Otherwise, the Executive Editor/Associate Executive Editor assigns the manuscript to the one of Associate Editors.

The Associate Editor can decide, after inspecting the submitted manuscript, that it should be rejected without further processing. Otherwise, the Associate Editor assigns the manuscript to minimum two external reviewers for peer-review. These external reviewers may or may not be from the list of potential reviewers of IJCCC database.

The reviewers submit their reports on the manuscripts along with their recommendation of one of the following actions to the Associate Editor: Publish Unaltered; *Publish after Minor Changes; Review Again after Major Changes; Reject* (Manuscript is flawed or not sufficiently novel).

When all reviewers have submitted their reports, the Associate Editor can make one of the following editorial recommendations to the Executive Editor: Publish Unaltered; Publish after Minor Changes; Review Again after Major Changes; Reject.

If the Associate Editor recommends *"Publish Unaltered"*, the Executive Editor/Associate Executive Editor is notified so he/she can inspect the manuscript and the review reports. The Executive Editor/Associate Executive Editor can either override the Associate Editor's recommendation in which case the manuscript is rejected or approve the Associate Editor's recommendation in which case the manuscript is accepted for publication.

If the Associate Editor recommends "Review Again after Minor Changes", the Executive Editor/Associate Executive Editor is notified of the recommendation so he/she can inspect the manuscript and the review reports.

If the Executive Editor/Associate Executive Editor overrides the Associate Editor's recommendation, the manuscript is rejected. If the Executive Editor approves the Associate Editor's recommendation, the authors are notified to prepare and submit a final copy of their manuscript with the required minor changes suggested by the reviewers. Only the Associate Editor, and not the external reviewers, reviews the revised manuscript after the minor changes have been made by the authors. Once the Associate Editor is satisfied with the final manuscript, the manuscript can be accepted.

If the Associate Editor recommends *"Review Again after Major Changes"*, the recommendation is communicated to the authors. The authors are expected to revise their manuscripts

in accordance with the changes recommended by the reviewers and to submit their revised manuscript in a timely manner. Once the revised manuscript is submitted, the original reviewers are contacted with a request to review the revised version of the manuscript. Along with their review reports on the revised manuscript, the reviewers make a recommendation which can be "Publish Unaltered" or "Publish after Minor Changes" or "Reject". The Associate Editor can then make an editorial recommendation which can be "Publish Unaltered" or "Review Again after Minor Changes" or "Reject".

If the Associate Editor recommends rejecting the manuscript, either after the first or the second round of reviews, the rejection is immediate.

Only the Associate Editor-in-Chief can approve a manuscript for publication, where Executive Editor/Associate Executive Editor recommends manuscripts for acceptance to the Editor-in-Chief/Associate Editor-in-Chief.

Finally, recommendation of acceptance, proposed by the Associate Editor Chief, has to be approved by the Editor-in-Chief before publication.

Instructions for authors

Concurrent/Duplicate Submission

Submissions to IJCCC must represent original material.

Papers are accepted for review with the understanding that the same work has been neither submitted to, nor published in, another journal or conference. If it is determined that a paper has already appeared in anything more than a conference proceeding, or appears in or will appear in any other publication before the editorial process at IJCCC is completed, the paper will be automatically rejected.

Papers previously published in conference proceedings, digests, preprints, or records are eligible for consideration provided that the papers have undergone substantial revision, and that the author informs the IJCCC editor at the time of submission.

Concurrent submission to IJCCC and other publications is viewed as a serious breach of ethics and, if detected, will result in immediate rejection of the submission.

Preliminary/Conference Version(s)

If any portion of your submission has previously appeared in or will appear in a conference proceeding, you should notify us at the time of submitting, make sure that the submission references the conference publication, and supply a copy of the conference version(s) to our office. Please also provide a brief description of the differences between the submitted manuscript and the preliminary version(s).

Please be aware that editors and reviewers are required to check the submitted manuscript to determine whether a sufficient amount of new material has been added to warrant publication in IJCCC. If you have used your own previously published material as a basis for a new submission, then you are required to cite the previous work(s) and clearly indicate how the new submission offers substantively novel or different contributions beyond those of the previously published work(s). Any manuscript not meeting these criteria will be rejected. Copies of any previously published work affiliated with the new submission must also be included as supportive documentation upon submission.

Manuscript Preparing/Submission

The papers must be prepared using a LaTeX typesetting system. A template for preparing the papers is available on the journal website. In the template.tex file you will find instructions that will help you prepare the source file. Please, read carefully those instructions. (We are using MiKTeX 2.7).

Any graphics or pictures must be saved in Encapsulated PostScript (.eps) format.

Papers must be submitted electronically to the following email address: ccc.journal@gmail.com. You should send us the LaTeX source file (just one file - do not use bib files) and the graphics in a separate folder. You must send us also the pdf version of your paper.

The maximum number of pages of one article is 20. The publishing of a 12 page article is free of charge (including a bio-sketch). For each supplementary page there is a fee of 50 Euro/page that must be paid after receiving the acceptance for publication. The authors do not receive a print copy of the journal/paper, but the authors receive by email a copy of published paper in pdf format.

The papers must be written in English. The first page of the paper must contain title of the paper, name of author(s), an abstract of about 300 words and 3-5 keywords. The name, affiliation (institution and department), regular mailing address and email of the author(s) should be filled in at the end of the paper. Manuscripts must be accompanied by a signed copyright transfer form. The copyright transfer form is available on the journal website.

Please note: We will appreciate if the authors writes their own final version of the paper in LaTeX. But if the authors have difficulties with LaTeX and wish to send us their manuscript in Microsoft Word, the technical secretariat can do the transcription of the document. In this case, the paper can be sent in MS Word format with the following specifications: paper A4, font TNR 12p, single column. The graphics will be placed in the document but it has to be also attached separately in jpeg format.

Checklist:

1. Completed copyright transfer form.
2. Source (input) files.
    - One LaTeX file for the text.
    - EPS files for figures in a separate folder.
3. Final PDF file (for reference).

Order

If you are interested in having a subscription to "Journal of Computers, Communications and Control", please fill in and send us the order form below:

ORDER FORM		
I wish to receive a subscription to "Journal of Computers, Communications and Control"		
NAME AND SURNAME:		
Company:		
Number of subscription:	Price Euro	for issues yearly (4 number/year)
ADDRESS: City: Zip code: Country: Fax: Telephone: E-mail:		
Notes for Editors (optional)		

1. Standard Subscription Rates for Romania (4 issues/year, more than 400 pages, including domestic postal cost): 200 EURO.
2. Standard Subscription Rates for EU member countries (4 issues/year, more than 400 pages, including international postal cost): 360 EURO.
3. Standard Subscription Rates for other countries (4 issues/year, more than 400 pages, including domestic postal cost): 450 US dollars

For payment subscription rates please use following data:
HOLDER: Fundatia Agora, CUI: 12613360
BANK: BANK LEUMI ORADEA
BANK ADDRESS: Piata Unirii nr. 2-4, Oradea, ROMANIA
IBAN ACCOUNT for EURO: RO02DAFB1041041A4767EU01
IBAN ACCOUNT for LEI/ RON: RO45DAFB1041041A4767RO01
SWIFT CODE (eq. BIC): DAFBRO22
Mention, please, on the payment form that the fee is "for IJCCC".

EDITORIAL ADDRESS:

CCC Publications
Piata Tineretului nr. 8
ORADEA, jud. BIHOR
ROMANIA
Zip Code 410526
Tel.: +40 259 427 398
Fax: +40 259 434 925
E-mail: ccc@univagora.ro, Website: www.journal.univagora.ro

Copyright Transfer Form

To The Publisher of the International Journal of Computers, Communications & Control

This form refers to the manuscript of the paper having the title and the authors as below:

The Title of Paper (hereinafter, "Paper"):

. . . . . .

The Author(s):

. . . . . .

. . . . . .

. . . . . .

. . . . . .

The undersigned Author(s) of the above mentioned Paper here by transfer any and all copyright-rights in and to The Paper to The Publisher. The Author(s) warrants that The Paper is based on their original work and that the undersigned has the power and authority to make and execute this assignment. It is the author's responsibility to obtain written permission to quote material that has been previously published in any form. The Publisher recognizes the retained rights noted below and grants to the above authors and employers for whom the work performed royalty-free permission to reuse their materials below. Authors may reuse all or portions of the above Paper in other works, excepting the publication of the paper in the same form. Authors may reproduce or authorize others to reproduce the above Paper for the Author's personal use or for internal company use, provided that the source and The Publisher copyright notice are mentioned, that the copies are not used in any way that implies The Publisher endorsement of a product or service of an employer, and that the copies are not offered for sale as such. Authors are permitted to grant third party requests for reprinting, republishing or other types of reuse. The Authors may make limited distribution of all or portions of the above Paper prior to publication if they inform The Publisher of the nature and extent of such limited distribution prior there to. Authors retain all proprietary rights in any process, procedure, or article of manufacture described in The Paper. This agreement becomes null and void if and only if the above paper is not accepted and published by The Publisher, or is withdrawn by the author(s) before acceptance by the Publisher.

Authorized Signature (or representative, for ALL AUTHORS): . . . . . .

Signature of the Employer for whom work was done, if any: . . . . . .

Date: . . . . . .

Third Party(ies) Signature(s) (if necessary): . . . . . .