RGNC REPORT 1/2009

Seventh Brainstorming Week on Membrane Computing

Sevilla, February 2–February 6, 2009

Volume I

Rosa Gutiérrez-Escudero, Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Ignacio Pérez-Hurtado, Agustín Riscos-Núñez

Editors

Research Group on Natural Computing

REPORTS

UNIVERSIDAD DE SEVILLA

Seventh Brainstorming Week on Membrane Computing

Sevilla, February 2–February 6, 2009

Volume I

Rosa Gutiérrez-Escudero, Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Ignacio Pérez-Hurtado, Agustín Riscos-Núñez

Editors

Seventh Brainstorming Week on Membrane Computing

Sevilla, February 2–February 6, 2009

Volume I

Rosa Gutiérrez-Escudero, Miguel Angel Gutiérrez-Naranjo,
Gheorghe Păun, Ignacio Pérez-Hurtado,
Agustín Riscos-Núñez
Editors

RGNC REPORT 1/2009

Research Group on Natural Computing
Sevilla University

Fénix Editora, Sevilla, 2009

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

Preface

These proceedings, consisting of two volumes, contain the papers emerged from the Seventh Brainstorming Week on Membrane Computing (BWMC), held in Sevilla, from February 2 to February 6, 2009, in the organization of the Research Group on Natural Computing from the Department of Computer Science and Artificial Intelligence of Sevilla University. The first edition of BWMC was organized at the beginning of February 2003 in Rovira i Virgili University, Tarragona, and the next five editions took place in Sevilla at the beginning of February 2004, 2005, 2006, 2007, and 2008, respectively.

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

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

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

A selection of the papers from these volumes will be considered for publication in a special issues of *International Journal of Computers, Control and Communication.* After the first BWMC, a special issue of *Natural Computing* – volume 2, number 3, 2003, and a special issue of *New Generation Computing* – volume 22, number 4, 2004, were published; papers from the second BWMC have appeared in a special issue of *Journal of Universal Computer Science* – volume 10, number 5, 2004, as well as in a special issue of *Soft Computing* – volume 9, number 5, 2005; a selection of papers written during the third BWMC have appeared in a special issue of *International Journal of Foundations of Computer Science* – volume 17, number 1, 2006); after the fourth BWMC a special issue of *Theoretical Computer Science* was edited – volume 372, numbers 2-3, 2007; after the fifth edition, a special issue of *International Journal of Unconventional Computing* was edited – volume 5, number 5, 2009; finally, a selection of papers elaborated during the sixth BWMC has appeared in a special issue of *Fundamenta Informaticae* – volume 87, number 1, 2008. Other papers elaborated during the seventh BWMC will be submitted to other journals or to suitable conferences. The reader interested in the final version of these papers is advised to check the current bibliography of membrane computing available in the domain website `http://ppage.psystems.eu`.

***

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

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

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

34. Romero-Jiménez Alvaro, University of Sevilla, Spain,
`Alvaro.Romero@cs.us.es`
35. Sburlan Dragoş, Ovidius University, Constanţa, Romania,
`dsburlan@univ-ovidius.ro`
36. Sempere Luna José María, Polytechnical University of Valencia, Spain,
`jsempere@dsic.upv.es`
37. Vaszil György, Hungarian Academy of Sciences, Budapest, Hungary,
`vaszil@sztaki.hu`
38. Woods Damien, University of Sevilla, Spain,
`dwoods@us.es`
39. Zandron Claudio, University of Milano-Bicocca, Italy,
`zandron@disco.unimib.it`

As mentioned above, the meeting was organized by the Research Group on Natural Computing from Sevilla University (`http://www.gcn.us.es`)– and all the members of this group were enthusiastically involved in this (not always easy) work. The meeting was supported from various sources: (i) Proyecto de Excelencia de la Junta de Andalucía, grant TIC 581, (ii) Proyecto de Excelencia con investigador de reconocida valía, de la Junta de Andalucía, grant P08 – TIC 04200, (iii) Proyecto del Ministerio de Educación y Ciencia, grant TIN2006 – 13425, (iv) IV Plan Propio de la Universidad de Sevilla, (v) Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía, well as by the Department of Computer Science and Artificial Intelligence from Sevilla University.

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

Contents

Contents of Volume 2

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

Artiom Alhazov[2,1], Svetlana Cojocaru[1], Ludmila Malahova[1], Yurii Rogozhin[3,1]

[1] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova, Academiei 5, Chişinău MD-2028 Moldova
{artiom,sveta,mal,rogozhin}@math.md

[2] IEC, Department of Information Engineering, Graduate School of Engineering
Hiroshima University, Higashi-Hiroshima 739-8527 Japan

[3] Rovira i Virgili University, Research Group on Mathematical Linguistics
Pl. Imperial Tàrraco 1, Tarragona 43005 Spain

Summary. Membrane computing is a formal framework of distributed parallel computing. In this paper we implement working with the prefix tree by P systems with strings and active membranes.

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 solution is that based on applying parallelism; it has started to be promoted already in 1970s. For instance, the possibilities of applying massive parallelism in Machine Translation are considered in [4, 1]. We mention that 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 applying 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 there are handling of the dictionaries, e.g., dictionary lookup and dictionary completion. Exactly these problems constitute the subject of the present paper. In our approach we speak about dictionary represented by a prefix tree.

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 started by Gh. Păun in [2]. 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 [3] for an overview of the domain and [5] for the comprehensive bibliography.

2.1 Computing by P systems

Let O be a finite set of elements called symbols; the 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*

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

- *O is the working alphabet of the system whose elements are called objects.*
- *$\mathit{\Sigma}$ is an input alphabet.*
- *H is an alphabet whose elements are called labels.*
- *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 membranes labeled by elements of H.*
- *i_0 identifies the input region.*

A configuration of a P system is its “snapshot”, i.e., the current membrane structure and the multisets of strings of objects present in regions of the system. While 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, 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 $\mathit{\Sigma} \subseteq O$, then the *initial configuration* of a P system $\mathit{\Pi}$ with an input M over alphabet $\mathit{\Sigma}$ and input region i_0 is

$$(\mu, M_1, \cdots, M_{i_0-1}, M_{i_0} \cup M, M_{i_0+1}, \cdots, M_p).$$

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 can consider consecutive symbols in a string (otherwise, the time complexity would be much higher).
- 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, in the sense that 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 $h \in H, e \in E, a, b, c \in O^*$
(like the previous case, 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; the object can be modified during this process, as well as 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; the object can be modified during this process; 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 and 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).

The rules can only be applied simultaneously if they involve different objects and membranes (we repeat that rules of type (a) are not considered to involve a membrane), and such parallelism is maximal if no further rules are applicable to objects and membranes that were not involved.

3 Dictionary

Dictionary search represents computing a string-valued function

$$\{u_i \longrightarrow v_i \mid 1 \leq i \leq d\}$$

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 $\{u_i \mid 1 \leq i \leq d\}$, with strings $\$v_i\$'$ in regions corresponding to the nodes associated to u_i. 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

$$[\ [\ ]^0_a [\ [\ [\ \$flying\$'\ ]^0_t\ ]^0_a [\ [\ \$stored\$'\ ]^0_t\ ]^0_i\ ]_b [\ ]^0_c \cdots [\ ]^0_z\ ]^0_0$$

Let A_1, A_2 be the alphabets of the source and target languages, respectively. Consider a P system corresponding to the given dictionary.

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

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

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

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

Marking the region corresponding to the source word.

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

Replicating the translation.

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

Sending one copy of the translation to the environment.

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

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 $[\ ? \rightarrow ?_1 || ?_2\ ]_0^0$

Replicate the input.

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

Delay the second copy of the input for one step.

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

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

F4 $?_3 l[\ ]_l^+ \rightarrow [\ ?_3\ ]_l^0,\ l \in A_1$

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

F5 $[\ ?_1 l \rightarrow [\ ?_4\ ]_l^-\ ]_k^0,\ l,k \in A_1$

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 $[\ ?_1 ?' \rightarrow ?_7\ ]_l^0,\ l \in A_1$

Target location found, marking the first input copy.

F7 $[\ ?_7\ ]_l^0 \rightarrow [\ \ ]_l^- \emptyset,\ l \in A_1$

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

Replicating the translation.

F9 $[\ \$_2\ ]_l^e \rightarrow [\ \ ]_l^0 \$_2,\ l \in H,\ e \in \{0,-\}$

Sending one copy of the translation out.

F10 $[\ \$_1 \rightarrow \$\ ]_l^0,\ l \in A_1$

Keeping the other copy in the dictionary.

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

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

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

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

It waits for one step.

F14 $[\ ?_6 \rightarrow \emptyset\ ]_l^0,\ l \in A_1$

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

F15 $[\ ?_6\ ]_l^- \rightarrow [\ \ ]_l^0 ?_8,\ l \in A_1$

If not, the search fails.

F16 $[\ ?_8\ ]_l^0 \rightarrow [\ \ ]_l^0 ?_8,\ l \in A_1$

Sending the fail notification to the skin.

F17 $[\ ?_8 l \rightarrow ?_8\ ]_0^0$

Erasing the remaining part of the source word.

F18 $[\ ?_8?'\ ]_l^0 \rightarrow [\ \ ]_l^0$`fail`

Answering fail.

F19 $[\ ?_4 \rightarrow ?_9\ ]_l^-,\ l \in A_1$

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

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

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 $[\ ?_{11}\ ]_l^0 \rightarrow \emptyset,\ l \in A_1$

Erasing the auxiliary membrane.

3.3 Dictionary update

To add a pair of words $u \longrightarrow v$ to the dictionary, input the string $!u\$v\$'$ in region 1. Consider the following rules.

U1 $[\ ! \rightarrow !_1||!_2\ ]_0^0$

Replicate the input.

U2 $[\ !_2 \rightarrow !_3\ ]_0^0$

Delay the second copy of the input for one step.

U3 $!_1 l[\ \ ]_l^0 \rightarrow [\ !_1\ ]_l^+,\ l \in A_1$

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

U4 $!_3 l[\ \ ]_l^+ \rightarrow [\ !_3\ ]_l^0,\ l \in A_1$

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

U5 $[\ !_1 \rightarrow !_4\ ]_l^0,\ l \in A_1$

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 $[\ !_4 l \rightarrow [\ !_4\ ]_l^+\ ]_k^0,\ l, k \in A_1$

Creating missing membranes.

U7 $[\ !_4\$ \rightarrow \$\ ]_l^0,\ l \in A_1$

Releasing the target word in the corresponding location.

U8 $[\ !_3\$ \to \emptyset\ ]_l^0,\ l \in A_1$

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".

4 Discussion

In this paper we presented the algorithms of searching in a dictionary and completing 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, thesaura, dictionaries of exceptions, and even databases.

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.

References

1. H. Kitano: Challenges of massive parallelism. *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Chambery, France, 1993, vol. 1, 813–834.
2. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
3. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, 2002.
4. 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.
5. P systems webpage. `http://ppage.psystems.eu/`.

P Systems with Minimal Insertion and Deletion

Artiom Alhazov[1,3], Alexander Krassovitskiy[2],
Yurii Rogozhin[2,3], Sergey Verlan[4]

[1] IEC, Department of Information Engineering, Graduate School of Engineering
Hiroshima University, Higashi-Hiroshima 739-8527 Japan
[2] Research Group on Mathematical Linguistics, Rovira i Virgili University
Av. Catalunya, 35, Tarragona 43002 Spain
alexander.krassovitskiy@estudiants.urv.cat
[3] Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Academiei 5, Chişinău MD-2028 Moldova
{artiom,rogozhin}@math.md
[4] LACL, Département Informatique, Université Paris 12
61 av. Général de Gaulle, 94010 Créteil, France
verlan@univ-paris12.fr

Summary. In this paper we consider insertion-deletion P systems with priority of deletion over the insertion. We show that such systems with one symbol context-free insertion and deletion rules are able to generate $PsRE$. If one-symbol one-sided context is added to insertion or deletion rules but no priority is considered, then all recursively enumerable languages can be generated. The same result holds if a deletion of two symbols is permitted. We also show that the priority relation is very important and in its absence the corresponding class of P systems is strictly included in MAT.

1 Introduction

The operations of insertion and deletion are fundamental in formal language theory, and generative mechanisms based on them were considered (with linguistic motivation) for some time, see [14] and [6]. Related formal language investigations can be found in several places; we mention only [8], [10], [16], [19]. In the last years, the study of these operations has received a new motivation from molecular computing, see [3], [9], [21], [23], [15].

In general form, an insertion operation means adding a substring to a given string in a specified (left and right) context, while a deletion operation means removing a substring of a given string from a specified (left and right) context. A finite set of insertion-deletion rules, together with a set of axioms provide a language generating device: starting from the set of initial strings and iterating insertion-deletion operations as defined by the given rules we get a language. The number of axioms, the length of the inserted or deleted strings, as well as the length of the contexts where these operations take place are natural descriptional complexity measures in this framework. As expected, insertion and deletion operations with context dependence are very powerful, leading to characterizations of recursively enumerable languages. Most of the papers mentioned above contain such results, in many cases improving the complexity of insertion-deletion systems previously available in the literature.

Some combinations of parameters lead to systems which are not computationally complete [17], [11] or even decidable [24]. However, if these systems are combined with the distributed computing framework of P systems [20], then their computational power may strictly increase, see [12], [13].

In this paper we study P systems with insertion and deletion rules of one symbol without context. We show that this family is strictly included in MAT, however some non-context-free languages may be generated. If Parikh vectors are considered, then the corresponding family equals to $PsMAT$. When a priority of deletion over insertion is introduced, $PsRE$ can be characterized, but in terms of language generation such systems cannot generate a lot of languages because there is no control on the position of an inserted symbol. If one-sided contextual insertion or deletion rules are used, then this can be controlled and all recursively enumerable languages can be generated. The same result holds if a context-free deletion of two symbols is allowed.

2 Definitions

All formal language notions and notations we use here are elementary and standard. The reader can consult any of the many monographs in this area – for instance, [22] – for the unexplained details.

We denote by $|w|$ the length of a word w and by $|w|_a$ the number of occurrences of symbol a in w. For a word $w \in V^*$ we denote by $\Delta(w)$ all words w' having the same number of letters as w, $\Delta(w) = \{w' \mid, |w'|_a = |w|_a \text{ for all } a \in V\}$ and we denote by $⧢$ the binary shuffle operation. By $card(V)$ we denote the cardinality of the set V.

An *InsDel system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet, $T \subseteq V$, A is a finite language over V, and I, D are finite sets of triples of the form (u, α, v), $\alpha \neq \lambda$, where u and v are strings over V and λ denotes the empty string. The elements of T are *terminal* symbols (in contrast, those of $V - T$ are called nonterminals), those of A are *axioms*, the triples in I are *insertion rules*, and those from D are *deletion rules*. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted in between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from the context (u, v). As stated otherwise, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \to u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \to uv$. We refer by $\Longrightarrow$ to the relation defined by an insertion or deletion rule.

The language $L(ID)$ generated by ID is defined as $\{w \in T^* \mid A \ni x \Longrightarrow^* w\}$.

The complexity of an InsDel system $ID = (V, T, A, I, D)$ is described by the vector $(n, m, m'; p, q, q')$ called *size*, where

$$\begin{aligned} n &= \max\{|\alpha| \mid (u,\alpha,v) \in I\}, \; p = \max\{|\alpha| \mid (u,\alpha,v) \in D\},\\ m &= \max\{|u| \mid (u,\alpha,v) \in I\}, \; q = \max\{|u| \mid (u,\alpha,v) \in D\},\\ m' &= \max\{|v| \mid (u,\alpha,v) \in I\}, \; q' = \max\{|v| \mid (u,\alpha,v) \in D\}. \end{aligned}$$

We also denote by $INS_n^{m,m'}DEL_p^{q,q'}$ corresponding families of languages. Traditionally, in the literature, instead of pairs m/m' and q/q' the maximum of both numbers is used. However, such a complexity measure is not accurate and it cannot distinguish between universality and non-universality cases, see [24] and [11]. If some of the parameters n, m, m', p, q, q' is not specified, then we write symbol $*$ instead. For example, $INS_*^{0,0}DEL_*^{0,0}$ denotes the family of languages generated by *context-free InsDel systems*. InsDel systems of a “sufficiently large” size characterize RE, the family of recursively enumerable languages.

Now we present a definition of insertion-deletion P systems. The *insertion-deletion tissue P systems* are defined in an analogous manner.

An *insertion-deletion P system* is a construct

$\Pi = (O, T, \mu, M_1, \cdots, M_n, R_1, \cdots, R_n)$, where

- O is a finite alphabet,
- $T \subseteq O$ is the terminal alphabet,
- μ is the membrane (tree) structure of the system which has n membranes (nodes) and it can be represented by a word over the alphabet of correctly nested marked parentheses,
- M_i, for each $1 \le i \le n$, is a finite language associated to the membrane i,
- R_i, for each $1 \le i \le n$, is a set of insertion and deletion rules with target indicators associated to region i, of the following forms: $(u, x, v; tar)_a$, where (u, x, v) is an insertion rule, and $(u, x, v; tar)_e$, where (u, x, v) is a deletion rule, and the *target indicator tar* is from the set $\{here, in_j, out \mid 1 \le j \le n\}$.

An n-tuple $(N_1, \cdots, N_n)$ of finite languages over O is called a configuration of Π. The transition between the configurations consists in applying the insertion and deletion rules in parallel to all possible strings, non-deterministically, and following the target indications associated with the rules.

A sequence of transitions between configurations of a given insertion-deletion P system Π starting from the initial configuration is called a computation with respect to Π. We say that Π generates $L(\Pi)$, the result of its computations. It consists of all strings over T ever sent out of the system during its computations.

We denote by $ELSP_k(ins_p^{m,m'}, del_p^{q,q'})$ the family of languages $L(\Pi)$ generated by insertion-deletion P systems with at most $k \ge 1$ membranes and insertion and deletion rules of size at most $(n, m, m'; p, q, q')$. We omit the letter E if $T = O$. In this paper we also consider insertion-deletion P systems where deletion rules have a priority over insertion rules; the corresponding class is denoted as

$(E)LSP_k(ins_p^{m,m'} < del_p^{q,q'})$. Letter "t" is inserted before P to denote classes for the tissue case, e.g., $ELStP_k(ins_p^{m,m'}, del_p^{q,q'})$.

A *register machine* (introduced in [18], see also [4]) is a construct

$M = (d, Q, q_0, h, P)$, where

- d is the number of registers,
- Q is a finite set of bijective labels of instructions of P,
- $q_0 \in Q$ is the initial label,
- $h \in Q$ is the halting label, and
- P is the set of instructions of the following forms:

1. $p : (\mathrm{ADD}(k), q, s)$, with $p, q, s \in Q$, $1 \leq k \leq d$ ("increment" -instruction). Add 1 to register k and go to one of the instructions with labels q, s.
2. $p : (\mathrm{SUB}(k), q, s)$, with $p, q, s \in Q$, $1 \leq k \leq d$ ("decrement" -instruction). Subtract 1 from the positive value of register k and go to the instruction with label q, otherwise (if it is zero) go to the instruction with label s.
3. $h : HALT$ (the halt instruction). Stop the computation of the machine.

For generating languages over T, we use the model of a *register machine with output tape* (introduced in [18], see also [1]), which also uses a tape operation:

4. $p : (\mathrm{WRITE}(A), q)$, with $p, q \in Q$, $A \in T$.

The configuration of a register machine is $(q, n_1, \cdots, n_d)$, where $q \in Q$, $n_i \geq 0$, $1 \leq i \leq d$. A register machine generates an m-dimensional vector as follows: let the first m registers be output registers, and the computation starts from $(q_0, 0, \cdots, 0)$; if the configuration $(h, n_1, \cdots, n_d)$ is reached, then the resulting vector is $(n_1, \cdots, n_m)$. Without restricting generality we assume $(n_{m+1}, \cdots, n_d) = (0, \cdots, 0)$. The set of all vectors generated in this way by M is denoted by $Ps(M)$. It is known (e.g., see [18], [25]) that register machines generate $PsRE$. If the WRITE instruction is used, then RE can be generated.

In the case when a register machine cannot check whether a register is empty we say that it is partially blind; the second type of instructions is then written as $p : (\mathrm{SUB}(k), q)$ and the transition is undefined if register k is zero.

The word "partially" stands for an implicit test for zero at the end of a (successful) computation: counters $m + 1, \cdots, d$ should be empty. It is known, [4], that partially blind register machines generate exactly $PsMAT$ (Parikh sets of languages of matrix grammars without appearance checking).

3 Minimal Context-free Insertion-Deletion P Systems

It has been shown, [24], that systems in $INS_1^{0,0}DEL_*^{0,0}$ only generate strings obtained by inserting any number of specific symbols anywhere in words of a finite language; this is included in the regular languages family; strictly as, e.g., for

$L = \{a^*b^*\}$ the system has no control on the place of insertion or deletion in the string and the initial language is finite. Therefore, $INS_1^{0,0}DEL_1^{0,0} \subset REG$.

When a membrane structure is added to minimal insertion-deletion systems without context, their computational power is increased.

Theorem 1. *$PsStP_*(ins_1^{0,0}, del_1^{0,0}) = PsMAT$.*

Proof. It is not difficult to see that dropping the requirement of the uniqueness of the instructions with the same label, the power of partially blind register machines does not change, see, e.g., [4]. We use this fact for the proof.

The inclusion $PsStP_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsMAT$ follows from the simulation of minimal context-free insertion-deletion P systems by partially blind register machines, which are known to characterize $PsMAT$ [4]. Indeed, any rule $(\lambda, a, \lambda; q)_a \in R_p$ is simulated by instructions $p : (ADD(a), q)$. Similarly, rule $(\lambda, a, \lambda; q)_e \in R_p$ is simulated by instructions $p : (SUB(a), q)$.

The output region i_0 is associated to the final state, while the halting is represented by absence of the corresponding symbols (final zero-test) as follows. We assume that R_{i_0} has no insertion rules ($\emptyset$ can be generated by a trivial partially blind register machine), and the output registers correspond to those symbols that cannot be deleted by rules from R_{i_0}.

The converse inclusion follows from the simulation of partially blind register machines by P systems. Indeed, with every instruction p of the register machine we associate a cell. Instruction $p : (ADD(A_k), q)$ is simulated by rule $(\lambda, A_k, \lambda; q)_a \in R_p$, and instruction $p : (SUB(A_k), q)$ by $(\lambda, A_k, \lambda; q)_e \in R_p$. Final zero-tests: rules $(\lambda, A_k, \lambda; \#)_e \in R_h$, $k \geq m$, should be inapplicable ($R_\# = \emptyset$).

As the membrane structure is a tree, one-way inclusion follows.

Corollary 1. *$PsSP_*(ins_1^{0,0}, del_1^{0,0}) \subseteq PsMAT$.*

In terms of the generated language the above systems are not too powerful, even with priorities. Like in the case of insertion-deletion systems there is no control on the position of insertion. Hence, the language $L = \{a^*b^*\}$ cannot be generated, for insertion strings of any size. Hence we obtain:

Theorem 2. *$REG \backslash LStP_*(ins_n^{0,0} < del_1^{0,0}) \neq \emptyset$, for any $n > 0$.*

However, there are non-context-free languages that can be generated by such P systems (even without priorities and deletion).

Theorem 3. *$LStP_*(ins_1^{0,0}, del_0^{0,0}) \setminus CF \neq \emptyset$.*

Proof. It is easy to see that the language $\{w \in \{a, b, c\}^* : |w|_a = |w|_b = |w|_c\}$ is generated by such a system with 3 nodes, inserting consecutively a, b and c.

For the tree case the language $\{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$ can be generated in a similar manner.

We show a more general inclusion:

Theorem 4. *$ELStP_*(ins_n^{0,0}, del_1^{0,0}) \subset MAT$, for any $n > 0$.*

Proof. As in [11] we can suppose that there are no deletions of terminal symbols. We also suppose that there is only one initial string in the system, because there is no interaction between different evolving strings and the result matches the union of results for the systems with only one string. Consider a tissue P system Π with alphabet O, terminal symbols T, the set H of unique cell labels and the initial string w in cell labeled p_0. Such a system can be simulated by the following matrix grammar $G = (O \cup H, T, S, P)$.

For insertion instruction $(\lambda, a_1 \cdots a_n, \lambda; q)_a$ in cell p, the matrix $(p \to q, D \to Da_1D \cdots Da_nD) \in P$. For any deletion instruction $(\lambda, A, \lambda; q)_e$ in cell p, the matrix $(p \to q, A \to \lambda) \in P$. Three additional matrices $(h \to \lambda)$, $(D \to \lambda)$ and $(S \to q_0Da_1D \cdots Da_mD)$ $(w = a_1 \cdots a_m)$ shall be also added to P.

The above construction correctly simulates the system Π. Indeed, symbols D represent placeholders for all possible insertions. The first rule in the matrix permits simulates the navigation between cells.

Nevertheless, minimal context-free insertion-deletion systems with priorities do generate $PsRE$. This is especially clear for the tissue P systems: jumping to an instruction corresponds to sending a string to the associated region, and the entire construction is a composition of graphs shown in Figure 1. The decrement instruction works correctly because of priority of deletion over insertion.

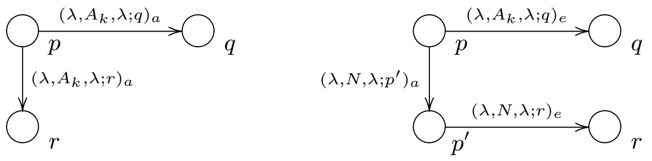

$(\lambda, A_k, \lambda; q)_a$
p
q
$(\lambda, A_k, \lambda; r)_a$
r
$(\lambda, A_k, \lambda; q)_e$
p
q
$(\lambda, N, \lambda; p')_a$
$(\lambda, N, \lambda; r)_e$
p'
r

Fig. 1. Simulating $p : (ADD(k), q, r)$ (left) and $p : (SUB(k), q, r)$ (right).

We now give a more sophisticated proof for the tree-like membrane structure.

Theorem 5. *$PsSP_*(ins_1^{0,0} < del_1^{0,0}) = PsRE$.*

Proof. The proof is done by showing that for any register machine $M = (n, Q, q_0, h, P)$ there is a P system $\Pi \in PsSP_*(ins_1^{0,0} < del_1^{0,0})$ with $Ps(M) = Ps(\Pi)$, and the result follows from the existence of register machines generating $PsRE$.

Let Q_+ (Q_-) be the sets of labels of increment (conditional decrement, respectively) instructions of a register machine, and let $Q = Q_+ \cup Q_- \cup \{h\}$ represent all labels. Consider a P system with alphabet $Q \cup \{A_i \mid 1 \le i \le d\} \cup \{Y\}$ and the following structure (illustrated in Figure 2)

$$\mu = [\ [\ [\ \prod_{p \in Q_+} \mu_{\langle p+ \rangle} \prod_{p \in Q_-} \mu_{\langle p- \rangle}\ ]_3\ ]_2\ ]_1, \text{ where}$$

$$\mu_{\langle p+ \rangle} = [\ [\ [\ ]_{p_3^+}\ ]_{p_2^+}\ ]_{p_1^+}\ ,\ p - \text{ increment},$$

$$\mu_{\langle p- \rangle} = [\ [\ [\ ]_{p_3^-}\ ]_{p_2^-}\ [\ [\ ]_{p_3^0}\ ]_{p_2^0}\ ]_{p_1^-}\ ,\ p - \text{ conditional decrement}.$$

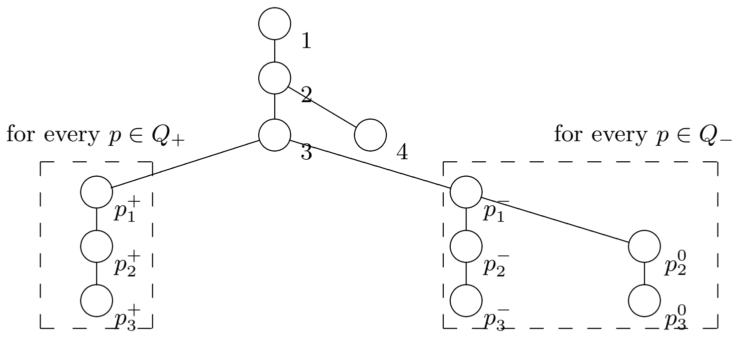


Fig. 2. Membrane structure for Theorem 5. The structures in the dashed rectangles are repeated for every instruction of the register machine.

Initially there is a single string q_0 in membrane 3. The rules are the following.

$$\begin{array}{lll}
R_1 = \{ & 1 :(\lambda, Y, \lambda; out)_e\}, & \\
R_2 = \{ & 2.1 :(\lambda, Y, \lambda; out)_a, & 2.2 :(\lambda, Y, \lambda; in_4)_e\}, \\
R_3 = \{ & 3.1 :(\lambda, p, \lambda; in_{p_1^+})_e \mid p \in Q_+\} \cup & \{3.2 :(\lambda, p, \lambda; in_{p_1^-})_e \mid p \in Q_-\} \\
\cup \{ & 3.3 :(\lambda, Y, \lambda; here)_e, & 3.4 :(\lambda, h, \lambda; out)_e\},
\end{array}$$

For any instruction $p : (ADD(k), q, s)$, $R_{p_3^+} = \emptyset$ and

$$\begin{array}{lll}
R_{p_1^+} = \{ & a.1.1 :(\lambda, A_k, \lambda; in_{p_2^+})_a, & a.1.2 :(\lambda, Y, \lambda; out)_a\}, \\
R_{p_2^+} = \{ & a.2.1 :(\lambda, q, \lambda; out)_a, & a.2.1' :(\lambda, s, \lambda; out)_a, \\
 & a.2.2 :(\lambda, q, \lambda; in_{p_3^+})_e, & a.2.2' :(\lambda, s, \lambda; in_{p_3^+})_e\},
\end{array}$$

For any instruction $p : (SUB(k), q, s)$, $R_{p_3^-} = R_{p_3^0} = \emptyset$ and

$$\begin{array}{llll}
R_{p_1^-} = \{ & e.1.1 :(\lambda, A_k, \lambda; in_{p_2^-})_e, & e.1.2 :(\lambda, Y, \lambda; in_{p_2^-})_a, & e.1.3 :(\lambda, Y, \lambda; out)_e\}, \\
R_{p_2^-} = \{ & e.2.1 :(\lambda, q, \lambda; out)_a, & e.2.2 :(\lambda, q, \lambda; in_{p_3^-})_e, & \\
 & e.2.3 :(\lambda, s, \lambda; in_{p_3^-})_e, & e.2.4 :(\lambda, Y, \lambda; here)_a\}, & \\
R_{p_2^0} = \{ & e.3.1 :(\lambda, s, \lambda; out)_a, & e.3.2 :(\lambda, q, \lambda; in_{p_3^0})_e, & e.3.3 :(\lambda, s, \lambda; in_{p_3^0})_e\}.
\end{array}$$

Configurations $(p, x_1, \cdots, x_n)$ of M are encoded by strings $\Delta(pA_1^{x_1} \cdots A_n^{x_n} Y^t)$, $t \geq 0$, in membrane 3. We say that such strings have a *simulating* form. Clearly, in the initial configuration the string is already in the simulating form.

To prove that system Π correctly simulates M we prove the following claims:

1. For any transition $(p, x_1, \cdots, x_n) \Longrightarrow (q, x_1', \cdots, x_n')$ in M there exists a computation in Π from the configuration containing $\Delta(pA_1^{x_1} \cdots A_n^{x_n} Y^t)$ in membrane 3 to the configuration containing $\Delta(qA_1^{x_1'} \cdots A_n^{x_n'} Y^{t'})$, $t' \geq 0$, in membrane 3 such that during this computation membrane 3 is empty in all intermediate steps and, moreover, this computation is unique.

2. For any successful computation in Π (yielding a non-empty result), membrane 3 contains only strings of the above form.
3. The result $(x_1, \cdots, x_n)$ in Π is obtained if and only if a string of form $\Delta(hA_1^{x_1} \cdots A_n^{x_n})$ appears in membrane 3.

Now we prove each claim from above. Consider a string $\Delta(pA_1^{x_1} \cdots A_n^{x_n}Y^t)$, $t \geq 0$, in membrane 3 of Π. Take an instruction $p : (ADD(k), q, s) \in P$. The only applicable rule in Π is from group 3.1 (in the future we simply say rule 3.1) yielding the string $\Delta(A_1^{x_1} \cdots A_n^{x_n}Y^t)$ in membrane p_1^+. After that rule $a.1.1$ is applied yielding string $\Delta(A_1^{x_1} \cdots A_k^{x_k+1} \cdots A_n^{x_n}Y^t)$ in membrane p_2^+. After that one of rules $a.2.1$ or $a.2.1'$ is applied; then rule $a.1.2$ yields one of strings $\Delta(zA_1^{x_1} \cdots A_k^{x_k+1} \cdots A_n^{x_n}Y^{t+1})$, $z \in \{q, s\}$, which is in the simulating form.

Now suppose that there is an instruction $p : (SUB(k), q, s) \in P$. Then the only applicable rule in Π is 3.2 which yields the string $\Delta(A_1^{x_1} \cdots A_n^{x_n}Y^t)$ in membrane p_1^-. Now if $x_k > 0$, then, due to the priority, rule $e.1.1$ will be applied followed by application of rules $e.2.4$, $e.2.1$ and $e.1.3$ which yields the string $\Delta(qA_1^{x_1} \cdots A_k^{x_k-1} \cdots A_n^{x_n}Y^{t'})$ that is in the simulating form. If $x_k = 0$, then rule $e.1.2$ will be applied (provided that all symbols Y were previously deleted by rule 3.3), followed by rules $e.3.1$ and $e.1.3$ which leads to the string $\Delta(sA_1^{x_1} \cdots A_n^{x_n})$ that is in the simulating form.

To show that membrane 3 is empty during the intermediate steps, we prove the following invariant:

Invariant 1 *During a successful computation, any visited membrane p_1^+ or p_1^- is visited an even number of times as follows: first a string coming from membrane 3 is sent to an inner membrane (p_2^+, p_2^- or p_2^0) and after that a string coming from an inner membrane is sent to membrane 3.*

Indeed, since there is only one string in the initial configuration, it is enough to follow only its evolution. Hence, a string may visit the node p_1^+ or p_1^- only if in the previous step symbol p was deleted by one of rules 3.1 or 3.2. If one of rules $a.1.2$ or $e.1.3$ is applied, then membrane 3 will contain a string of form $\Delta(A_1^{x_1} \cdots A_n^{x_n}Y^t)$ which cannot evolve anymore because all rules in membrane 3 imply the presence of a symbol from the set Q. Hence, the string is sent to an inner membrane. In the next step the string will return from the inner membrane by one of rules $a.2.1$, $a.2.1'$, $e.2.1$ or $e.3.1$ inserting a symbol from Q. If the string enters an inner membrane again, then it will be sent to a trap membrane (p_3^+, p_3^- or p_3^0) by rules deleting symbols from Q. Hence the only possibility is to go to membrane 3 (a string that visited membrane p_2^- will additionally use rule $e.2.4$).

For the second claim, it suffices to observe that the invariant above ensures that in membrane 3 only one symbol from Q can be present in the string.

The third claim holds since a string may move to membrane 2 if and only if the final label h of M appears in membrane 3. Then, the string is checked for the absence of symbols Y by rule 2.2 (note that symbols Y can be erased in membrane 3 by rule 3.3) and sent to the environment by rules 2.1 and 1.

By induction on the number of computational steps we obtain that Π simulates any computation in M. Claim 1 and 2 imply it is not possible to generate other strings and Claim 3 implies that the same result is obtained.

We remark that an empty string may be obtained during the proof. This string can still evolve using insertion rules. If we would like to forbid such evolutions, it suffices to use a new symbol, e.g., X, in the initial configuration, add new surrounding membrane and a rule that deletes X from it.

4 Small Contextual Insertion-Deletion P Systems

Although Theorem 5 shows that the systems from the previous section are quite powerful, they cannot generate RE without control on the place where a symbol is inserted. Once we allow a context in insertion and deletion rules, they can.

Theorem 6. *$LSP(ins_1^{0,1} < del_1^{0,0}) = RE$.*

Proof. We simulate a register machine with WRITE instructions. We implement this instruction as an ADD instruction, except the added symbol has to be inserted to the left of a special marker, deleted at the end, as follows:

- Replace any writing instruction $p : (WRITE(A), q, s)$, $A \in T$, of the machine by instructions $p : (ADD(A), q, s)$, considering output symbols A like new dummy registers. Construct the system Π as in Theorem 5.
- Change the initial string in membrane 3 to q_0M;
- Replace rules $a.1.1$ $((\lambda, A, \lambda; in_{p_2^+})_a \in R_{p_1^+})$ by $(\lambda, A, M; in_{p_2^+})_a$ for $A \in T$;
- Surround membrane 1 by a new skin membrane s and add to it the following rule $R_s = \{(\lambda, M, \lambda; out)_e\}$.

It is easy to see that the above construction permits to correctly simulate the register machine with writing instructions.

Taking M in the left context yields the mirror language. Since RE is closed with respect to the mirror operation we get the following corollary:

Corollary 2. *$LSP(ins_1^{1,0} < del_1^{0,0}) = RE$.*

A similar result holds if contextual deleting operation is allowed.

Theorem 7. *$LSP_*(ins_1^{0,0} < del_1^{1,0}) = RE$.*

Proof. As in Theorem 6, we use the construction from Theorem 5. However, an additional membrane is needed to simulate the writing instructions.

We modify the construction of Theorem 5 as follows. Let Q_s be the set of labels of WRITE instructions of a register machine. We add the following substructures $\mu_{\langle ps \rangle}$ inside membrane 3 (shown in Figure 3):

$$\mu = [\;[\;[\;\prod_{p\in Q_+}\mu_{\langle p+\rangle}\prod_{p\in Q_-}\mu_{\langle p-\rangle}\prod_{p\in Q_s}\mu_{\langle ps\rangle}\;]_3\;]_2\;]_1,\text{ where}$$

$$\mu_{\langle p+\rangle},\mu_{\langle p-\rangle}\text{ are defined as in Theorem 5 and,}$$

$$\mu_{\langle ps\rangle} = [\;[\;[\;[\;[\;[\;]_{p_7^s}\;]_{p_4^s}[\;]_{p_6^s}\;]_{p_3^s}[\;]_{p_5^s}\;]_{p_2^s}\;]_{p_1^s}.$$

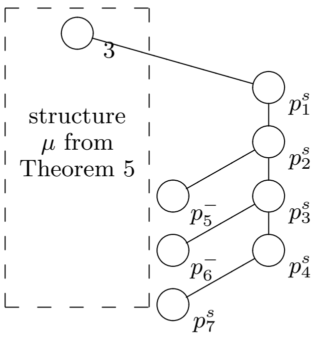


Fig. 3. Membrane structure for Theorem 7.

As in Theorem 5 the initial configuration contains a single string q_0 in region 3. The system contains sets of rules R_1, R_2, $R_{p_1^+}, R_{p_2^+}, R_{p_3^+}$, $R_{p_1^-}, R_{p_2^-}, R_{p_3^-}$, $R_{p_2^0}, R_{p_3^0}$ defined as in Theorem 5. There are also following additional rules for instructions $p : (WRITE(A), q)$ (the ruleset R'_3 shall be added to R_3).

$$\begin{aligned}
R'_3 &= \{ && 3.5 :(\lambda, p, \lambda; in_{p_1^s})_e \mid p \in Q_s\}, \\
R_{p_1^s} &= \{ && w.1.1 :(\lambda, M, \lambda; in_{p_2^s})_a, && w.1.2 :(\lambda, M, \lambda; out)_e\}, \\
R_{p_2^s} &= \{ && w.2.1 :(\lambda, M', \lambda; in_{p_3^s})_a, && w.2.2 :(\lambda, M', \lambda; out)_e\} \\
&\cup \{ && w.2.3 :(M, x, \lambda; in_{p_5^s})_e \mid x \in O\}, \\
R_{p_3^s} &= \{ && w.3.1 :(\lambda, A, \lambda; in_{p_4^s})_a, && w.3.2 :(\lambda, Y, \lambda; out)_a\} \\
&\cup \{ && w.3.3 :(x, M, \lambda; in_{p_6^s})_e \mid x \in O \setminus \{M', q\}\}, \\
R_{p_4^s} &= \{ && w.4.1 :(\lambda, q, \lambda; out)_a, && w.4.2 :(M', M, \lambda; in_{p_7^s})_e\}, \\
R_{p_5^s} &= \emptyset, && R_{p_6^s} = \emptyset,\ R_{p_7^s} = \emptyset.
\end{aligned}$$

We simulate the WRITE instruction as follows. Suppose the configuration of register machine is $pA_1^{x_1}\cdots A_d^{x_d}$ and the word $a_1\cdots a_n$ is written on the output tape. The corresponding simulating string in Π will be of form $p\Delta w$, where $w = \Delta(A_1^{x_1}\cdots A_d^{x_d}Y^t) ⧢ a_1\cdots a_n$, $t \geq 0$. After the deletion of the state symbol p, a marker M is inserted in the string by rule $w.1.1$. If M is not inserted at the right end of the string, in the next step rule $w.2.3$ is applicable and the string enters the trap membrane p_5^s. In the next step symbol M' is inserted in the string. If it is not inserted before M, then the string is sent to membrane p_6^s by rule $w.3.3$. Hence, at this moment the contents of membrane p_3^s is $wM'M$. If rule $w.3.2$ is used, then the string $Y ⧢ w$ reaches membrane 3 and no rule is applicable anymore. Otherwise, symbol A is inserted by rule $w.3.1$. If it is not between M' and M, then rule $w.4.2$ is applicable and the string enters membrane p_7^s. After that q is inserted between A and M, otherwise the trapping rule $w.3.3$ is applicable. At this moment, the configuration of the system consists of the string $w_t M' A q M$ in membrane p_3^s. Now if the rule $w.3.1$ is used, then the string is sent to the trap membrane by rule $w.4.1$. Otherwise, rule $w.3.2$ should be used followed by the application of rules $w.2.2$ and $w.1.2$, leading to string $Y \sqcup\!\sqcup\, wAq$ in membrane 3. Hence, the symbol A is appended at the end of the string. At the end of the computation, all symbols from $O - T$ are deleted and a word generated by M is obtained.

Since RE is closed with respect to the mirror operation we obtain:

Corollary 3. *$LSP(ins_1^{0,0} < del_1^{0,1}) = RE$.*

We remark that the contextual deletion was used only to check for erroneous evolutions. Therefore we can replace it by a context-free deletion of two symbols.

Theorem 8. *$LSP_*(ins_1^{0,0} < del_2^{0,0}) = RE$.*

Proof. We modify the proof of Theorem 7 as follows.

- Replace rules $w.2.3$ ($(M, x, \lambda; in_{p_5^s})_e \in R_{p_2^s}$) by rules $(\lambda, Mx, \lambda; in_{p_5^s})_e$,
- Replace rules $w.3.3$ ($(M, x, \lambda; in_{p_6^s})_e \in R_{p_3^s}$) by rules $(\lambda, xM, \lambda; in_{p_6^s})_e$,
- Replace rules $w.4.2$ ($(M', M, \lambda; in_{p_7^s})_e \in R_{p_4^s}$) by rules $(\lambda, M'M, \lambda; in_{p_7^s})_e$.

The role of the new rules is the same as the role of the rules that were replaced. More exactly, the system checks whether two certain symbols are consecutive and if so, the string is blocked in a non-output region.

We mention that the counterpart of Theorem 8 obtained by interchanging parameters insertion and deletion rules is not true, see Theorem 2.

5 Conclusions

We showed several results concerning P systems with insertion and deletion rules of small size. Surprisingly, systems with context-free rules inserting and deleting only one symbol are quite powerful and generate $PsRE$ if the priority of deletion over insertion is used. From the language generation viewpoint such systems are not too powerful and no language specifying the order of symbols can be generated. To be able to generate more complicated languages we considered systems with one-symbol one-sided insertion or deletion contexts. In both cases we obtained that any recursively enumerable language can be generated. The same result holds if a context-free deletion of two symbols is allowed. The counterpart of the last result is not true, moreover Theorem 2 shows that the insertion of strings of an arbitrary size still cannot lead to generating languages like a^*b^*.

We also have considered one-symbol context-free insertion-deletion P systems without the priority relations and we showed that in terms of Parikh sets these systems characterize the $PsMAT$ family. However, in terms of the generated language such systems are strictly included in MAT.

Most of results above were obtained using rules with target indicators. It is interesting to investigate the computational power of systems with non-specific target indicators *in* or *go*. Another open problem is to replace the priority relation by some other mechanism without decreasing the computational power.

Acknowledgments

The first author acknowledges the support of the Japan Society for the Promotion of Science, and the Grant-in-Aid, project 20·08364. The first, the third and the fourth authors acknowledge the support by the Science and Technology Center in Ukraine, project 4032. The third author gratefully acknowledges the support of the European Commission, project MolCIP, MIF1-CT-2006-021666. The second author acknowledges the support PIF program of University Rovira i Virgili, and project no. MTM2007-63422 from the Ministry of Science and Education of Spain.

References

1. A. Alhazov, R. Freund, A. Riscos-Núñez: One and two polarizations, membrane creation and objects complexity in P systems. *Proc. SYNASC'05*, IEEE Computer Society, 2005, 385–394.
2. A. Alhazov, R. Freund, Yu. Rogozhin: Computational power of symport/antiport: History, advances, and open problems. *Proc. WMC 2005*, Vienna, LNCS 3850, Springer, 2006, 1–30.
3. M. Daley, L. Kari, G. Gloor, R. Siromoney: Circular contextual insertions/deletions with applications to biomolecular computation. *Proc. of 6th Int. Symp. on String Processing and Information Retrieval*, SPIRE'99, Cancun, 47–54.
4. R. Freund, O.H. Ibarra, Gh. Păun, H.-C. Yen: Matrix languages, register machines, vector addition systems. *Third Brainstorming Week on Membrane Computing*, Sevilla, 2005, 155–168.
5. R. Freund, M. Oswald: GP systems with forbidding context. *Fundamenta Informaticae*, 49, 1–3 (2002), 81–102.
6. B.S. Galiukschov: Semicontextual grammars. *Matematika Logica i Matematika Linguistika*, Tallin University, 1981, 38–50 (in Russian).
7. S.A. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7 (1978), 311–324.
8. L. Kari: *On Insertion and Deletion in Formal Languages*. PhD Thesis, University of Turku, 1991.
9. L. Kari, Gh. Păun, G. Thierrin, S. Yu: At the crossroads of DNA computing and formal languages: Characterizing RE using insertion-deletion systems. *Proc. DNA Based Computers, 1997*, Philadelphia, 318–333.
10. L. Kari, G. Thierrin: Contextual insertion/deletion and computability. *Information and Computation*, 131, 1 (1996), 47–61.

11. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: Further results on insertion-deletion systems with one-sided contexts. *Proc. LATA 2008*, LNCS 5196, Springer, 2008, 347–358.
12. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: One-sided insertion and deletion: Traditional and P systems case. *Proc. CBM 2008*, Vienna, 53–64.
13. A. Krassovitskiy, Yu. Rogozhin, S. Verlan: Computational power of P systems with small size insertion and deletion rules. *Proc. CSP 2008*, Cork, 137–148.
14. S. Marcus: Contextual grammars. *Rev. Roum. Math. Pures Appl.*, 14 (1969), 1525–1534.
15. M. Margenstern, Gh. Păun, Yu. Rogozhin, S. Verlan: Context-free insertion-seletion Systems. *Theoretical Computer Science*, 330 (2005), 339–348.
16. C. Martin-Vide, Gh. Păun, A. Salomaa: Characterizations of recursively enumerable languages by means of insertion grammars. *Theoretical Computer Science*, 205, 1–2 (1998), 195–205.
17. A. Matveevici, Yu. Rogozhin, S. Verlan: Insertion-deletion systems with one-sided contexts. LNCS 4664, Springer, 2007, 205–217.
18. M.L.Minsky: *Computation. Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, NJ, 1967.
19. Gh. Păun: *Marcus Contextual Grammars*. Kluwer, Dordrecht, 1997.
20. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.
21. Gh. Păun, G. Rozenberg, A. Salomaa: *DNA Computing. New Computing Paradigms*. Springer, Berlin, 1998.
22. G. Rozenberg, A. Salomaa, eds.: *Handbook of Formal Languages*. Springer, Berlin, 1997.
23. A. Takahara, T. Yokomori: On the computational power of insertion-deletion systems. *Proc. DNA Based Computers 2002*, Sapporo, LNCS 2568, Springer, 2003, 269–280.
24. S. Verlan: On minimal context-free insertion-deletion systems. *Journal of Automata, Languages and Combinatorics*, 12, 1-2 (2007), 317-328.
25. D.Wood: *Theory of Computation.* Harper and Row, New York, 1987.

A Short Note on Reversibility in P Systems

Artiom Alhazov[1,2], Kenichi Morita[1]

[1] IEC, Department of Information Engineering, Graduate School of Engineering
Hiroshima University, Higashi-Hiroshima 739-8527 Japan
morita@iec.hiroshima-u.ac.jp

[2] Institute of Mathematics and Computer Science
Academy of Sciences of Moldova, Academiei 5, Chişinău MD-2028 Moldova
artiom@math.md

Summary. Membrane computing is a formal framework of distributed parallel computing. In this paper we study the reversibility and maximal parallelism of P systems from the computability point of view. The notions of reversible and strongly reversible systems are considered. The universality is shown for one class and a negative conjecture is stated for a more restricted class of reversible P systems. For one class of strongly reversible P systems, a very strong limitation is found, and it is shown that this limitation does not hold for a less restricted class.

1 Introduction

Reversibility is an important property of computational systems. It has been well studied for circuits of logical elements ([3]), circuits of memory elements ([7]), cellular automata ([8]), Turing machines ([1], [10]), register machines ([6]). Reversibility as a syntactical property is closely related to the microscopic reversibility, so it implies that the computation does not increase the entropy of the system, which in turn assume better miniaturization possibilities for potential implementation.

A slightly different view on reversible systems is given for type-0 grammars ([9]). The so-called uniquely parsable grammars are studied. In very simple words, this property (still being syntactical) implies that the generation of any word in the language is unique (modulo the order of applying the rules in case when the composition of applying them is commutative). The advantage of having such a property is that it is easier to analyze their behavior.

Clearly, this reason remains valid even if the property of reversibility becomes undecidable (just like the property of determinism in certain membrane systems). Moreover, reversibility essentially is backward determinism. Reversible P systems already were considered ([4]), but the model is energy-based (so the parallelism is invariant-driven rather than maximal) and the main result is the simulation of the Fredkin gate (so construction of a universal system in this way would use an infinite structure). In this paper we focus on the interplay between reversibility and maximal parallelism, from the viewpoint of computability.

2 Definitions

In this paper we illustrate the reversibility concepts on P systems with symport/antiport and one membrane, sometimes with inhibitors or priorities. For simplicity, we also assume that the environment contains an unbounded supply of all objects. The system thus can be defined by the alphabet, the initial multiset, the set of rules associated to the membrane, and the set of terminal objects. Throughout this paper we represent multisets by strings. We write an antiport rule sending a multiset x out and bringing a multiset y in as x/y, and the symport case corresponds to $y = \lambda$. If a rule has inhibitor a, we write it as $x/y|_{\neg a}$. The priority relationship is denoted by $>$. It is not difficult to generalize the definitions for the models with multiple membranes and changing membrane structure, but it is not important here.

Consider a P system Π with alphabet O. In our setting, a configuration is defined by the multiset of objects inside the membrane, represented by some string $u \in O^*$. The space $\mathcal{C}$ of configurations is essentially $|O|$-dimensional space with non-negative integer coordinates. We use the usual definitions of maximally parallel transition ([11]). It induces an infinite graph of $\mathcal{C}$. Notice that the halting configurations (and only them) have out-degree zero.

We call Π **strongly reversible** if every configuration has in-degree at most one. We call Π **reversible** if every reachable configuration has in-degree at most one.

A property equivalent to reversibility is determinism of a dual P system ([2]).

The result of a halting computation is the number of terminal objects inside the membrane when the system halts. The set $N(\Pi)$ of numbers generated by a P system Π is the set of results of all its computations. The family of number sets generated by reversible P systems with features α is denoted by $NrOP_1(\alpha)_T$, where $\alpha \subseteq \{sym_*, anti_*, inh, Pri\}$ and the braces of the set notation are omitted. Subscript T means that only terminal objects contribute to the result of computations; if $T = O$, we may omit specifying it in the description and we then also omit the subscript T in the notation. To bound the weight (i.e., maximal number of objects sent in a direction) of symport or antiport rule, associated $*$ is replaced by the actual number. For strong reversible systems, we replace in the notation r by r_s.

2.1 Register machines

In this paper we consider register machines with increment, unconditional decrement and test instructions, see also [6].

A register machine is a tuple $M = (n, Q, q_0, q_f, I)$ where

- n is the number of registers;
- I is a set of instructions labeled by elements of Q;
- $q_0 \in Q$ is the initial label;
- $q_f \in Q$ is the final label.

The allowed instructions are:

- $(q : i?, q', q'')$ - jump to instruction q'' if the contents of register i is zero, otherwise proceed to instruction q';
- $(q : i+, q', q'')$ - add one to the contents of register i and proceed to either instruction q' or q'', non-deterministically;
- $(q : i-, q', q'')$ - subtract one from the contents of register i and proceed to either instruction q' or q'', non-deterministically;
- $(q_f : halt)$ - finish the computation; it is a unique instruction with label q_f.

If $q' = q''$ for every instruction $(q : i+, q', q'')$ and for every instruction $(q : i-, q', q'')$, then the machine is called deterministic.

A register machine is called reversible if for some state q there is more than one instruction leading to it, then exactly two exist, they test the same register, one leads to q if the register is zero and the other one leads to q if the register is positive. More formally, for any two different instructions $(q_1 : i_1\alpha_1, q_1', q_1'')$ and $(q_2 : i_2\alpha_2, q_2', q_2'')$, it holds that $q_1' \neq q_2'$ and $q_1'' \neq q_2''$. Moreover,

$$\text{if } q_1' = q_2'' \text{ or } q_1'' = q_2', \text{ then } \alpha_1 = \alpha_2 =? \text{ and } i_1 = i_2.$$

It has been shown ([6]) that reversible register machines are universal. It follows that non-deterministic reversible register machines can generate any recursively enumerable set of non-negative integers as a value of the first register by all its possible computations starting from all registers having zero value.

3 Examples and Universality

We now present a few examples to illustrate the definitions.

Example 0: Consider a P system $\Pi_0 = (\{a, b\}, a, \{a/ab\})$. It is strongly reversible (for a preimage, remove as many copies of b as there are copies of a, in case it is possible and there is at least one copy of a), but no halting configuration is reachable. Therefore, $\emptyset \in Nr_sOP_1(anti_*)$.

Example 1: Consider a P system $\Pi_1 = (\{a, b, c\}, a, \{a/ab,\ a/c\})$. It generates the set of positive integers and it is reversible (for the preimage, replace c with a or ab with b), but not strongly reversible (e.g., $aa \Rightarrow cc$ and $ac \Rightarrow cc$). Hence, $\mathbb{N}_+ \in NrOP(anti_2)$.

Example 2: Consider a P system $\Pi_2 = (\{a, b\}, aa, \{aa/ab,\ ab/bbb\})$. It is reversible ($aa$ has in-degree 0, while ab and bbb have in-degree 1, and no other configuration is reachable), but not strongly reversible (e.g., $aab \Rightarrow abbb$ and $aabb \Rightarrow abbb$).

Example 3: Any P system with a rule x/λ, $x \in O^+$, is not reversible. Therefore, symport rules cannot be actually used in a reversible P systems with one membrane.

Example 4: Any P system with rules x_1/y, x_2/y that applied at least one of them in some computation is not reversible.

We now show that reversible P systems with either inhibitors or priorities are universal.

Theorem 1. *$NrOP_1(anti_2, Pri)_T = NrOP_1(anti_2, inh)_T = NRE$.*

Proof. We reduce the theorem statement to the claim that such P systems simulate the work of any reversible register machine $M = (n, Q, q_0, q_f, I)$. Consider a P system

$$\begin{aligned}
\Pi &= (O, q_0, R, \{r_1\}), \text{ where}\\
O &= \{r_i \mid 1 \le i \le n\} \cup Q,\\
R &= \{q/q'r_i, q/q''r_i \mid (q : i+, q', q'') \in I\}\\
&\quad \cup \{qr_i/q', qr_i/q'' \mid (q : i-, q', q'') \in I\} \cup R_t,\\
R_t &= \{q/q''|_{\neg r_i}, qr_i/q'r_i \mid (q : i?, q', q'') \in I\}.
\end{aligned}$$

Inhibitors can be replaced by priorities by redefining R_t as follows.

$$R_t = \{qr_i/q'r_i > q/q'' \mid (q : i?, q', q'') \in I\}.$$

Since there is a bijection between the configurations of Π containing one symbol from Q and the configurations of M, the reversibility of Π follows from the correctness of simulation, the reversibility of M, and from the fact that the number of symbols from Q is preserved by transitions of Π.

4 Limitations

The construction in the theorem above uses both cooperation and additional control. It is natural to ask whether both inhibitors and priorities can be avoided. Yet, consider the following situation. Let $(p : i?, s, q''), (q : i?, q', s) \in I$. It is usual for reversible register machines to have this, since the preimage of configuration sC depends on register i. Nevertheless, P systems with maximal parallelism without additional control can only implement a zero-test by try-and-wait-then-check strategy. In this case, the object containing the information about the register p finds out the result of checking after a possible action of the object related to the register. Therefore, when the state represented in the configuration of the system changes to s, it obtains an erroneous preimage representing state q. This leads to the following

Conjecture 1. Reversible P systems without priorities and without inhibitors are not universal.

Now consider a strongly reversible P system. The following theorem establishes a very serious limitation on such systems if no additional control is used.

Theorem 2. *In strongly reversible P systems without inhibitors and without priorities, every configuration is either halting or induces only infinite computation(s).*

Proof. If the right-hand side of every rule contains a left-hand side of some rule, then the claim holds. Otherwise, let x/y be a rule of the system such that y does not contain the left-hand side of any rule. Then $x \Rightarrow y$ and y is a halting configuration. It is not difficult to see that $xy \Rightarrow yy$ (objects y are idle) and $xx \Rightarrow yy$ (the rule can be applied twice). Therefore, such a system is not strongly reversible, which proves the theorem.

Therefore, the strongly universal systems without additional control can only generate singletons, i.e., $Nr_sOP_1(anti_*)_T = \{\{n\} \mid n \in \mathbb{N}\}$, and only in a degenerate way, i.e., without actually computing.

It turns out that the theorem above does not hold if inhibitors are used. Consider a system $\Pi_3 = (\{a, b\}, a, \{a/b|_{\neg b}\})$. If at least one object b is present or no objects a are present, such a configuration is a halting one. Otherwise, all objects a are exchanged by objects b. Therefore, the only possible transitions in the space of all configurations are of the form $a^n \Rightarrow b^n$, $n > 1$, and the system is strongly reversible.

5 Discussion

We outlined the concepts of reversibility and strong reversibility for P systems, concentrating on the case of symport/antiport rules (possibly with control such as priorities or inhibitors) with one membrane, assuming that the environment contains an unbounded supply of all objects.

We showed that reversible P systems with control are universal, and we conjectured that this result does not hold without control. Moreover, the strongly reversible P systems without control do not halt unless the starting configuration is halting, but this is no longer true if inhibitors are used.

Showing related characterizations might be quite interesting. Many other problems are still open, e.g., reversibility of P systems with active membranes.

Acknowledgments

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. He also acknowledges the support by the Science and Technology Center in Ukraine, project 4032.

References

1. C.H. Bennett: Logical reversibility of computation. *IBM J. Res. Dev.*, 17 (1973), 525-532.
2. O. Agrigoroaiei, G. Ciobanu: Dual P systems. *Membrane Computing - 9th International Workshop*, LNCS 5391, 2009, 95–107.
3. E. Fredkin, T. Toffoli: Conservative logic. *Int. J. Theoret. Phys.*, 21 (1982), 219-253.
4. A. Leporati, C. Zandron, G. Mauri: Reversible P systems to simulate Fredkin circuits. *Fundam. Inform.*, 74, 4 (2006), 529–548.
5. M.L. Minsky: *Computation: Finite and Infinite Machines.* Prentice-Hall, Englewood Cliffs, NJ, 1967.
6. K. Morita: Universality of a reversible two-counter machine. *Theoret. Comput. Sci.*, 168 (1996), 303-320.
7. K. Morita: A simple reversible logic element and cellular automata for reversible computing. *Proc. 3rd Int. Conf. on Machines, Computations, and Universality*, LNCS 2055, 2001, 102-113.
8. K. Morita: Simple universal one-dimensional reversible cellular automata. *J. Cellular Automata*, 2 (2007), 159-165.
9. K. Morita, N. Nishihara, Y. Yamamoto, Zh. Zhang: A hierarchy of uniquely parsable grammar classes and deterministic acceptors. *Acta Inf.*, 34, 5 (1997), 389–410.
10. K. Morita, Y. Yamaguchi: A universal reversible Turing machine. *Proc. 5th Int. Conf. on Machines, Computations, and Universality*, LNCS 4664, 2007, 90-98.
11. Gh. Păun: *Membrane Computing. An Introduction.* Springer, Berlin, 2002.
12. P systems website: `http://ppage.psystems.eu/`.

Mutual Mobile Membranes Systems with Surface Objects

Bogdan Aman, Gabriel Ciobanu

[1] Romanian Academy, Institute of Computer Science
[2] A.I.Cuza University of Iaşi, Romania
baman@iit.tuiasi.ro, gabriel@info.uaic.ro

Summary. In this paper we introduce mutual mobile membranes with surface objects, systems which have biological motivation. In P systems with mobile membranes with surface objects, a membrane may enter or exit another membrane. The second membrane just undergoes the action, meaning that it has no control on when the movement takes place. This kind of movement illustrates the lack of an agreement (synchronization) similar to an asynchronous evolution. In mutual mobile membranes with surface objects this aspect is adjusted: any movement takes place only if both participants agree by synchronizing their evolution. In membranes two kinds of competition can occur: resource competition and location competition. Resource competition refers to rules which request the same resources, and the available resources can only be allocated to some of the rules. Location competition refers to the movement of a membrane in the hierarchical structure of the membrane systems under the request of some conflict rules. We use the two variants of membrane systems in order to describe and explain these kinds of competition, and introduce synchronizing objects in mutual mobile membranes which will help to solve the resource and location competitions.

1 Introduction

Two recent computational models have been inspired from the structure and the functioning of the living cell: membrane systems [16, 17] and brane calculus [8]. Although the models start from the same observation, they are build having in mind different goals: membrane systems investigate formally the computational nature and power of various features of membranes, while the brane calculus is intended to give a faithful and intuitive representation of the biological reality. In [9] the initiators of these two formalisms describe the goals they had in mind: “While membrane computing is a branch of natural computing which tries to abstract computing models, in the Turing sense, from the structure and the functioning of the cell, making use especially of automata, language, and complexity theoretic tools, brane calculi pay more attention to the fidelity to the biological reality, have as a primary target systems biology, and use especially the framework of process algebra.”

A *membrane system* consists of a hierarchy of membranes which do not intersect, with a distinguishable membrane called *skin* surrounding all of them. A membrane without any other membranes inside is *elementary*, while a non-elementary membrane is a *composite* membrane. The membranes define demarcations between *regions*; for each membrane there is a unique associated region. Since we have a one-to-one correspondence, we sometimes use membrane instead of region, and vice-versa. The space outside the skin membrane is called the environment. Regions contain multisets of *objects*, *evolution rules* and possibly other membranes. Only rules in a region delimited by a membrane act on the objects in that region. More details about membrane systems can be found in [17].

Exocytosis is the movement of materials out of a cell via membranous vesicles. These processes allow patches of membrane to flow from compartment to compartment, and require us to think of a cell as a dynamic, rather than static, structure. *Endocytosis* is a general term for a group of processes that bring macromolecules, large particles, small molecules, and even small cells into the eukaryotic cell. There are three types of endocytosis: *pinocytosis*, *phagocytosis* and *receptor-mediated endocytosis.* In all three, the plasma membrane folds inward around materials from the environment, forming a small pocket. The pocket deepens, forming a vesicle. This vesicle separates from the plasma membrane and migrates with its contents to the cell interior.

In brane calculus we have a membrane structure, in which the membranes represent the sites of activity. Opposite to the initial classes of membrane systems in which a computation took place inside the membranes, in brane calculi a computation happens on the membrane. The operations of the two basic brane calculi are directly inspired by biologic processes such as endocytosis, exocytosis and mitosis. The calculus formed using *pino, exo, phago* operations is more expressive then the calculus formed by *mate, drip, bud*, because we can simulate the latter operations using the former ones. Another difference regarding the semantics is expressed in [6]: "whereas brane calculi are usually equipped with an interleaving, sequential semantics (each computational step consists of the execution of a single instruction), the usual semantics in membrane computing is based on maximal parallelism (a computational step is composed of a maximal set of independent interactions)."

Some work was done trying to relate these two models [6, 7, 10, 11]. Inspired by brane calculus, a model of the membrane system having objects attached to the membranes has been introduced in [9]. In [5], a class of membrane systems containing both free floating objects and objects attached to membranes have been proposed, while in [20] a simulation of a bounded symport antiport membrane system using brane calculus is proposed. In [2] we have related membrane computing with brane calculi, namely a translation of the PEP class into a special class of membrane systems, while in this paper we introduce a membrane system having objects and co-objects attached to the membrane.

The structure of the paper is as follows. In Section 2 we define the class of membrane systems with surface objects and co-objects together with the biological motivation for the rules used. In Section 3 we present the notions of competitions in membrane systems. Conclusions and references end the paper.

2 Mutual Membrane Systems with Surface Objects

The phospholipid bilayer serves as a lipid “lake” in which some proteins “float” (see Figure 1).

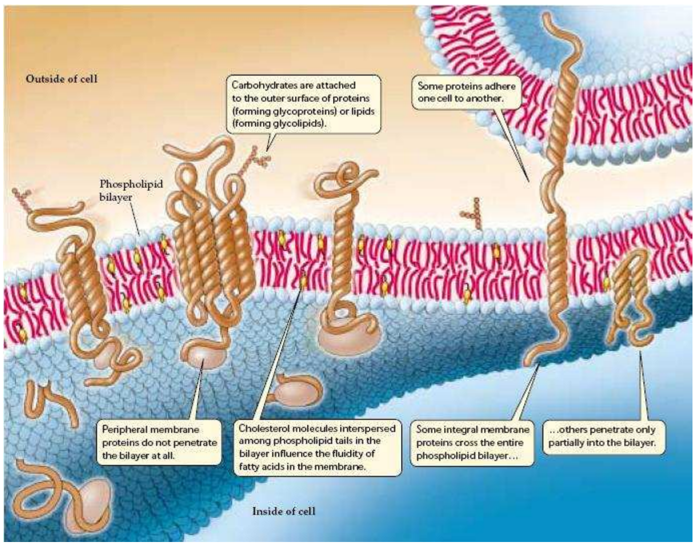

Outside of cell
Carbohydrates are attached to the outer surface of proteins (forming glycoproteins) or lipids (forming glycolipids).
Some proteins adhere one cell to another.
Phospholipid bilayer
Peripheral membrane proteins do not penetrate the bilayer at all.
Cholesterol molecules interspersed among phospholipid tails in the bilayer influence the fluidity of fatty acids in the membrane.
Some integral membrane proteins cross the entire phospholipid bilayer…
…others penetrate only partially into the bilayer.
Inside of cell

Fig. 1. The Fluid Mosaic Model: The general molecular structure of biological membranes is a continuous phospholipid bilayer in which proteins are embedded.

2.1 Endocytosis and Exocytosis in Biology

Endocytosis is a general term for a group of processes that bring macromolecules, large particles, small molecules, and even small cells into another cell. There are three types of endocytosis: phagocytosis, pinocytosis, and receptor-mediated endocytosis. In all three, the membrane invaginates (folds inward) around materials from the environment, forming a small pocket. The pocket deepens, forming a vesicle. This vesicle separates from the membrane and migrates with its contents to the cell’s interior.

In phagocytosis (“cellular eating”), part of the membrane engulfs large particles or even entire cells. Phagocytosis is used as a cellular feeding process by unicellular protists and by some white blood cells that defend the body by engulfing foreign cells and substances. In pinocytosis ("cellular drinking"), vesicles also form. However, these vesicles are smaller, and the process operates to bring small dissolved substances or fluids into the cell. Like phagocytosis, pinocytosis is relatively nonspecific as to what it brings into the cell.

Receptor-mediated endocytosis (Figure 2) is used by animal cells to capture specific macromolecules from the cell's environment. This process depends on receptor proteins, integral membrane proteins that can bind to a specific molecule in the cell's environment. The uptake process is similar to nonspecific endocytosis, as already described. However, in receptor-mediated endocytosis, receptor proteins at particular sites on the extracellular surface of the plasma membrane bind to specific substances. These sites are called coated pits because they form a slight depression in the plasma membrane. The cytoplasmic surface of a coated pit is coated by proteins, such as clathrin.

When a receptor protein binds to its specific macromolecule outside the cell, its coated pit invaginates and forms a coated vesicle around the bound macromolecule. Strengthened and stabilized by clathrin molecules, this vesicle carries the macromolecule into the cell.

Since only the receptor-mediated endocytosis uses receptors and co-receptors we are interested only in modeling this process.

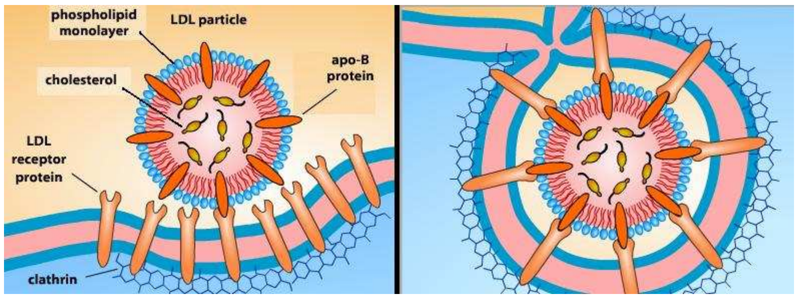


Fig. 2. Receptor-mediated endocytosis

Exocytosis (Figure 3) is the movement of materials out of a cell via membranous vesicles. These processes allow patches of membrane to flow from compartment to compartment, and require us to think of a cell as a dynamic, rather than static, structure. SNARES (Soluble NSF (N-ethylmaleimide Sensitive Factor) Attachment Protein Receptor) located on the vesicles (v-SNARES) and on the target membranes (t-SNARES) interact to form a stable complex that holds the vesicle very close to the target membrane.

The B lymphocyte cell searches for antigen matching its receptors. If it finds such antigen, it connects to it, and inside the B cell a triggering signal is set off. In order to become fully activated a B cell needs proteins produced by helper T cells. This is simulated using mutual contextual evolution (Figure 4).

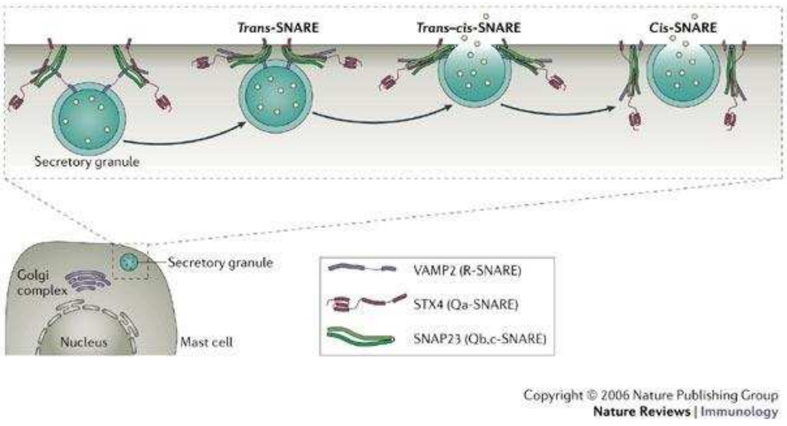


Fig. 3. SNARE-mediated exocytosis

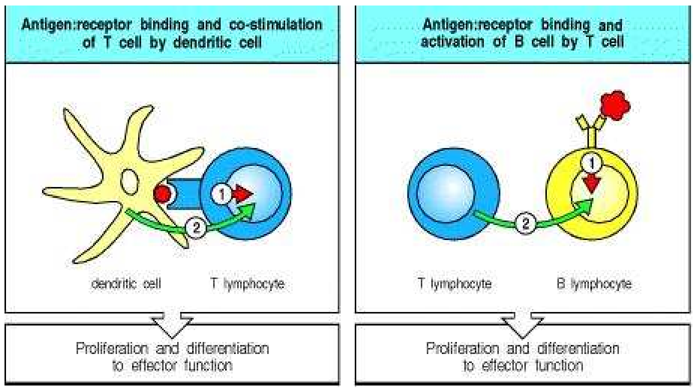

Antigen:receptor binding and co-stimulation of T cell by dendritic cell
1
2
dendritic cell
T lymphocyte
Proliferation and differentiation to effector function
Antigen:receptor binding and activation of B cell by T cell
1
2
T lymphocyte
B lymphocyte
Proliferation and differentiation to effector function

Fig. 4. Contextual evolution

This provides a biological motivation of using objects and co-objects for the exo, endo and contextual evolution rules.

We define now the membrane systems with surface objects and co-objects. Let $\mathbb{N}$ be a set of positive integers, and consider a finite alphabet Γ of symbols. A multiset over Γ is a mapping $u : \Gamma \to \mathbb{N}$. The empty multiset is represented by λ. For any $a \in \Gamma$, the value $u(a)$ denotes the multiplicity of a in u (i.e., the number of occurrences of symbol a in u). Given two multisets u, v over Γ, for any $a \in \Gamma$, we have $(u \uplus v)(a) = u(a) + v(a)$ as the multiset union, and $(u \backslash v)(a) = max\{0, u(a) - v(a)\}$ as the multiset difference. We use the string representation of multisets used in the membrane systems. An example of such a representation $u = aabca$, where $u(a) = 3$, $u(b) = 1$, $u(c) = 1$. Using such a representation, the operations over multisets are defined as operations over strings.

2.2 Definition

Definition 1. *A* mutual mobile membrane system with surface objects *is a construct*

$$\Pi = (P, \mu, w_1, \ldots, w_n, R, i_O)$$

where:

1. $n \geq 1$ *(the initial* degree *of the system);*
2. P *is an alphabet of proteins (its elements are called* objects*);*
3. i_O *is the output membrane;*
4. μ *is a* membrane structure*, consisting of n membranes. A membrane structure is a hierarchically arranged set of membranes, where we distinguish the external membrane (usually called the "skin" membrane) and several internal membranes; a membrane without any other membrane inside it is said to be elementary;*
5. $w_1, \ldots, w_n$ *are strings over V, describing the initial markings of the n membranes of μ;*
6. R *is a finite set of* developmental rules*, of the following forms:*
   a) $[\ ]_{uv}[\ ]_{\overline{u}v'} \rightarrow [[\ ]_w]_{w'}$ *for* $u, \overline{u}, w, w' \in P^+; v, v' \in P^*$

   mutual endocytosis

   An elementary membrane containing the multiset uv on the membrane enters the adjacent membrane containing the multiset $\overline{u}v'$ on the membrane; the multisets uv and $\overline{u}v'$ are transformed into multisets w and w' during the evolution;

   b) $[[\ ]_{uv}]_{\overline{u}v'} \rightarrow [\ ]_{uw}[\ ]_{\overline{u}w'}$, $u, \overline{u}, w, w' \in P^+; v, v' \in P^*$

   mutual exocytosis

   An elementary membrane containing the multiset uv on the membrane exits the adjacent membrane containing the multiset $\overline{u}v'$ on the membrane; the multisets uv and $\overline{u}v'$ are transformed into multisets w and w' during the evolution;

   c) $[\ ]_{uv}[\ ]_{\overline{u}v'} \rightarrow [\ ]_w[\ ]_{w'}$, $u, \overline{u}, w, w' \in P^+; v, v' \in P^*$

   mutual contextual evolution

   The multisets of objects uv and $\overline{u}v'$ placed on two sibling membranes are transformed into the multisets w and w'.

The rules are applied according to the following principles:

1. All rules are applied in parallel, non-deterministically choosing the rules, the membranes, and the objects, but in such a way that the parallelism is maximal; this means that in each step we apply a set of rules such that no further rule can be added to the set.
2. The membrane containing the multiset u on it from the rules of type $(a) - (b)$ is said to be active, while the membrane containing the multiset $\overline{u}$ on it is said to be passive. In any step of a computation, any object and any active membrane can be involved in at most one rule, but the passive membranes

are not considered involved in the use of the rules (hence they can be used by several rules at the same time as passive membranes).
3. When a membrane is moved across another membrane, by endocytosis or exocytosis, its whole contents (its objects) are moved.
4. If a membrane exits the system (by exocytosis), then its evolution stops.
5. All objects and membranes which do not evolve at a given step (for a given choice of rules which is maximal) are passed unchanged to the next configuration of the system.

3 Competitions

We start with an example from [19] which motivates biologically the study of competitions.

Example 1. Bacteriophage λ is a temperate phage, meaning that it can undergo either a lytic or a lysogenic cycle (see Figure 5). When there is a rich medium available and its host bacterium is growing rapidly, the prophage takes advantage of its favorable cellular environment and remains lysogenic. When the host bacteria are not as healthy, the prophage senses this and, as a survival mechanism, leaves the host chromosome and becomes lytic.

The phage makes this decision by means of a "genetic switch": Two regulatory viral proteins, labeled cI and Cro, compete for two operator/promoter sites on phage DNA. The two operator/promoter sites control the transcription of the viral genes involved in the lytic and the lysogenic cycles, respectively, and the two regulatory proteins have opposite effects on the two operators (Figure 2). Phage infection is essentially a "race" between these two regulatory proteins. In a healthy E. coli host cell, Cro synthesis is low, so cI "wins" and the phage enters a lysogenic cycle. If the host cell is damaged by mutagens or other stress, Cro synthesis is high, promoters for phage DNA and viral coat proteins are activated, and bacterial lysis ensues.

Remark 1. Rarely, two viruses infect a cell at the same time. This is an unusual event, as once an infection cycle is under way, there is usually not enough time for an additional infection. In addition, an early protein prevents further infections in some cases. In this case the protein is the one who prevents further races inside the infected cell.

In concurrency, a competition occurs when more than one rule are engaged for the same resources. Here is an example:

Example 2. Suppose at a given moment of computation we have a membrane containing the multiset of objects aa and two rules: $aa \to b$ and $aa \to c$. We observe that the rules have similar left objects. The application of one of the rules blocks the application of the other rule.

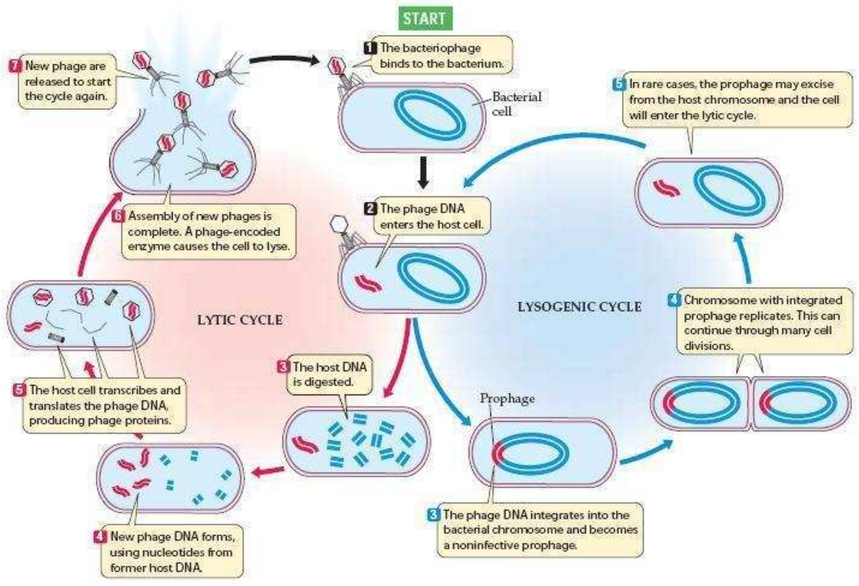

START
1 The bacteriophage binds to the bacterium.
Bacterial cell
2 The phage DNA enters the host cell.
3 The host DNA is digested.
4 New phage DNA forms, using nucleotides from former host DNA.
5 The host cell transcribes and translates the phage DNA, producing phage proteins.
6 Assembly of new phages is complete. A phage-encoded enzyme causes the cell to lyse.
7 New phage are released to start the cycle again.
LYTIC CYCLE
LYSOGENIC CYCLE
Prophage
3 The phage DNA integrates into the bacterial chromosome and becomes a noninfective prophage.
4 Chromosome with integrated prophage replicates. This can continue through many cell divisions.
5 In rare cases, the prophage may excise from the host chromosome and the cell will enter the lytic cycle.

Fig. 5. The Lytic and Lysogenic Cycles of Bacteriophage: Infection by viral DNA leads to the multiplication of the virus and lysis of the host bacterial cell. In the lysogenic cycle, an inactive prophage is replicated as part of the host's chromosome.

This kind of competitions are called *resource competitions*. Such a competition occurs when two rules try to use at least one common resource. Resource competitions are desirable in membrane computing for simulating non-deterministic behaviors. In mutual mobile membranes we also find a different form of competition, namely the *location competition*. Consider the following example:

Example 3. Suppose at a given moment of the computation we have the following membrane configuration: $[\ ][\ ]_a[\ ]_b$, and two rules: $[\ ][\ ]_a \rightarrow [[\ ]_c]$ and $[\ ][\ ]_b \rightarrow [[\ ]_d]$. We observe that membrane containing a is included in the left part of both rules. We have two possible evolutions:

- If we first apply the rule $[\ ][\ ]_a \rightarrow [[\ ]_c]$, we obtain the membrane structure $[[\ ]_c][\ ]_b$ and the other rule cannot be applied anymore.
- If we first apply the rule $[\ ][\ ]_b \rightarrow [[\ ]_d]$ we obtain the membrane structure $[\ ][[\ ]_d]_a$ and the other rule cannot be applied since now membrane n is not elementary. We refer to the rules of mobile membrane systems in which only elementary membranes can pass through other membranes [12].

The choice of which rule is applied first makes the other rule useless, so the system should be described initially with only one of these rules, depending on the expected evolution.

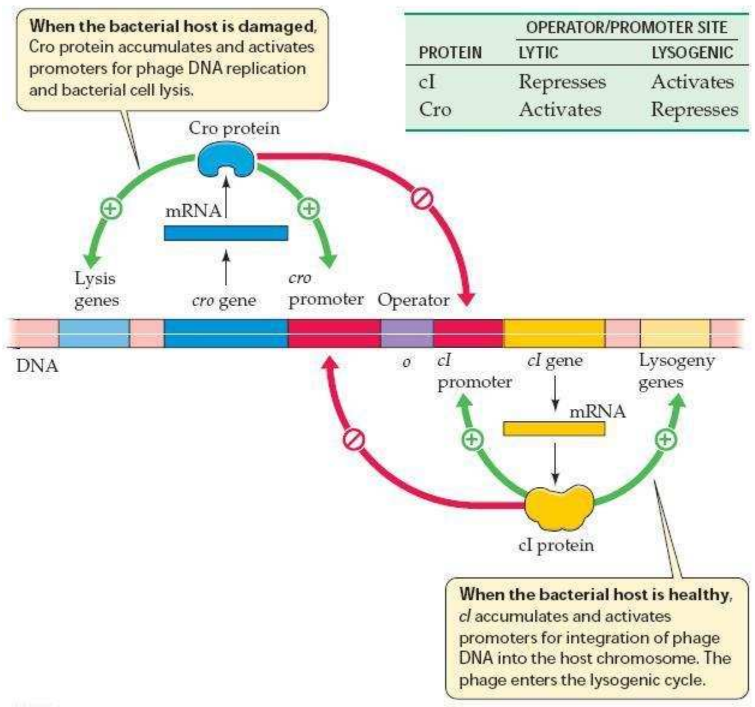

When the bacterial host is damaged, Cro protein accumulates and activates promoters for phage DNA replication and bacterial cell lysis.
OPERATOR/PROMOTER SITE
PROTEIN
LYTIC
LYSOGENIC
cI
Represses
Activates
Cro
Activates
Represses
Cro protein
mRNA
Lysis genes
cro gene
cro promoter
Operator
DNA
o
cI promoter
cI gene
mRNA
Lysogeny genes
cI protein
When the bacterial host is healthy, cI accumulates and activates promoters for integration of phage DNA into the host chromosome. The phage enters the lysogenic cycle.

Fig. 6. Control of Phage λ Lysis and Lysogeny: *Cro* and *cI* compete for the operator/promoter sites controlling the gene transcription for viral lysis and lysogeny.

The movement in mutual mobile membrane systems is *local*, namely a membrane can only interact with neighboring membranes. Locality of movement implies that a membrane written to accomplish a certain task in a given membrane, it shall not work correctly in another membrane.

The location competition raises some problem:

1. it is difficult to describe membrane systems which behave as expected in all contexts;
2. it is difficult to prove behavioral properties of membrane systems.

The rules of mobile membranes allow a membrane to enter, or to exit another membrane. The second membrane just undergoes the action, meaning that it has no control on when the movement takes place. As a consequence, it is hard to control the resources inside a given membrane. By defining mutual mobile membranes this is rectified: any movement takes place only if both participants agree. This is achieved by using objects and co-objects to control the movement. The inspiration comes from biology where we have receptors and co-receptors which control the interaction between membranes. Location competitions also exist in mutual mobile membranes, but they are easier to detect.

4 Conclusion

In the area of membrane computing the authors usually consider that a system is synchronous if the rules are applied in a maximally parallel manner, otherwise it is asynchronous. On the other hand, in process algebra the authors consider that two processes are synchronized if they interact by using actions and co-actions. We adapt the second approach to membrane systems, by replacing the actions and co-actions with objects and co-objects.

As related work we can mention [13] and [14] where the authors study the plain and grave interferences which appear in mobile ambients, and try to remove them by defining safe mobile ambients and an appropriate type system. The work presented in this paper corresponds to the first step described in their work, more exactly we define the competitions in mutual mobile membranes. Further work will include the use of a type system for mutual mobile membranes in order to limit the competitions which appear during the evolution of a membrane system.

Regarding the asynchronous aspects, we define in [4] a compositional asynchronous membrane system based on a handshake mechanism implemented by using antiport rules and promoters. Such a system is used to evaluate arithmetical expressions starting from simple membranes for addition, subtraction, multiplication and division.

References

1. B. Aman, G. Ciobanu: Structural properties and observability in membrane systems. *SYNASC*, IEEE Computer Society, 2007, 74–84.
2. B. Aman, G. Ciobanu: Membrane systems with surface objects. *Proceedings of the International Workshop on Computing with Biomolecules (CBM 2008)*, 2008, 17–29.
3. C. Bodei, A. Bracciali, D. Chiarugi: Control flow analysis for brane calculi. *Electronic Notes in Theoretical Computer Science*, 227 (2009), 59–75.
4. C. Bonchis, C. Izbasa, G. Ciobanu: Compositional asynchronous membrane systems. *Progress in Natural Science*, 17, 4 (2007), 411–416.
5. R. Brijder, M. Cavaliere, A. Riscos-Núñez, G. Rozenberg, D. Sburlan: Membrane systems with marked membranes. *Electronic Notes in Theoretical Computer Science,* 171, 2 (2007), 25–36.
6. N. Busi: On the computational power of the mate/bud/drip brane calculus: Interleaving vs. maximal parallelism. *Workshop on Membrane Computing*, LNCS 3850, Springer, 2006, 144–158.
7. N. Busi, R. Gorrieri: On the computational power of Brane calculi. *Third Workshop on Computational Methods in Systems Biology*, 2005, 106–117.
8. L. Cardelli: Brane calculi. Interactions of biological membranes. *Lecture Notes in BioInformatics*, 3082, Springer, 2004, 257–278.

9. L.Cardelli, Gh. Păun: An universality result for a (mem)brane calculus based on mate/drip operations. *ESF Exploratory Workshop on Cellular Computing (Complexity Aspects)*, Sevilla, 2005, 75–94.
10. M. Cavaliere, S. Sedwards: Membrane systems with peripherial proteins: Transport and evolution. *Electronic Notes in Theoretical Computer Science*, 171, 2 (2007), 37–53.
11. S.N. Krishna: Universality results for P systems based on brane calculi operations. *Theoretical Computer Science*, 371 (2007), 83–105.
12. S.N. Krishna, Gh. Păun: P systems with mobile membranes. *Natural Computing*, 4, 3 (2005), 255–274.
13. F. Levi, D. Sangiorgi: Controlling interference in ambients. *Principles of Programming Languages*, 2000, 352–364.
14. F. Levi, D. Sangiorgi: Mobile safe ambients. *Transactions on Programming Languages and Systems*, 25, 1 (2003), 1–69,.
15. R. Milner: *Communicating and Mobile Systems: The pi-calculus.* Cambridge University Press, 1999.
16. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
17. Gh. Păun: *Membrane Computing. An Introduction.* Springer, 2002.
18. Gh. Păun: Membrane computing and brane calculi (Some personal notes). *Electronic Notes in Theoretical Computer Science*, 171 (2007), 3–10.
19. W.K. Purves, D. Sadava, G.H. Orians, H.C. Heller: *Life: The Science of Biology*, 7th Edition, W.H. Freeman & Co, 2004.
20. A. Vitale, G. Mauri, C. Zandron: Simulation of a bounded symport antiport P system with brane calculi. *BioSytems*, 91, 3 (2008), 558–571.

Communication and Stochastic Processes in Some Bacterial Populations: Significance for Membrane Computing

Ioan I. Ardelean

Institute of Biology of the Romanian Academy
Centre of Microbiology
Splaiul Independentei 296, Bucharest, Romania
ioan.ardelean@ibiol.ro

Summary. Intercellular communication between bacterial cells belonging to the same population is well documented in Microbiology, sporulation and cannibalism in *B. Subtilis* and genetic competence and fratricide in *S. pneumoniae* being deeply studied in the last years. The investigation of individual cell behavior has revealed that populations of these bacteria sometimes bifurcate into phenotypically distinct, but genetically identical, subpopulations by random switching mechanisms. The probabilistic nature of the random switching mechanisms, the occurrence of some biochemical processes related to it at plasma membrane and the need to study the processes at the level of each individual cell make intercellular communication and stochastic processes very suitable to be modeled by P systems.

Motto: *But a system which has spherical symmetry, and whose state is changing because of chemical reactions and diffusion, will remain spherically symmetrical for ever.(...) It certainly cannot result in an organism such a horse, which is not spherically symmetrical.* [Turing, 1952]

1 Introduction

The concept of intercellular communication within a bacterial population belonging to the same species originates in the discovery of genetic competence in *Streptococcus pneumoniae* (1965) and of quorum sensing (1970) in *Vibrio* (Bassler and Losick, 2006). In the last decades, with special emphasis in the last years intercellular communication within a bacterial population started to be deeply documented in such a copious way that the scientific community started to introduce within their scientific language expressions such as “bacterially speaking”, competence and “fratricide”, “sporulation and cannibalism” (Bassler and Losick, 2006; Dubnau and Losick, 2006; Claverys and Havarstein, 2007) whereas few scientists put forward and claim that bacterial communication includes assignment of contextual meaning and sentences (semantic/syntax functions) and conduction of "dialogue" – the fundamental aspects of linguistic communication (Ben Jacob et al., 2006, and citations herein), the same authors seeking for the foundation of cognition in bacteria (Ben Jacob et al., 2006). (It is to be noticed that for dialogue the authors used the commas, whereas for cognition they did not.)

In this short report we focus on some of these new research on intercellular communication within a bacterial population because:

- Communication and stochastic processes become more deeply known when they started to be studied at the level of individual bacterial cell, trend belonging to the so-called single cell microbiology (SCM) which opens a new vision on bacterial world (Brehm-Stecher and Johnson, 2004; Kearns and Losick, 2005; Claverys and Havarstein, 2007); furthermore it was put forward that P systems could become a specific tool to study single bacterial cells as each cell contain a relative small number of important signalling molecules whose behavior could be better described by a discrete systems than by a continuous one (Ardelean, 2006).
- SCM investigation by improved techniques has revealed that populations of certain bacteria sometimes bifurcate into phenotypically distinct, but genetically identical, subpopulations, bifurcation which is called bistability (Dubnau and Losick, 2006). The need to study the processes in the each individual cell was originally put forward by Turing (Turing, 1952) who wrote: "To find the rate of change due to chemical reactions only needs to know the concentration of all morphogens at that moment *in the one cell concerned*" (my underline), a suggestion largely ignored (forgotten?) for decades.
- Bistability in our opinion could be appropriately modeled by P systems because of their probabilistic nature of the processes occurring within plasma membrane (as well as in a bulk phase).
- Bistability is a random mechanism that switches on different genetic programmes within identical bacteria grown under the same conditions (Dubnau and Losick, 2006). This passage from homogeneity to heterogeneity, this bifurcation both at the level of biochemical reaction and at the level of cell population remembers the bifurcation of chemical reaction (starting from o homogenous medium) mathematically first demonstrated by Turing in his interdisciplinary scientific paper on chemical basis of morphogenesis (Turing, 1952). This type of bifurcation named by Prigogine "Turing bifurcation" (Prigogine, 1977) is one of the tools used by Prigogine (Nobel Prize 1977) to physically explain how biological life (an anti-entropic process) is physically possible in an Universe whose overall entropy is under increase.

2 Communication and Stochastic Processes in Some Bacterial Populations

The recent advent of techniques like flow cytometry and fluorescence microscopy that facilitate the investigation of individual cell behavior has revealed that populations of certain bacteria sometimes bifurcate into phenotypically distinct, but genetically identical, subpopulations by random switching mechanisms. This bifurcation of genetically identical bacterial populations, also called clonal populations exhibit (unimodal) variation in the expression of a given gene, due to random fluctuations in the rates of synthesis and degradation of the cognate gene product, which is referred to as 'noise' and we will employ this usage. Sometimes, the noise gives rise to another type of variation that is non-unimodal, meaning that the population bifurcates into subpopulations, phenotypic phenomenon known as 'bistability' (Dubnau and Losick, 2006). For example, when *Bacillus Subtilis* cells encounter conditions of nutrient deprivation or reach a critical cell density, the cell can choose between two type of bifurcation involving entirely different genetic programmes, according to culture conditions. They can fully induce motility and enter stationary growth, enter sporulation, which cumulates in the formation of an enduring spore or enter the state of competence, in which they are able to take up DNA from the environment for integration into their chromosome via homologous recombination. Both programmes, competence and sporulation, involve the formation of a bistable culture; about 20% of cells will become competent, or a maximum of 80% of the cells will initiate sporulation. The remaining 80% or 20% of the cells, respectively, simply enter stationary phase and, in the case of sporulation, are even killed by the sporulating cells, which secrete a specific toxin, to serve as a nutrient source (Gonzalez-Pastor et al., 2003). Thus, even though all cells encounter identical culture conditions, only a (relatively well-defined) subpopulation fully throws the switch towards the new mode of development. Nevertheless, the non-competent or non-sporulating cells can switch to a new developmental state at a later time (Graumann, 2006). It could be important from P systems the fact that bistability arise stochastically in populations of genetically identical cells, grown in homogeneous and theoretically identical environments (e.g. in liquid media in well-stirred flasks), the choice of which individual cells exhibit altered gene expression being random (Dubnau and Losick, 2006). So far, there are several examples communication and stochastic processes in some bacterial populations but for the sick of simplicity in this paper we will briefly focus only on two of them which involve bistability: sporulation and cannibalism in *B. Subtilis* and genetic competence and fratricide in *S. pneumoniae.*

2.1 Sporulation and cannibalism

When the nutrients are limited many types of bacteria including *B. subtilis* entry into sporulation, an elaborate developmental process that culminates in the formation of a specialized cell called spore or endospore. The spore is a dormant cell type being able to resist environmental extremes: it is involved in the propagation in time and space not in the multiplication of the bacterial population to which it belongs. The master regulator for spore formation is Spo0A, a protein response regulator whose activity is governed by phosphorylation Spo0A is activated under conditions in which cells are limited for nutrients, but as demonstrated over a decade ago by flow cytometry some cells in a population of nutrient-limited cells activate the master regulator (Spo0A-ON cells) and some do not (Spo0A-OFF cells).

The results show that nutrient limitation is a prerequisite for entry into sporulation, but during nutrient limitation not all the bacterial cells starts the sporulation process because the activation of Spo0A is additionally subject to a bistable switch.

The scientists ask about the biological significance of subjecting entry into sporulation to bistability (Bassler and Losick, 2006; Claverys and Havarstein, 2007). One possible explanation comes from the fact that spore formation is an energy intensive process that becomes irreversible at an early stage. Thus, if the nutrient scarcity that triggers the activation of Spo0A in a population of cells proves to be fleeting, cells that have not entered the pathway to sporulate (Spo0A-OFF cells) will be able to rapidly resume growth when nutrients become available again. Studies of cells under conditions of high cell-population density reinforce the view that a mixed population of Spo0A-ON and Spo0A-OFF cells is a mechanism to cope with uncertainty in the future availability of nutrients. Furthermore, clonal colonies of *B. subtilis* cells are observed to exhibit a behavior referred to as cannibalism in which the Spo0A-ON cells in the population trigger the lysis of non-sporulating siblings (Spo0A-OFF cells) via the elaboration of a killing factor and a toxin (Gonzalez-Pastor et al., 2003) (Figure 1). Nutrients released from the non-sporulating siblings arrest or slow further progression into sporulation by the Spo0A-ON cells, impeding those cells from entering the irreversible phases of spore formation. Cannibalism is therefore a delaying tactic that helps the population to certify that lack of nutrients is not a fleeting condition. According to this view, the cost of fratricide is off set by the advantage of delaying commitment for as long as possible (Bassler and Losick, 2006; Claverys and Havarstein, 2007).

2.2 Competence and fratricide

The interplay between bacteria and antibiotics is very complex and very important for mankind both fundamentally and practically. It is universally accepted that the use of antibiotics will lead to antimicrobial resistance. Traditionally, the explanation to this phenomenon was based on : i) random mutation; ii) exchange of genetic information by horizontal gene transfer and iii) amplification by selective pressure. Subsequently, others mechanisms of antibiotic-induced antimicrobial resistance acquisition were proposed, based on the expected occurrence of bacterial transformation with DNA still present in antibiotic even after its purification (Woo et al., 2006) or on the fratricide behavior (Claverys and Havarstein, 2007). Bacteria

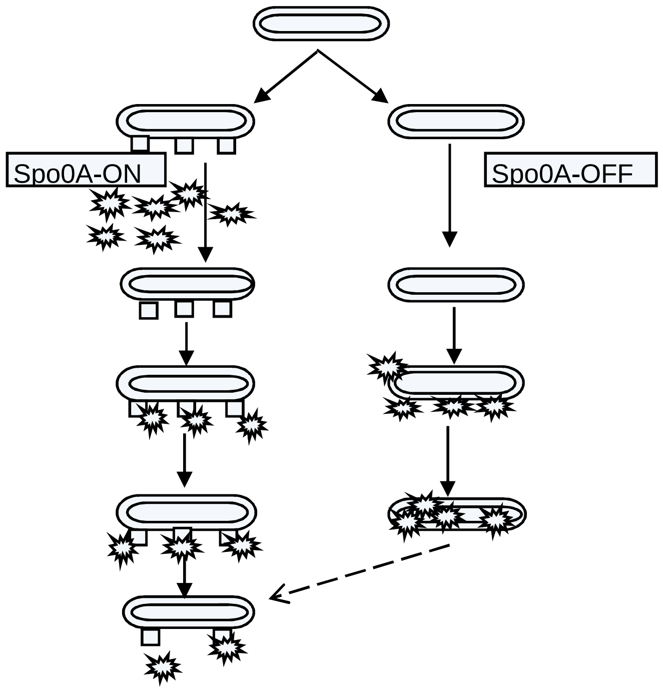


Fig. 1. Sporulation and canibalism in *Bacillus subtilis*. In the medium with nutrient limitation a clonal population of *Bacillus subtilis*, as an expression of bistability, bifurcate in two subpopulations: one in which the master regulator of sporulation is activated (Spo0A-ON) and the other one in which the master regulator of sporulation remains inactivated (Spo0A-OFF). The subpopulation entering the sporulation synthesize a toxin (killing factor - illustrated in the figure as a star) and the corresponding immunity device (illustrated in the figure as square located at the cell surface). The molecules of toxin destroy the nonsporulating cells whose chemical constituents become available (interrupted arrow in the figure) as nutrients for the cells in which the master regulator of sporulation is activated (Spo0A-ON) (modified after Claverys and Havarstein, 2007).

exchange genetic information either through the direct uptake of DNA (transformation), phage-mediated transduction, through inter-organism contact with DNA exchange (conjugation) or mobilization of DNA within organisms' genomes (transposition). Genetic transformation is possible when a cell has the ability to take up foreign DNA from the medium, ability which is named competence. Conjugation is one type of mechanism to transfer genes from one living bacterium to another living bacterium, the physical contact between the donor cell (called male type) and the recipient cell (called female type) being essential for gene transfer. Conjugation depends on the presence of certain plasmids, DNA closed molecules which are physically independent with respect to bacterial chromosome.

The ability of bacteria to sense some chemicals in the medium and to process this information is further illustrated by another bacterium *S. pneumoniae.* Transformation (the uptake and genomic integration of exogenous DNA) in *S. pneumoniae* can only occur when the bacteria are competent, which is a transitory state in bacteria. Although the competence state regulation is rather well understood, the signals that trigger it remain elusive. Recent evidence suggests that in *S. pneumoniae* competence is a stress response to environmental change (Prudhomme et al., 2006), who wondered whether antibiotic-induced stress might trigger competence. Out of the dozen or so antibiotics that they checked, six up-regulated the competence pathway when used at concentrations that killed approximately 50% of the bacteria. These antibiotics kill bacteria by either damaging DNA, inhibiting protein synthesis, or blocking DNA synthesis. The main conclusion is that the mechanism of action of a particular antibiotic cannot be used to predict its ability to induce competence further complicated the realities related to the generation of antibiotic-resistant bacteria; however the choose for clinical treatment of those antibiotics that do not promote genetic exchange may help to minimize future problems. These new findings (Prudhomme et al., 2006) which show that some bacteria become competent (able to take up foreign DNA from the external medium) as a response to the presence of an antibiotic in the external medium, further argue that bacteria posses a complex behavior. The molecular mechanism is not yet known but it could be speculated that it should involve signal transduction machinery which activity in other bacterial processes has already been modeled in the framework of P systems (Ardelean et al., 2006).

What it is really interesting is the fact that during the installation of competence, the competent cells synthesize a substance which is lethal for non-competent cells, which release in the growth medium their intracellular constituents, including nutrients and DNA. This type of killing is called fratricide and is a new type of bacterial behavior which further argue for the diversity of the interactions between bacterial cell and environment. The biological significance of fratricide related to environmental signal (e.g. antibiotic) induced competence is based on the hypothesis which invokes the provision of genetically diverse DNA molecules in the extra-cellular space to generate diversity by genetic transformation (Claverys and Havarstein, 2007). In *Enterococcus faecalis*, a Gram-positive species that commonly resides in the human intestine, it was discovered (Dunny et al. 1978) that gene transfer by conjugation can reach a very high frequency (10^2) compared with conjugative gene transfer in all other bacteria (frequency 10^4). It was shown that this significant difference is determined by the ability of some cells to synthesize and secrete in the extra-cellular medium a specific peptide. This peptide is called sex pheromone because it is involved in the attraction between donor and recipient bacterial cells, thus enhancing the frequency of conjugation. The pheromones are hydrophobic octa- or hepta-peptides and nowadays there are known more than four types, each pheromone being encoded by a gene located on the chromosome. All *Enterococcus faecalis* cells contain on their chromosome one type of gene for one type of pheromone but not all these gene are active. The pheromone gene is active only when in that particular bacterial cell the corresponding plasmid is missing, because the plasmid contains a gene whose product inhibit pheromone synthesis. Each type of pheromone acts as a sex pheromone on bacterial cells which have a plasmid specific for that pheromone, plasmid encoding the resistance to a given antibiotic. To simplify the biological notations, the bacterial cell able to synthesize the sex pheromone A will attract cells belonging to the same species (population) which have the corresponding plasmid alpha; this last type of cells can not synthesize the sex pheromone A whereas the cells able to synthesize the sex pheromone A have not the plasmid alpha. The pheromone response is characterized by an induced aggregation of donor and recipient (plasmid-free) cells, which can lead to mating frequencies greater than 10^2 per donor cell within a few hours. In a liquid environment, pheromone induces donor cells to synthesize a plasmid-encoded "aggregation substance" (AS), a surface protein that binds to recipients and initiates the contact necessary for transfer of plasmid DNA. However, the pheromone induces the synthesis of several additional plasmid-encoded products necessary for DNA transfer by conjugation (Clewell, 1989; 2004). When a recipient strain acquires a copy of the plasmid there is a "shutdown" of the corresponding pheromone activity, as transconjugant themselves become donors. When a former recipient cell receive the plasmid alfa it stops the synthesis of A pheromone, but not the synthesis of other type of pheromones for which there are no corresponding plasmids in the cell. Thus, transconjugants (the former recipient cell), continue to secrete other different pheromones specific for donors carrying conjugative plasmids which confer resistance to other type of antibiotics. Gene transfer by conjugation is essential for those bacterial cells, devoid of a given plasmid carrying the gene for the resistance to a given (class) of antibiotics, to survive in media where antibiotics are present; this is the case of human body under clinical treatment, *Enterococcus faecalis* being one of the most common bacteria involved in nosocomial (hospital-acquired) infections and are notorious for being resistant to multiple antibiotics. The production of sex pheromones is another type of example of chemical communication between bacterial cells, enabling them to announce other cells that they are ready to receive foreign genetic material conferring them resistance to antibiotic 1, (especially?) when the recipient cells are in the presence of antibiotic 1 to which they are sensitive. Furthermore, the increase in the frequency of conjugation could be one possible explanation on those results concerning antibiotic-induced enterococcal expansion in the mouse intestine which correlates poorly with suppression of competing bacteria already present in the intestine as "normal" bacteria, suggesting that other factors favor the adherence and multiplication of *E. faecium* in the gastrointestinal tract of antibiotic-treated mammals (Woo et al., 2006). This biological phenomenon could probably be modeled by P systems with so- called query symbols in the string (E. Csuhaj-Varju and G. Vaszil, personal communication).

In my opinion, the recently discovered bistability in some bacterial populations is an example of biochemical bifurcation, which was first studied as a basic process in living cells more than half a century ago by Turing (Turing, 1952). This bifurcation would deserve further mathematical approach. The probabilistic nature of bacterial bistability, the occurrence of some biochemical processes related to bistability at plasma membrane and the need to study the processes at the level of each individual cell make bistability very suitable to be modeled by P systems. The biologists involved in the study of intercellular communication and stochastic processes in some bacterial populations as it is illustrated by bistability could benefit from the collaboration with scientists working on P systems to start the mathematical modeling of these discrete biological processes and to understand what kind of biological experiments are still needed to further reach the full power of modeling and calculation of P systems.

References

1. I. Ardelean: Biological roots and applications of P systems. Further suggestions. *Membrane Computing, WMC 2006* (H.J. Hoogeboom et al., eds.), LNCS 4361, Springer, 2006, 1-17.
2. B.L. Bassler, R. Losick: Bacterially speaking. *Cell*, 125 (2006), 237–246.
3. I. Ardelean, D. Besozzi, M.H. Garzon, G. Mauri, S. Roy: P system models for mechanosensitive channels. *Applications of Membrane Computing* (G. Ciobanu, Gh. Păun, M.J. Pérez-Jiménez, eds.), Springer, 2006, 43–81.
4. R.M. Berka, J. Hahn, M. Albano, I. Draskovic, M. Persuh, X. Cui, et al.: Microarray analysis of the *Bacillus subtilis* K-state: genome-wide expression changes dependent on ComK. *Mol. Microbiol.*, 43 (2002), 1331–1345.
5. B.F. Brehm-Stecher, E.A. Johnson: Single-cell microbiology: tools, technologies, and applications. *Microbiol. Mol. Biol. Rev.*, 68 (2004), 538–559.
6. J.P. Claverys, L.S. Havarstein: Cannibalism and fratricide: mechanism and raisons d'être. *Nature Microbiology Reviews*, 5 (2007), 229–239.
7. D.B. Clewell, M.V. Francia: Conjugation in Gram-positive bacteria. *Plasmid Biology* (B. Funnell ed.), American Society for Microbiology, Washington, DC, 2004, 227–256.
8. D.B. Clewell, K.E. Weaver: Sex pheromones and plasmid transfer in *Enterococcus faecalis*. *Plasmid*, 21 (1989), 175–184.
9. E. Csuhaj-Varju, G. Vaszil: Personal communication, 2009.
10. D. Dubnau, R. Losick: Bistability in bacteria. *Mol. Microbiol.*, 61 (2006), 564–572.
11. G.M. Dunny, B.L. Brown, D.B. Clewell: Induced cell aggregation and mating in *Streptococcus faecalis*: evidence for a bacterial sex pheromone. *Proc. Natl. Acad. Sci. USA*, 75 (1978), 3479–3483.
12. J. Dworkin, R. Losick: Developmental commitment in a bacterium. *Cell*, 121 (2005), 401–409.
13. M.V. Francia, K.E. Weaver, P. Goicoechea, P. Tille, D.B. Clewell: Characterization of an active partition system for the *Enterococcus faecalis* pheromone-responding plasmid pAD1. *J. Bacteriol.*, 189 (2007), 8546–8555.
14. J.E. Gonzalez-Pastor, E.C. Hobbs, R. Losick: Cannibalism by sporulating bacteria. *Science*, 301 (2003), 510–513.
15. P.L. Graumann: Different genetic programmes within identical bacteria under identical conditions: the phenomenon of bistability greatly modifies our view on bacterial populations. *Molecular Microbiology*, 61, 3 (2006), 560–563.

16. E.B. Jacob, Y. Shapira Y., Touber, A. I. 2006. Seeking the foundation of cognition in bacteria: From Schrdinger's negative entropy to talent information Physica A 359, 495-524.
17. D.B. Kearns, R. Losick: Cell population heterogeneity during growth of *Bacillus subtilis*. *Genes Dev.*, 19 (2005), 3083–3094.
18. I. Prigogine: *Time, Structure and Fluctuations*. Nobel Lecture, 8 December 1977, `www.nobel.org`.
19. M. Prudhomme, L. Attaiech, G. Sanchez, B. Martin J.-P. Claverys: Antibiotic stress induces genetic transformability in the human pathogen *Streptococcus pneumoniae*. *Science*, 313 (2006), 89–92.
20. A.M. Turing: Thechemical basis of moprhogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237, 641 (1952), 37–72.
21. P.C.Y. Woo, A.P.C. To, S.K.P. Lau, K.Y. Yuen: Facilitation of horizontal transfer of antimicrobial resistance by transformation of antibiotic-induced cell-wall-deficient bacteria. *Medical Hypothesis*, 61 (2003), 503–508.

P Systems with Endosomes

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

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

Summary. 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 examples.

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, others are sent *out* of the membrane, others are sent *into* the inner membranes, which are identified by their labels. 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 environment 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 [2, 11] for the definition of these (and other) variants of P Systems and [13] 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 these features:

- objects can be contained both inside and on the surfaces of the membranes (as in P Systems with peripheral proteins [5, 9]);
- 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 of the body because most substances important to them are large polar molecules that cannot pass through the hydrophobic plasma membrane or cell membrane. There exist three kind 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. Figure 1[1] summarizes the kinds of endocytosis. By 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 kind 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 [1], would provide a powerful and complete modeling language for naturally describing transportation mechanism of molecules inside cells.

[1] Pictures taken from http://cellbiology.med.unsw.edu.au/units/science/lecture0806.htm

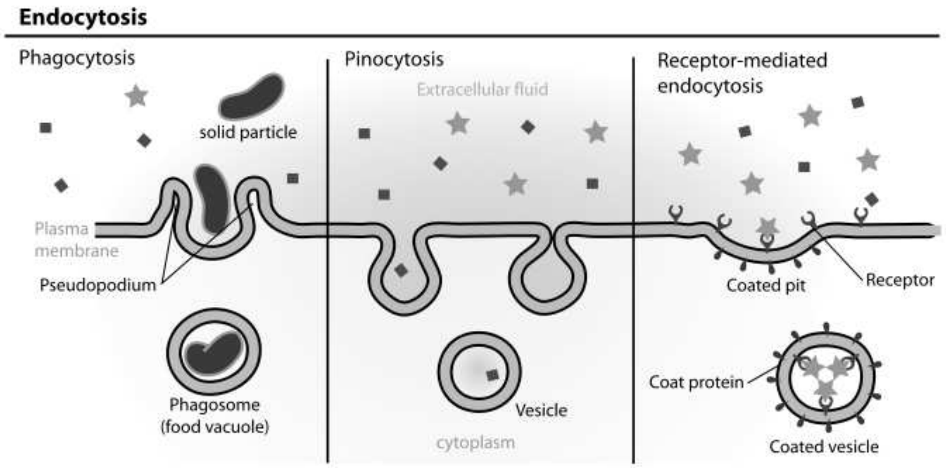


Fig. 1. Three kind of endocytosis: *phagocytosis*, *pinocytosis* and *receptor–mediated endocytosis*.

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 examples 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 recall the definition of standard P Systems, and then we define their extension with endosomes. We will denote multisets over a finite alphabet as strings of alphabet symbols. More precisely, let V^* be the set of all strings over an alphabet V, including the empty one, denoted by λ. For $a \in V$ and x in V^* we denote by $|x|_a$ the number of occurrences of a in x. If $V = \{a_1, \ldots, a_n\}$ (the ordering is important here), then the Parikh mapping of x is defined by $\Psi_V(x) = (|x|_{a_1}, \ldots, |x|_{a_n})$. The definition is extended in the natural way to languages. A string x represents the multiset over V with the multiplicities of objects $a_1, \ldots, a_n$ as given by $\Psi_V(x)$.

2.1 P Systems

A P System consists of a *hierarchy of membranes* that do not intersect, with a distinguishable membrane, called the *skin membrane*, surrounding them all. As usual, we assume membranes to be labeled by natural numbers. Given a set of objects V, a membrane m contains a multiset of *objects* in V^*, a set of *evolution rules*, and possibly other membranes, called *child* membranes (m is also called the *parent* of its child membranes). Objects represent molecules swimming in a chemical solution, and evolution rules represent chemical reactions that may occur inside the membrane containing them. For each evolution rule there is a multiset of objects representing the reactants, and a multiset of objects representing the products of the chemical reaction. A rule in a membrane m can be applied only to objects in m, meaning that the reactants should be precisely in m, and not in its child membranes. The rule must contain target indications, specifying the membranes where the new objects produced by applying the rule are sent. The new objects either remain in m, or can be sent out of m, or can be sent into one of its child membranes, precisely identified by its label. Formally, the products of a rule are denoted with a multiset of *messages* of the forms:

- $(v, here)$, meaning that the multiset of objects v produced by the rule remain in the same membrane m;
- (v, out), meaning that the multiset of objects v produced by the rule are sent out of m;
- (v, in_l), meaning that the multiset of objects v produced by the rule are sent into the child membrane l.

Let TAR be the set of message targets $\{here, out\} \cup \{in_i \mid i \in \mathbb{N}\}$. Given a set of objects O we denote with O_{tar} the corresponding set of messages $O \times TAR$. As a consequence, we denote with V_{tar} the set of all messages and we can define an evolution rule as a rule $u \to v$ such that $u \in V^*$ and $v \in V^*_{tar}$.

The size of the left–hand side u of an evolution rule is also called the *radius* of such a rule. If a P System contains rules of radius greater than one, then it is called a *cooperative* system. Otherwise, it is called *non–cooperative.*

Application of evolution rules is done with maximal parallelism, namely at each evolution step a multiset of instances of evolution rules is chosen non–deterministically such that no other rule can be applied to the system obtained by removing all the objects necessary to apply all the chosen rules.

A P System has a tree–structure in which the skin membrane is the root and the membranes containing no other membranes are the leaves. We assume membranes labels to be unique. A membrane structure can be represented as a balanced sequence of labeled brackets and, graphically, as a Venn diagram.

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

- V *is a finite* alphabet *whose elements are called* objects*;*
- $\mu \subset \mathbb{N} \times \mathbb{N}$ *is a* membrane structure*, such that* $(i, j) \in \mu$ *denotes that the membrane labeled by* j *is contained in the membrane labeled by* i*;*
- w_i *with* $1 \leq i \leq n$ *are strings from* V^* *representing multisets over* V *associated with the membranes* $1, 2, \ldots, n$ *of* μ*;*
- R_i *with* $1 \leq i \leq n$ *are finite sets of* evolution rules *associated with the membranes* $1, 2, \ldots, n$ *of* μ*;*

A sequence of transitions between configurations of a given P System Π is called a *computation.* A computation is *successful* if and only if it reaches a configuration in which no rule is applicable. The result of a successful computation is the multiset

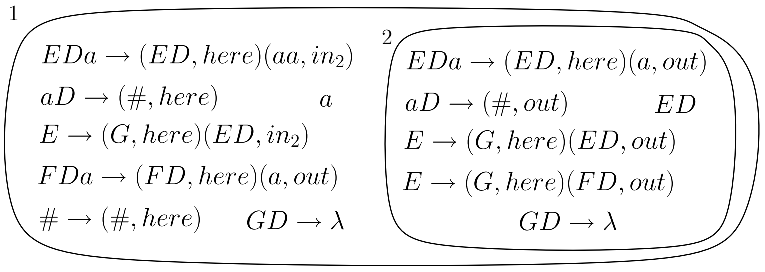

1
$EDa \rightarrow (ED, here)(aa, in_2)$
$aD \rightarrow (\#, here)$
a
$E \rightarrow (G, here)(ED, in_2)$
$FDa \rightarrow (FD, here)(a, out)$
$\# \rightarrow (\#, here)$
$GD \rightarrow \lambda$
2
$EDa \rightarrow (ED, here)(a, out)$
$aD \rightarrow (\#, out)$
ED
$E \rightarrow (G, here)(ED, out)$
$E \rightarrow (G, here)(FD, out)$
$GD \rightarrow \lambda$

Fig. 2. Example of P System that computes $\{a^{2^n} \mid n \in \mathbb{N}\}$.

of objects sent out of the skin membrane during the computation. Unsuccessful computations (computations which never halt) yield no result. Given a P System Π whose set of object is V, the result $x \in V^*$ of a computation of Π can be represented as the vector of natural numbers $\Phi_V(x)$. The set of all vectors of natural numbers computed by Π is denoted $Ps(\Pi)$.

In Fig. 2 we show an example of P System Π_1 computing $\{a^{2^n} \mid n \in \mathbb{N}\}$, namely such that $Ps(\Pi_1) = \{2^n \mid n \in \mathbb{N}\}$. Initially, only in membrane 2 there are rules which are applicable and send either objects F and D or objects E and D into membrane 1. In the former case the object a in membrane 1 is sent out and the computation halts. In the latter case object a in membrane 1 is consumed and two occurrences of a are sent into membrane 2. Subsequently, E is consumed and sent into membrane 2 together with D. Note that the rule which sends ED into membrane 2 cannot be applied while there are still objects a in membrane 1, otherwise by the maximal parallelism also the rule producing $\#$ would be applied giving rise to an infinite (unsuccessful) computation. Objects a sent into membrane 2 are then sent back into membrane 1. The process of doubling and sending into membrane 2 we have explained, could be repeated an arbitrary number of times. Note that all the rules consuming a act on a single occurrence of a at a time and hence the time complexity of the computation is proportional to 2^{n+2}.

2.2 Extension with Endosomes

In this section we formally define P Systems with endosomes (PE Systems). To this extent, we start by assuming the same membrane structure μ of a P System. As regards objects, similarly to P Systems with peripheral proteins [5, 9], 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 a 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 O we denote with O_{tar} the corresponding set of messages $O \times TAR$, and we denote with V_{tar} the set of all messages.

We can now introduce the evolution rules of PE Systems; rules are conceptually divided in evolutionary 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. The former class of rules are of the form $u \to v$ where $u \in V_{tar}^{+}$ and $v \in V_{tar}^{*}$. The definition of cooperative and non–cooperative rules are the same as for P Systems.

Notice that this format for evolutionary 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 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 behavior can be obtained by replacing the rule $u \to (v, in_{l'})$ in the membrane l, as in usual P Systesm, with the PE System rule $(u, out) \to (v, in)$ on the surface of the membrane l'. The behavior 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 result 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 $endo_E(u \in V^*, v \in V^*)$ where:

- E is a set of evolutionary 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.

Notice that each endosome has got its own evolutionary rules in set E. These rules model the behavior 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 (in some sense they can be seen as the receptors) and that objects in v are present outside of the membrane creating the endosome (in some sense they can be seen as the molecules to be engulfed). More formally, the applicability of the 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, than 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 a 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 2. *A* PE System *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*, such that* $(i, j) \in \mu$ *denotes that the membrane labeled by* j *is contained in the membrane labeled by* i*;*
- 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 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. As a consequence, before giving the result and its proof, we recall from [11] the definition of such 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 the 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 $F = \varnothing$) is denoted by MAT^{λ}, when λ–rules are allowed, and by MAT, when such rules are not allowed. It is known that (i) $MAT \subset MAT_{ac} \subset CS$; (ii) $MAT^{\lambda} \subset MAT^{\lambda}_{ac} = RE$, 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, 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, F consists exactly of all rules $A \to \#$ appearing in matrices of type 3 and $ac(G) \leq 2$. 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 the 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 the 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 $k + 1 \leq i \leq n$. For $j \in \{1, 2\}$, we denote with ℓ_j the set $\{i \mid \text{ the matix } m_i \text{ contains the rule } B^{(j)} \to \#\}$. For uniformity, we also denote $\ell_0 = \{1, 2, \ldots, k\}$ and $\ell = \ell_0 \cup \ell_1 \cup \ell_2 = \{1, 2, \ldots, n\}$ (note that $0 \notin \ell$). Clearly, the sets ℓ_0, ℓ_1 and ℓ_2 are disjoint.

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 PMC 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 1. *$PsE_2(ncoo) = PsRE$.*

Proof. It is enough to show that for a 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 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. All the objects encoding the grammar will be stored inside membrane 1 and the matrixes 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 $z_2 = \{c\}$.

This PE System works as follows: it has a cyclic behavior 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, than the computation has not to halt. In the case in which it starts the interpretation 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 applying the corresponding matrix to the non terminals. Subsequently, these objects are sent out to membrane 1 to restart the cyclic behavior. We recall that during this process no other endosomes can be created, so no other matrixes 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 matrixes of type 2 and 4 even though, at the end of the application of this matrix, instead of restarting with the cyclic behavior, 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 it is not found, an endosome will restore the configuration of this PE System so that the cyclic behavior 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. $\{(a, here) \to (a, out) \mid \forall a \in T\}$. All terminal objects in membrane 1 are sent out as output.

All the interpretation of any matrix is done by the rules of the membrane 2 which are the following:

1. $\forall X \in N_1 \cup N_2$. $endo_{(X,in)\to(\#,out)}(c, X)$. 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. $\forall a \in N_1 \cup N_2 \cup T$. $(a, in) \to (a, out)$. Every terminal and non terminal present inside membrane 2 is sent out to membrane 1.
3. $(c, in) \to (c, here)$. Object c inside membrane 2 is restored on the surface of membrane 2 so that other endosomes can be created.
4. $(\#, in) \to (\#, in)$. The trap symbol lets this computation not to be recognized such.
5. $\forall(X \to \alpha, A \to x) \in \{m_i \mid 1 \leq i \leq k\}$.
$endo_{(X,in)\to(\alpha,out),(A,in)\to(x,out),(c,here)\to(c,out)}(c, XA)$. 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. $\forall(X \to Y, A \to \#) \in \{m_i \mid k+1 \leq i \leq n\}$.
$endo_{(X,in)\to(Y,out),(c,here)\to(c_i,out)}(c, X)$. 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 the objects are sent out to membrane 2.
7. $(c_i, in) \to (c_i, here)(d_i, here)$. Object c_i, together with a new object d_i, is moved on the surface of membrane 2.
8. $endo_{(B^{(i)},in)\to(\#,out)}(c_i, B^{(i)})$. This implements the appearance checking 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. $(d_i, here) \to (d'_i, here)$. 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 behavior.
10. $endo_{(c_i,here)\to(c,out),(d'_i,here)\to\lambda}(c_i d'_i, \emptyset)$. This endosome lets Π_G restart its cyclic behavior. 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

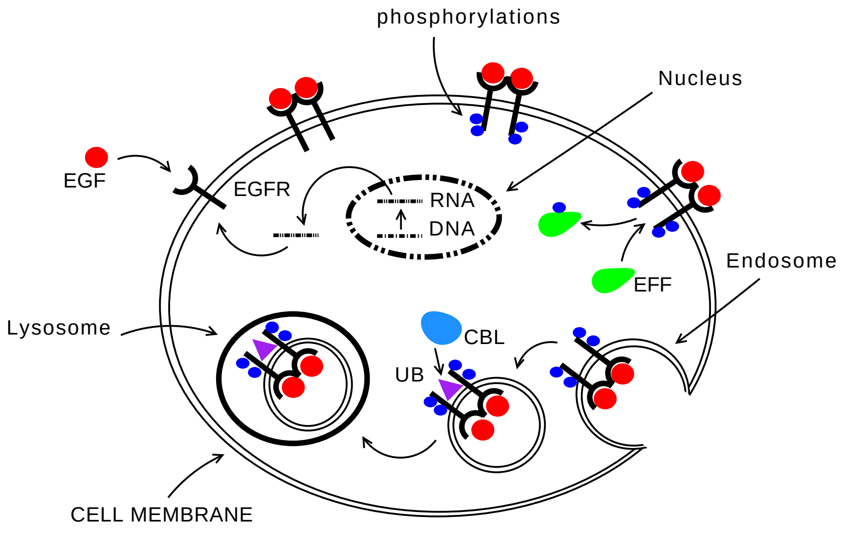


Fig. 3. The EGF signaling pathway.

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.

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 protein 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 behavior.

A complex signal transduction cascade, that modulates cell proliferation, survival, adhesion, migration and differentiation, is based on a family of receptors

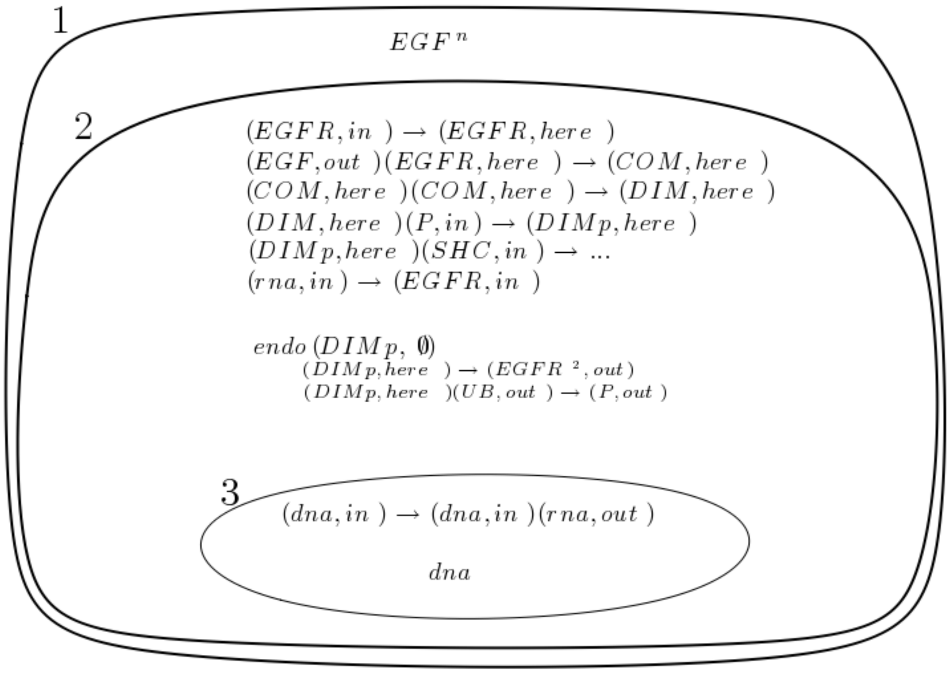

1
EGF^n
2
$(EGFR, in) \rightarrow (EGFR, here)$
$(EGF, out)(EGFR, here) \rightarrow (COM, here)$
$(COM, here)(COM, here) \rightarrow (DIM, here)$
$(DIM, here)(P, in) \rightarrow (DIMp, here)$
$(DIMp, here)(SHC, in) \rightarrow ...$
$(rna, in) \rightarrow (EGFR, in)$
$endo\,(DIMp,\ \emptyset)$
$(DIMp, here) \rightarrow (EGFR^2, out)$
$(DIMp, here)(UB, out) \rightarrow (P, out)$
3
$(dna, in) \rightarrow (dna, in)(rna, out)$
dna

Fig. 4. A PE systems model of the EGF signaling pathway.

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 3).

The PE system modeling the EGF is given in Figure 4. The rules which describe its behaviour are the following for membrane 2:

1. $(EGFR, in) \rightarrow (EGFR, here)$
2. $(EGF, out)(EGFR, here) \rightarrow (COM, here)$

3. $(COM, here)(COM, here) \rightarrow (DIM, here)$
4. $(DIM, here)(P, in) \rightarrow (DIMp, here)$
5. $(DIMp, here)(SHC, in) \rightarrow ...$
6. $(rna, in) \rightarrow (EGFR, in)$
7. $endo_{(DIMp,here)(UB,out)\rightarrow(P,out),(DIMp,here)\rightarrow(EGFR^2,out)}(DIMp, \emptyset)$

Notice that rule 5 is not complete in the sense that it should rewrite the $DIMp$ on the surface of the membrane and the contained SHC into a complex in order to start a sequence of intra–cellular events leading to the duplication of the cell. However, for the explanatory aim of this application example, it is not of interest to fully describe this part of the model.

Finally, membrane 3 contains just one rule $(dna, in) \rightarrow (dna, in)(rna, out)$.

The behavior of this PE System is straightforward, membrane 1 models the environment external 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 start a chain of actions we do not model here. Furthermore, it can be enclosed in an endosome which could, in presence of ubiquitin UB, reproduce the phosphorus inside the cell or, differently, the two receptors. 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 presented an extension of P Systems, called P Systems with Endosomes (PE Systems), in which endosomes can be explicitly modeled. 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. PE Systems extend P Systems maintaining the main features of the formalism but adding the possibility of explicitly modeling endosomes. Modeling endosomes is the basis for modeling vesicle–mediated transportation mechanisms, in particular endocytosis, which can be divided in three main forms (pynocytosis, phagocytosis and receptor–mediated endocytosis) can be clearly modeled by using PE Systems. 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 the works on P System with peripheral proteins [5, 9]. Furthermore, as regards other calculi, operations for modeling transportation mechanism already have been introduced in Brane Calculi [3]. Although similar, PE Systems permit to model in a clearer way these mechanisms. An analysis of PE Systems and Brane Calculi [3] (and also some of their variants like projective Brane Calculi [6]) could be done along the line of the one done in [4, 12] for P Systems and Brane Calculi.

As regards expressiveness of this formalism, we showed 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 of the initial phases of the EGFR signalling cascade.

References

1. R. Barbuti, A. Maggiolo–Schettini, P. Milazzo, S. Tini: P systems with transport and diffusion membrane channels. *Int. Workshop on Concurrency, Specification and Programming* (CS&P'08), Gross Vaeter, Germany, September, 2008.
2. P. Bottoni, C. Martin–Vide, Gh. Păun, G. Rozenberg: Membrane systems with promoters/inhibitors. *Acta Informatica*, 38 (2002), 695–720.
3. L. Cardelli: Brane calculi. Interactions of biological membranes. In V. Danos, V. Schachter, eds., LNCS 3082, 257–280.
4. L. Cardelli, Gh. Păun: An universality result for a (mem)brane calculus based on mate/drip operations. *Internat. J. Found. Comput. Sci.*, 17, 1 (2006), 49–68.
5. M. Cavaliere, S. Seawards: Membrane systems with peripheral proteins: transport and evolution. *ENTCS*, 171 (2007), 37–53.
6. V. Danos, S. Pradalier: Projective brane calculus. LNCS 3082, 134–148.
7. R. Freund, M. Oswald: P systems with activated/prohibited membrane channels. *Proc. of WMC 2002*, LNCS 2597, 2003, 261–269.
8. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20 (2002), 295–306.
9. A. Păun, B. Popa: P systems with proteins on membranes. *Fundamenta Informaticae*, 72, 4 (2006), 467–483.
10. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143
11. Gh. Păun: *Membrane Computing. An Introduction.* Springer 2002.
12. Gh. Păun: Membrane computing and brane calculi. Old, new, and future bridges. *Theoretical Computer Science*, 404, 1-2 (2008), 19–25.
13. P Systems web page. `http://ppage.psystems.eu/`.

P System Based Model of an Ecosystem of the Scavenger Birds

Mónica Cardona[1], M. Angels Colomer[1],
Antoni Margalida[4], Ignacio Pérez-Hurtado[2],
Mario J. Pérez-Jiménez[2], Delfí Sanuy[3],

[1] Dpt. of Mathematics, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
{mcardona,colomer}@matematica.udl.es

[2] Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence, University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

[3] Dpt. of Animal Production, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 Lleida, Spain
dsanuy@prodan.udl.cat

[4] Bearded Vulture Study & Protection Group
Adpo. 43 E-25520 El Pont de Suert (Lleida), Spain
margalida@inf.entorno.es

Summary. The Bearded Vulture (*Gypaetus Barbatus*) is an endangered species in Europe that feeds almost exclusively on bone remains provided by wild and domestic ungulates. In [1], we presented a P system in order to study the evolution of these species in the Pyrenees (NE Spain). Here, we present a new model that overcomes some limitations of the previous work incorporating other scavenger species (predatory) and additional prey species that provide food for the scavenger intraguild and interact with the Bearded Vulture in the ecosystem. After the validation, the new model can be a useful tool for the study of the evolution and management of the ecosystem. P systems provide a high level computational modelling framework which integrates the structural and dynamical aspects of ecosystems in a compressive and relevant way. The inherent stochasticity and uncertainty in ecosystems is captured by using probabilistic strategies.

1 Introduction

Since nature is very complex, the perfect model that explain it will be complex too. A complex model is not practical or good to use, so we should obtain a simple model that keeps the most important natural factors and consequently will be useful.

The P system presented in [1] gives good results in order to study the evolution of the ecosystem based on the Bearded Vulture in the Catalan Pyrenees, but it does not take into account important factors such as the population density or the feeding limitations.

In the Catalan Pyrenees, in the North-east of Spain, three vulture species inhabits sharing the geographic space and the existent food resources so in this work, we present a P system for modelling an ecosystem based on three vulture species and the prey species present from which scavengers obtain their food. The present model improves the results presented in [1]. Apart from adding two new predator species (the Egyptian Vulture *Neophron percnopterus* and Eurasian Griffon Vulture *Gyps fulvus*), we introduce new prey species (making a total of 12 species) in the new model that provide feeding resources for the scavenger community. Besides, new rules are introduced to limit the maximum amount of animals that can be supported by the ecosystem as well as the amount of grass available for the herbivorous species.

For a good and efficient management of the ecosystems, it is necessary to know the quantity and the evolution of the biomass that reaches every species. One of the contributions the P system presents is the evolution of the quantity of annual biomass that is left by every species.

The paper is organized as follows: First, an ecosystem, which is located in the Catalan Pyrenees and is to be modelled, is described. Section 3 shows a formal framework to model ecosystems by means of probabilistic P systems, and a P system modelling the above mentioned ecosystem is presented. In Section 4, we discuss the results obtained and we compare the model presented in this paper to the one presented in [1], using a P-lingua simulator [4].

2 Modelling the Ecosystem

The ecosystem to be modelled is located in the Catalan Pyrenees (NE Spain). This area contains a total of 36 breeding territories of the Bearded Vulture, 65 of the Egyptian Vulture and 525 of the Eurasian Griffon Vulture.

In addition to the three vultures, the ecosystem to be modelled is also composed of 9 prey species: the Pyrenean Chamois (*Rupicapra pyrenaica*), the Red Deer (*Cervus elaphus*), the Fallow Deer (*Dama dama*), the Roe Deer (*Capreolus capreolus*), sheep (*Ovis aries*), the cow (*Bos taurus*), the horse (*Equus caballus*), the goat (*Capra hircus*) and the wild boar (*Sus scrofa*). Prey species remains constitutes the basic feeding source for the vultures in the area under study.

The three vultures are cliff-nesting and long-lived species characterized by their low fecundity [3]. Concerning their diet, the Bearded Vulture is the only vertebrate that feeds almost exclusively on bone remains of medium size ungulates (see reviews in [6], [5]), whereas the Egyptian vulture feeds on dead small animals and the Eurasian Griffon vultures on medium and large sized animals [3].

The Bearded Vulture has a mind lifespan in wild birds of 21 years [2]. The mean age of first breeding is 8 years. The number of decedents for breeding is one chick. The female's annual fertility rate in Catalonia in the last years is estimated to be around 38%.

The Egyptian Vulture and the Griffon Vulture has a mind lifespan in wild birds of 25 years. The mean age of first breeding is 5 years. The number of descendants per breeding is one chick. The female's annual fertility rate in Catalonia in the last years is estimated to be around 59% for the Egyptian Vulture and 75% for the Griffon Vulture ([3],[5]).

It is accepted that there is the same proportion of females than males in the three types of vultures.

The ecosystem modelled in [1] is characterized by the presence of the domestic species, sheep. We considered that species stay all year in the mountain and thus, the bones they left when they die is available for the vultures. The real situation is that there are some domestic animals which live permanently in the mountain whereas others only spend the summer season there. This fact is assumed in the present paper.

Taking all this background information into consideration, the following data are required for each species:

- I_1: age at which adult size is reached. Age at which the animal eats like an adult animal does. Moreover, at this age it will have surpassed the critical early phase during which the mortality rate is high;
- I_2: age at which it starts to be fertile;
- I_3: age at which it stops being fertile;
- I_4: average life expectancy;
- I_5: fertility ratio (number of descendants by 100 fertile female);
- I_6: mortality ratio in the first years, $age < I_1$ (per cent);
- I_7: mortality ratio in adult animals, $age \geq I_1$ (per cent);
- I_8: ratio of females in the population (per cent).
- I_9: amount of bones from young animals when they die (kg)
- I_{10}: amount of bones from adult animals when they die (kg)
- I_{11}: amount of meet from young animals when they die (kg)
- I_{12}: amount of meet from adult animals when they die (kg)
- I_{13}: amount of bones necessary per year and animal (kg)
- I_{14}: amount of meet necessary per year and adult animal (kg)
- I_{15}: amount of grass necessary per year and adult animal (kg)

The required information about each species is shown in Table 4 (see Appendix).

3 A formal framework: A P System Based Model of the Ecosystem

In this section, we present a model of the ecosystem described in Section 2 by means of probabilistic P systems. We will study the behaviour of this ecosystem under different initial conditions.

First, we define the P systems based framework (probabilistic P systems), where additional features such as electrical charges which describe specific properties in a better way, are used.

Definition 1. *A probabilistic P system of degree q is a tuple*

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \dots, \mathcal{M}_q, R, \{c_r\}_{r \in R})$$

where:

- *Γ is the alphabet (finite and nonempty) of objects (the working alphabet);*
- *μ is a membrane structure (a rooted tree), consisting of q membranes, labelled $1, 2, \dots, q$. The skin membrane is labelled by 1. We also associate electrical charges with membranes from the set $\{0, +, -\}$, neutral and positive;*
- *$\mathcal{M}_1, \dots, \mathcal{M}_q$ are strings over Γ, describing the multisets of objects initially placed in the n regions of μ;*
- *R is a finite set of evolution rules. An evolution rule associated with the membrane labelled by i is of the form*

  $$r:\ u[\ v\ ]_i^{\alpha} \xrightarrow{c_r} u'[\ v'\ ]_i^{\alpha'}$$

  where u, v, u', v' are multisets over Γ, $\alpha, \alpha' \in \{0, +, -\}$, and c_r is a real number between 0 and 1. Besides, for each $u, v \in M(\Gamma)$, $i \in \{1, 2, \dots, q\}$ and $\alpha \in \{0, +, -\}$, it must verify $\sum_{j=1}^{t} c_{r_j} = 1$, being $r_1, \dots, r_t$ the rules whose left-hand side is $u[\ v\ ]_i^{\alpha}$.

We denote by $[\ v \to v'\ ]_i^{\alpha}$ the rule $u[\ v\ ]_i^{\alpha} \to u'[\ v'\ ]_i^{\alpha'}$ in the case $u = u' = \lambda$, and $\alpha = \alpha'$.

We assume that a global clock exists, marking the time for the whole system (for all compartments of the system); that is, all membranes and the application of all the rules are synchronized.

The multisets of objects present at any moment in the n *regions* of the system constitutes the *configuration* of the system at that moment. Particularly, tuple $(\mathcal{M}_1, \dots, \mathcal{M}_q)$ is the initial configuration of the system.

The P system can pass from one configuration to another one by using the rules from R as follows:

- A rule $u[\ v\ ]_i^{\alpha} \xrightarrow{c_r} u'[\ v'\ ]_i^{\alpha'}$ is applicable (with a probability c_r) to a membrane labelled by i, and with α as electrical charge, when the multiset u is contained in the father of membrane i, and the multiset v is contained in membrane i. When rule $u[\ v\ ]_i^{\alpha} \xrightarrow{c_r} u'[\ v'\ ]_i^{\alpha'}$ is applied, multiset u (resp. v) in the father of membrane i (resp. membrane i) is removed of that membrane and multiset u' (resp. v') is produced in that membrane.

- The rules are applied in a *maximal consistent parallelism*, that is, all those rules of type $u_1[\ v_1\ ]_i^{\alpha} \rightarrow u_1'[\ v_1'\ ]_i^{\alpha'}$ and $u_2[\ v_2\ ]_i^{\alpha} \rightarrow u_2'[\ v_2'\ ]_i^{\alpha'}$ might be applied simultaneously in a maximal way.
- The constants c_r associated with the rules indicate the affinity of the above mentioned rule for its application.

3.1 The model

The model proposed consists in the following probabilistic P system of degree 2 with two electrical charges (neutral and positive):

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \mathcal{M}_2, R, \{c_r\}_{r\in R})$$

where:

- $\Gamma = \{X_{ij},\ Y_{ij},\ V_{ij},\ Z_{ij}:\ 1 \leq i \leq n,\ 0 \leq j \leq g_{i,7}\} \cup \{b_{0i},\ b_i:\ 1 \leq i \leq n\} \cup \{B,\ G,\ M,\ B',\ G',\ M',\ C,\ C'\} \cup \{H_i,\ H_i',\ F_i,\ F_i',\ T_i,\ d_i,\ a_i:\ 1 \leq i \leq n\}$ is the working alphabet.

  In our model, $n = 17$ represents the different types of animals (according to the management) of the 12 species which compose the ecosystem under study. Symbols X, Y, V and Z represent the same animal but in different states. Index i is associated with the species and index j is associated with their age. It also contains the auxiliary symbols $B,\ B'$, which represent 0.5 kg of bones, $M,\ M'$, which represent 0.5 kg of meet and $G,\ G'$, which represent 0.5 kg of grass. Objects $H_i,\ H_i'$ represent 0.5 kg of biomass of bones, and objects $F_i,\ F_i'$ represent 0.5 kg of biomass meet left by specie i in different states. T_i is an object that is used for counting the existing animals of species i. If a species overcomes the maximum density, values will be regulated. At the moment when a regulation takes place, object a_i allows us to eliminate the number of animals of species i that exceed the maximum density. Object d_i is used to put under control domestic animals that are withdrawn from the ecosystem for their marketing.
- $\mu = [\ [\ ]_2\ ]_1$ is the membrane structure. We represent two regions, the skin and an inner membrane. The first is important to check the densities of every species do not overcome the threshold of the ecosystem. Animals reproduce, feed and die in the inner membrane. For the sake of simplicity, neutral polarization will be omitted.
- $\mathcal{M}_1$ and $\mathcal{M}_2$ are strings over Γ, describing the multisets of objects initially placed in the regions of μ (encoding the initial population and the initial food);

- $\mathcal{M}_1 = \{b_{0i},\ X_{ij}^{q_{ij}} : 1 \leq i \leq n,\ 0 \leq j \leq g_{i,7}\}$, where the multiplicity q_{ij} indicates the number of animals of species i whose age is j that are initially present in the ecosystem;
- $\mathcal{M}_2 = \{C\}$,

- The set R of evolution rules consists of:

  $r_0 \equiv [C \rightarrow G'^{\beta} C']_2^0$.

  $r_1 \equiv [b_{0,i} \rightarrow b_i]_1^0$.

  - Reproduction-rules.
    - Adult males:

      $r_2 \equiv [X_{ij} \xrightarrow{(1-k_{i,1})\cdot(1-k_{i,4})} Y_{ij}]_1^0,\ 1 \leq i \leq n,\ g_{i,5} \leq j \leq g_{i,7}$.
    - Adult females that reproduce:

      $r_3 \equiv [X_{ij} \xrightarrow{k_{i,2}\cdot k_{i,1}\cdot(1-k_{i,4})} Y_{ij} Y_{i0}^{k_{i,3}}]_1^0,\ 1 \leq i \leq n,\ i \neq 5,\ g_{i,5} \leq j < g_{i,6}$.

      $r_4 \equiv [X_{5j} \xrightarrow{0.5\cdot k_{5,1}} Y_{5j} Y_{50}^{k_{i,3}}]_1^0,\ g_{5,5} \leq j < g_{5,6}$.

      $r_5 \equiv [X_{5j} \xrightarrow{0.5\cdot k_{5,1}} Y_{5j} Y_{60}^{k_{i,3}}]_1^0,\ g_{5,5} \leq j < g_{5,6}$.
    - Adult females that do not reproduce:

      $r_6 \equiv [X_{ij} \xrightarrow{(1-k_{i,2})\cdot k_{i,1}\cdot(1-k_{i,4})} Y_{ij}]_1^0,\ 1 \leq i \leq n,\ g_{i,5} \leq j < g_{i,6}$.

      $r_7 \equiv [X_{ij} \xrightarrow{k_{i,1}\cdot(1-k_{i,4})} Y_{ij}]_1^0,\ 1 \leq i \leq n,\ g_{i,6} \leq j \leq g_{i,7}$.
    - Young animals that do not reproduce:

      $r_8 \equiv [X_{ij} \xrightarrow{1-k_{i,4}} Y_{ij}]_1^0,\ 1 \leq i \leq n,\ 1 \leq j < g_{i,5}$.
  - Growth rules.

    $r_9 \equiv [X_{ij} \xrightarrow{k_{i,5}\cdot k_{i,4}} Y_{i(g_{i,5}-1)} Y_{ij}]_1^0,\ 1 \leq i \leq n,\ g_{i,5} \leq j \leq g_{i,7}$.

    $r_{10} \equiv [X_{ij} \xrightarrow{(1-k_{i,5})\cdot k_{i,4}} Y_{ij}]_1^0,\ 1 \leq i \leq n,\ g_{i,5} \leq j \leq g_{i,7}$.
  - Mortality rules.

    $r_{11} \equiv b_i[\ ]_2^0 \rightarrow [b_i a_i^{g_{i,8}}]_2^+ :\ 1 \leq i \leq n$.
    - Young animals that survive:

      $r_{12} \equiv Y_{ij}[\ ]_2^0 \xrightarrow{1-m_{i,1}-m_{i,3}} [V_{ij} T_i]_2^+ :\ 1 \leq i \leq n,\ 0 \leq j < g_{i,4}$.
    - Young animals that die:

      $r_{13} \equiv Y_{ij}[\ ]_2^0 \xrightarrow{m_{i,1}} [H_i'^{f_{i,1}\cdot g_{i,3}} F_i'^{f_{i,2}\cdot g_{i,3}} B'^{f_{i,1}\cdot g_{i,3}} M'^{f_{i,2}\cdot g_{i,3}}]_2^+ :\ 1 \leq i \leq n,\ 0 \leq j < g_{i,4}$.

- Young animals that are retired from the ecosystem:

  $r_{14} \equiv [Y_{ij} \xrightarrow{m_{i,3}} \lambda]_1^0 :\ 1 \le i \le n,\ 0 \le j < g_{i,4}.$

- Adult animals that do not reach an average life expectancy and survive:

  $r_{15} \equiv Y_{ij}[\ ]_2^0 \xrightarrow{1-m_{i,2}} [V_{ij}T_i]_2^+ :\ 1 \le i \le n,\ g_{i,4} \le j < g_{i,7}.$

- Adult animals that do not reach an average life expectancy and die:

  $r_{16} \equiv Y_{ij}[\ ]_2^0 \xrightarrow{m_{i,2}} [H_i'^{f_{i,3}\cdot g_{i,3}} F_i'^{f_{i,4}\cdot g_{i,3}} B'^{f_{i,3}\cdot g_{i,3}} M'^{f_{i,4}\cdot g_{i,3}} V_{i,g_{i,5}-1}^{k_{i,4}} T_i^{k_{i,4}}]_2^+ :$ $1 \le i \le n,\ g_{i,4} \le j < g_{i,7}.$

- Animals that reach an average life expectancy and die in the ecosystem:

  $r_{17} \equiv Y_{ig_{i,7}}[\ ]_2^0 \xrightarrow{c_{17}} [H_i'^{f_{i,3}\cdot g_{i,3}} F_i'^{f_{i,4}\cdot g_{i,3}} B'^{f_{i,3}\cdot g_{i,3}} M'^{f_{i,4}\cdot g_{i,3}} V_{i,g_{i,5}-1}^{k_{i,4}} T_i^{k_{i,4}}]_2^+ :$ $1 \le i \le n$, being $c_{17} = k_{i,4} + (1-k_{i,4}) \cdot (m_{i,4} + (1-m_{i,4}) \cdot m_{i,2})$.

- Animals that reach an average life expectancy and retire from the ecosystem:

  $r_{18} \equiv [Y_{ig_{i,7}} \xrightarrow{(1-k_{i,4})\cdot(1-m_{i,4})\cdot(1-m_{i,2})} \lambda]_1 :\ 1 \le i \le n.$

– Regulation rules.

  $r_{19} \equiv [G' \to G]_2^+.$
  $r_{20} \equiv [B' \to B]_2^+.$
  $r_{21} \equiv [M' \to M]_2^+.$
  $r_{22} \equiv [C' \to C]_2^+.$
  $r_{23} \equiv [H_i' \to H_i]_2^+ :\ 1 \le i \le n.$
  $r_{24} \equiv [F_i' \to F_i]_2^+ :\ 1 \le i \le n.$

– Evaluation of the density of the different species in the ecosystem

  $r_{25} \equiv [T_i^{g_{i,8}} a_i^{g_{i,8}-g_{i,9}} \to \lambda]_2^+ :\ 1 \le i \le n.$

  $r_{26} \equiv [V_{ij} \to Z_{ij}]_2^+ :\ 1 \le i \le n,\ 0 \le j < g_{i,7}.$

– Feeding rules.

  $r_{27} \equiv [Z_{ij} a_i B^{f_{i,5}\cdot g_{i,3}} G^{f_{i,6}\cdot g_{i,3}} M^{f_{i,7}\cdot g_{i,3}}]_2^+ \to X_{i(j+1)}[\ ]_2^0 :\ 1 \le i \le n,\ 0 \le$ $j \le g_{i,7}.$

– Balance rules. The purpose of these rules is to make a balance at the end of the year. That is, the leftover food is not useful for the next year, so it is necessary to eliminate it. But if the amount of food not is enough, some animals die.

- Elimination of the remaining bones, meet and grass:

  $r_{28} \equiv [B \to \lambda]_2^0.$

$r_{29} \equiv [G \to \lambda]_2^0.$

$r_{30} \equiv [M \to \lambda]_2^0.$

$r_{31} \equiv [T_i \to \lambda]_2^0 : \ 1 \leq i \leq n.$

$r_{32} \equiv [a_i \to \lambda]_2^0 : \ 1 \leq i \leq n.$

$r_{33} \equiv [b_i]_2^0 \to b_i[\ ]_2^0 : \ 1 \leq i \leq n.$

$r_{34} \equiv [H_i]_2^0 \to H_i[\ ]_2^0 : \ 1 \leq i \leq n.$

$r_{35} \equiv [F_i]_2^0 \to F_i[\ ]_2^0 : \ 1 \leq i \leq n.$

· Young animals mortality:

$r_{36} \equiv [Z_{ij} \xrightarrow{g_{i,1}} H_i'^{f_{i,1}} F_i'^{f_{i,2}} B'^{f_{i,1}} M'^{f_{i,2}}]_2^0 : \ 1 \leq i \leq n, \ 0 \leq j < g_{i,4}.$

$r_{37} \equiv [Z_{ij}]_2^0 \xrightarrow{1-g_{i,1}} d_i[\ ]_2^0 : \ 1 \leq i \leq n, \ 0 \leq j < g_{i,4}.$

· Adult animals mortality:

$r_{38} \equiv [Z_{ij} \xrightarrow{g_{i,1}} H_i'^{f_{i,3}} F_i'^{f_{i,4}} B'^{f_{i,3}} M'^{f_{i,4}}]_2^0 : \ 1 \leq i \leq n, \ g_{i,4} \leq j \leq g_{i,7}.$

$r_{39} \equiv [Z_{ij} \xrightarrow{1-g_{i,1}} \lambda]_2^0 : \ 1 \leq i \leq n, \ g_{i,4} \leq j \leq g_{i,7}.$

$r_{40} \equiv [H_i \to \lambda]_1^0 : \ 1 \leq i \leq n.$

$r_{41} \equiv [F_i \to \lambda]_1^0 : \ 1 \leq i \leq n.$

The constants associated with the rules have the following meaning:

- $g_{i,1}$: 1 for wild animals and 0 for domestic animals.
- $g_{i,2}$: 1 for carnivorous animals and 0 otherwise.
- $g_{i,3}$: proportion of time that they remain in the mountain during the year.
- $g_{i,4}$: age at which adult size is reached. This is the age at which the animal eats like and adult does, and at which if the animal dies, the amount of biomass it leaves is similar to the total one left by an adult. Moreover, at this age it will have surpassed the critical early phase during which the mortality rate is high.
- $g_{i,5}$: age at which it starts to be fertile.
- $g_{i,6}$: age at which it stops being fertile.
- $g_{i,7}$: average life expectancy in the ecosystem.
- $g_{i,8}$: maximum density of the ecosystem.
- $g_{i,9}$: number of animals that survive after reaching maximum density of the ecosystem.
- $k_{i,1}$: proportion of females in the population (per one).
- $k_{i,2}$: fertility ratio (proportion of fertile female that reproduce).
- $k_{i,3}$: number of descendants per each fertile female that reproduce.

- $k_{i,4}$: it is equal to 0 when the species go through a natural growth (animals which remain in the same territory throughout their lives) and it is equal to 1 when animals are nomadic (the Bearded Vulture moves from one place to another until it is 6–7 years old, when it settles down).
- $k_{i,5}$: population growth (per one).
- $m_{i,1}$: natural mortality ratio in first years, $age < g_{i,4}$ (per one).
- $m_{i,2}$: mortality ratio in adult animals, $age \geq g_{i,4}$ (per one).
- $m_{i,3}$: percentage of domestic animals withdrawn in the first ages of the not stabilized populations.
- $m_{i,4}$: is equal to 1 if the animal dies at the age of $g_{i,7}$ and it is not retired, and it is equal to 0 if the animal not dies at the age of $g_{i,7}$ but it is retired from the ecosystem.
- $f_{i,1}$: amount of bones from young animals when they die, $age < g_{i,4}$.
- $f_{i,2}$: amount of meet from young animals when they die, $age < g_{i,4}$.
- $f_{i,3}$: amount of bones from adult animals when they die, $age \geq g_{i,4}$.
- $f_{i,4}$: amount of meet from adult animals when they die, $age \geq g_{i,4}$.
- $f_{i,5}$: amount of bones necessary per year and animal (1 unit is equal 0.5 kg of bones).
- $f_{i,6}$: amount of grass necessary per year and animal.
- $f_{i,7}$: amount of meet necessary per year and animal.

The constants used in this work are the same than those of [1], except for those referring to the maximum density of the population, the meet or the grass. However, they have been renamed in this work in order to be able to group them according to their characteristics. Thus, general characteristics are now named with g, those of reproduction with k, those corresponding to mortality with m and those of feeding with f. See Appendix in table 5.

3.2 Structure of the P system running

The model presented in [1], shows some restrictions which produce undesired effects for the study of the ecosystem evolution. Thus, in the first version of the model, it was not born in mind the resources for the feeding of herbivorous species, that is, it was assumed that there was enough grass at the ecosystem for all the population. Similarly, the maximum density of certain species in some areas of the ecosystem was not taken into consideration. It has been experimentally proved that, when the number of animals of a species exceeds a threshold, a phenomenon of autoregulation of the population takes place.

In this paper, we present a new model of the ecosystem that includes new ingredients with the aim to overcome the limitations previously described. More specifically, the modifications made are the following:

- It has been added new species which have active roles in the ecosystem under study, although their roles are perhaps less relevant that those of the first species studied. These species are the wild boar, the horse, the goat and the cow. Besides, it has been included greedy species such as the Egyptian Vulture and the Griffon Vulture which compete with the Bearded Vulture.
- A new module has been added in order to regulate the population density of the ecosystem.
- The mortality module has been modified in order to consider that after an animal dies, in addition to the bones it leaves at the ecosystem, its meet serves as food for other animals.
- The feeding module has also been modified because the feeding resources for the species at the ecosystem have been modelled in this new approach. For this reason, new objects have been introduced representing, apart from the bones, the amount of meet and grass available at the ecosystem.

In the model presented in Section 3.1, a new module devoted to control the density has been introduced. From the point of view of the execution of the system, the new module has been incorporated between the Mortality and the Feeding modules. These are depicted in Figure 1.

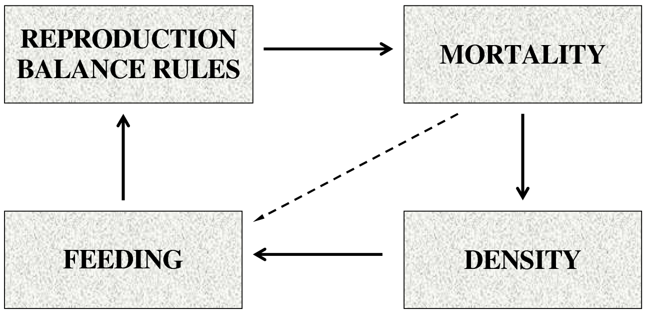


Fig. 1. Modules of the P systems

Let us recall that in the model presented in [1], objects X represent the different species along the execution of the reproduction module. Objects X evolve to objects Y when they pass to the mortality module, and these objects Y evolve to objects Z when they pass to the feeding module. Finally, the cycle is completed when objects Z evolve to objects X.

In order to keep the representation of the species at the different modules, we have used objects V to describe the species at the density module. For that purpose, objects Y (mortality module) evolve to objects V (density module), together with objects T which represent the number of individuals per each species. Then, objects V evolve to objects Z (feeding module). By the way, objects T will allow the activation of the process of auto–regulation of the ecosystem when the number of individuals of a species exceed the threshold of maximum density, which is codified by objects a.

When a cycle is produced, all objects which are not associated with species are eliminated, except the biomass generated by the animals that have died due to the process of regulation.

4 Results and discussions

The software tool used for the purposes of this paper is based on P-Lingua 2.0 [4], P-Lingua is a new programming language able to define P systems of different types (from now on, frameworks). For instance, P-Lingua can define any P system within the probabilistic framework mentioned in this paper.

Next, we describe how to implement in P–Lingua 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 said 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 give configuration, we proceed as follows:
   - It is computed the greatest number N so that u^N appears in the father 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.

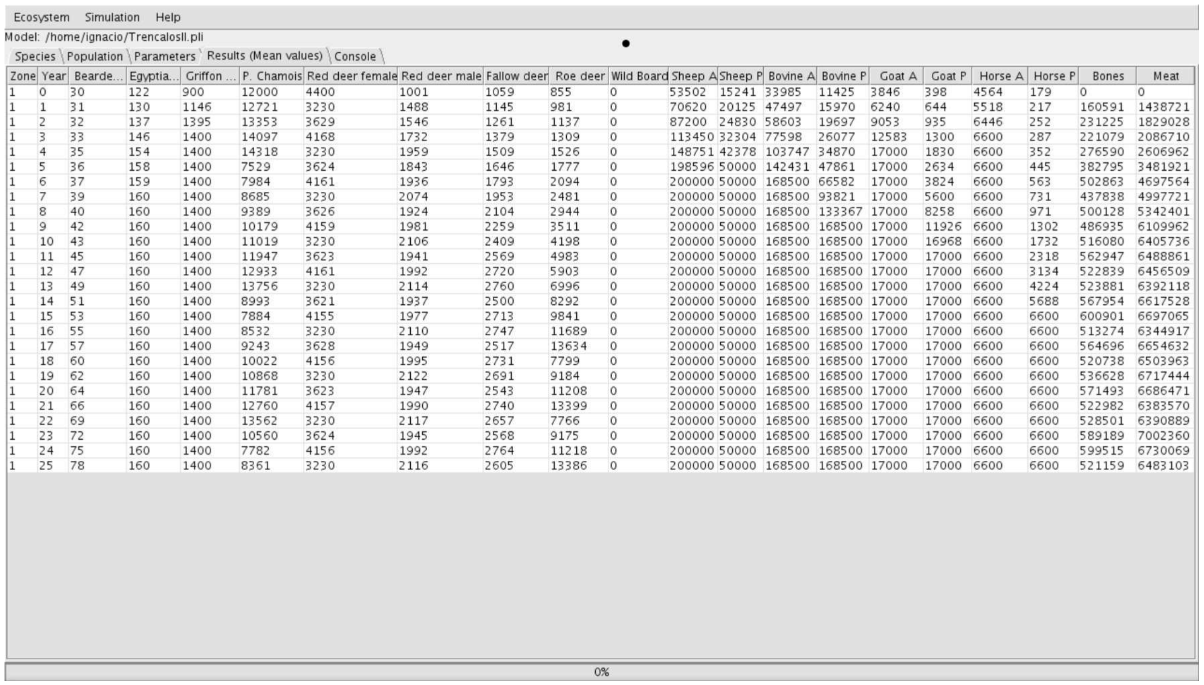
P-Lingua 2.0 provides a JAVA library that defines algorithms in order to simulate P system computations for each supported framework, so we are using a common algorithm for all P systems within the probabilistic framework.

By defining the ecosystem model by a P system written in P-Lingua, it is possible to check, validate and improve the model in a flexible way, instead of developing a new "ad hoc" simulator for each new model.

The application has a friendly user-interface, which sits on the P-Lingua JAVA library, allowing the user to change the initial parameters of the ecosystem in an easy way without special knowledge about the P system or the initial multisets. The main objective is to make virtual experiments over the ecosystem.

The current version of this software is a prototype GPL licensed [8].

In order to compare the model presented in this work to the one presented in [1], it has been used the P-lingua simulator [4]. We have simulated the evolution of the



Model: /home/ignacio/TrencalosII.pli

Zone	Year	Bearde...	Egyptia...	Griffon ...	P. Chamois	Red deer female	Red deer male	Fallow deer	Roe deer	Wild Board	Sheep A	Sheep P	Bovine A	Bovine P	Goat A	Goat P	Horse A	Horse P	Bones	Meat
1	0	30	122	900	12000	4400	1001	1059	855	0	53502	15241	33985	11425	3846	398	4564	179	0	0
1	1	31	130	1146	12721	3230	1488	1145	981	0	70620	20125	47497	15970	6240	644	5518	217	160591	1438721
1	2	32	137	1395	13353	3629	1546	1261	1137	0	87200	24830	58603	19697	9053	935	6446	252	231225	1829028
1	3	33	146	1400	14097	4168	1732	1379	1309	0	113450	32304	77598	26077	12583	1300	6600	287	221079	2086710
1	4	35	154	1400	14318	3230	1959	1509	1526	0	148751	42378	103747	34870	17000	1830	6600	352	276590	2606962
1	5	36	158	1400	7529	3624	1843	1646	1777	0	198596	50000	142431	47861	17000	2634	6600	445	382795	3481921
1	6	37	159	1400	7984	4161	1936	1793	2094	0	200000	50000	168500	66582	17000	3824	6600	563	502863	4697564
1	7	39	160	1400	8685	3230	2074	1953	2481	0	200000	50000	168500	93821	17000	5600	6600	731	437838	4997721
1	8	40	160	1400	9389	3626	1924	2104	2944	0	200000	50000	168500	133367	17000	8258	6600	971	500128	5342401
1	9	42	160	1400	10179	4159	1981	2259	3511	0	200000	50000	168500	168500	17000	11926	6600	1302	486935	6109962
1	10	43	160	1400	11019	3230	2106	2409	4198	0	200000	50000	168500	168500	17000	16968	6600	1732	516080	6405736
1	11	45	160	1400	11947	3623	1941	2569	4983	0	200000	50000	168500	168500	17000	17000	6600	2318	562947	6488861
1	12	47	160	1400	12933	4161	1992	2720	5903	0	200000	50000	168500	168500	17000	17000	6600	3134	522839	6456509
1	13	49	160	1400	13756	3230	2114	2760	6996	0	200000	50000	168500	168500	17000	17000	6600	4224	523881	6392118
1	14	51	160	1400	8993	3621	1937	2500	8292	0	200000	50000	168500	168500	17000	17000	6600	5688	567954	6617528
1	15	53	160	1400	7884	4155	1977	2713	9841	0	200000	50000	168500	168500	17000	17000	6600	6600	600901	6697065
1	16	55	160	1400	8532	3230	2110	2747	11689	0	200000	50000	168500	168500	17000	17000	6600	6600	513274	6344917
1	17	57	160	1400	9243	3628	1949	2517	13634	0	200000	50000	168500	168500	17000	17000	6600	6600	564696	6654632
1	18	60	160	1400	10022	4156	1995	2731	7799	0	200000	50000	168500	168500	17000	17000	6600	6600	520738	6503963
1	19	62	160	1400	10868	3230	2122	2691	9184	0	200000	50000	168500	168500	17000	17000	6600	6600	536628	6717444
1	20	64	160	1400	11781	3623	1947	2543	11208	0	200000	50000	168500	168500	17000	17000	6600	6600	571493	6686471
1	21	66	160	1400	12760	4157	1990	2740	13399	0	200000	50000	168500	168500	17000	17000	6600	6600	522982	6383570
1	22	69	160	1400	13562	3230	2117	2657	7766	0	200000	50000	168500	168500	17000	17000	6600	6600	528501	6390889
1	23	72	160	1400	10560	3624	1945	2568	9175	0	200000	50000	168500	168500	17000	17000	6600	6600	589189	7002360
1	24	75	160	1400	7782	4156	1992	2764	11218	0	200000	50000	168500	168500	17000	17000	6600	6600	599515	6730069
1	25	78	160	1400	8361	3230	2116	2605	13386	0	200000	50000	168500	168500	17000	17000	6600	6600	521159	6483103

Fig. 2. A tool for simulating ecosystems

Bearded Vulture, Red Deer, Fallow Deer, Roe Deer and Pyrenean Chamois species, in a period of 25 years. The starting information is the population registered in the year 2008. Some of the results are shown in Figure 3.

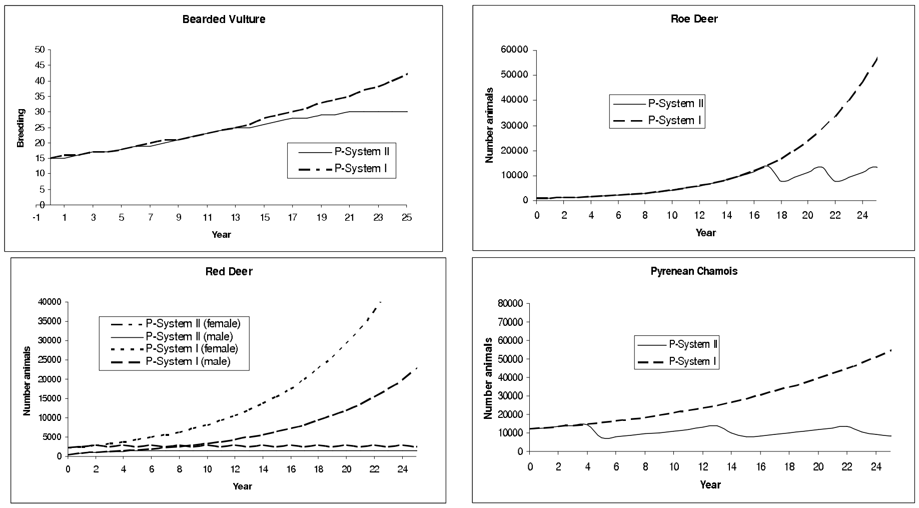


Fig. 3. Result of the simulation in 25 years time

It is observed that during the first years, the obtained values are almost equal. However, from a certain moment on, the population of the ecosystem experiments an exponential growth according to the P system [1] (P system I in the picture) whereas it maintains stable according to P systems II. The results of the new P system (P system II in the picture) are closer to the real situation.

The first step that will be taken is to validate the P system presented with real information, including all the proposed species with its specifications. When the model is validated, it will be able to apply it for the study of the evolution of an ecosystem under different situations. The model can be a useful tool for the management of the species.

In order to obtain a more realistic model, the following step should be to include possible movements of species among adjacent zones of the ecosystem.

5 Conclusions and Future works

A probabilistic P System which models an ecosystem related with Scavenger birds, and that is located in the Catalan Pyrenees, has been presented. By using this kind of P System, it has been possible to study the dynamics of the mentioned ecosystem adding new ingredients. That framework allows us to analyze how the ecosystem would evolve when different biological factors were modified either by nature or through human intervention, improving the results presented in [1].

A new JAVA software tool with a friendly user-interface sitting on the P-Lingua 2.0 JAVA library [4] has been developed in this paper. 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 hypotheses will be selected by the experts in order to be checked in real experiments.

In a future work, we will try to model neighbouring ecosystems with existing interactions among them, so it will be necessary to modify the scenario. Multi-environment P systems introduced in [7] could provides a friendly and flexible framework to get a new approach.

Acknowledgement

The authors acknowledge the support of the project TIN2006–13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. M. Cardona, M. A. Colomer, M.J. Pérez–Jiménez, D. Sanuy, A. Margalida. Modelling ecosystems using P Systems: The Bearded Vulture, a case of study. *Lecture Notes in Computer Science*, **5391** (2009), 137–156.
2. C.J.Brown. Population dynamics of the Bearded Vulture Gypaetus barbatus in southern Africa. *African Journal of Ecology*, **35** (1997), 53–63.
3. J.A. Donázar. *Los buitres ibéricos: biología y conservación*. J.M. Reyero Editor, Madrid, Spain, 1993.
4. M. García–Quismondo, R. Gutiérrez–Escudero, I. Pérez–Hurtado. P–Lingua 2.0: added features and first applications. In this volume.
5. A. Margalida, J.Bertran, R. Heredia: Diet and food preferences of the endangered Bearded Vulture Gypaetus barbatus: a basis for their conservation. *Ibis* **151** (2009), 235–243.
6. A. Margalida, D. García, A. Cortés-Avizanda. Factors influencing the breeding density of Bearded Vultures, Egyptian Vultures and Eurasian Griffon Vultures in Catalonia (NE Spain): management implications. *Animal Biodiversity and Conservation*, **30**, 2 (2007), 189–200.
7. F.J. Romero, M.J. Pérez–Jiménez. A model of the Quorum Sensing System in Vibrio Fischeri using P systems. *Artificial Life*, 14, 1 (2008), 95-109.
8. GPL license: `http://www.gnu.org/copyleft/gpl.html`

Appendix

Species	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}	I_{12}	I_{13}	I_{14}	I_{15}
Bearded Vulture	1	8	20	21	38	6	12	50	0	0	0	0	460	0	0
Egyptian Vulture	1	5	24	25	59.3	21	25	50	0	0	0	0	0	166	0
Griffon Vulture	1	5	24	25	75	14	1	50	0	0	0	0	0	400	0
Pyrenean Chamois	1	2	18	18	75	60	6	55	3	6	4	24	0	0	275
Red Deer	1	2	17	17-20	75	34	6	50	10	20	15	90	0	1270	0
Fallow Deer	1	2	12	12	55	50	6	75	1	2	14	37	0	550	0
Roe Deer	1	1	10	10	100	58	6	67	0.5	1	4	19	0	300	0
Wild Board	1	1	4	11	3.5	62-79	30-40	50	4	12	6	60	0	365	0
Sheep	1	2	8	8	75	15	3	96	3.5	7	4	28	0	660	0
Bovine	2	2	9	14	90	5.7	0.45	90	10.5	6	59.5	519	0	5500	0
Goat	1	2	8	6	90	12	2	97	3.5	9.5	4	37.5	0	700	0
Horse	3	3	9	20	90	3.4	1.42	97	10.5	9	59.5	891	0	6000	0

Fig. 4. Natural constants used in the model

Species	i	$g_{i,1}$	$g_{i,2}$	$g_{i,3}$	$g_{i,4}$	$g_{i,5}$	$g_{i,6}$	$g_{i,7}$	$g_{i,8}$	$g_{i,9}$	$k_{i,1}$	$k_{i,2}$	$k_{i,3}$	$k_{i,4}$	$k_{i,5}$	$m_{i,1}$	$m_{i,2}$	$m_{i,3}$	$m_{i,4}$	$f_{i,1}$	$f_{i,2}$	$f_{i,3}$	$f_{i,4}$	$f_{i,5}$	$f_{i,6}$	$f_{i,7}$
Bearded Vulture	1	1	0	1	1	8	20	21	120	120	0.5	0.08	1	1	0.04	0.06	0.12	0	1	0	0	0	0	460	0	0
Egyptian Vulture	2	1	0	0.5	1	5	24	25	160	160	0.5	0.593	1	0	0	0.17	0.07	0	1	0	0	0	0	0	0	332
Griffon Vulture	3	1	0	1	1	5	24	25	1400	1400	0.5	0.75	1	0	0	0.03	0.01	0	1	0	0	0	0	0	0	800
P. chamois	4	1	0	1	1	2	18	18	15000	7500	0.55	0.75	1	0	0	0.6	0.06	0	1	6	8	12	48	0	550	0
Red deer female	5	1	0	1	1	2	17	17	4615	3230	1	0.75	1	0	0	0.34	0.06	0	1	15	26	30	120	0	2540	0
Red deer male	6	1	0	1	1	2	20	20	2885	2020	0	0	0	0	0	0.34	0.36	0	1	24	30	48	192	0	2540	0
Fallow deer	7	1	0	1	1	2	12	12	3000	2400	0.75	0.55	1	0	0	0.5	0.06	0	1	2	28	4	74	0	1100	0
Roe deer	8	1	0	1	1	1	10	10	15000	7500	0.67	1	1	0	0	0.58	0.06	0	1	1	8	2	38	0	600	0
Wild Board	9	1	0	1	1	1	4	11	200000	200000	0.5	0.035	3	0	0	0.705	0.035	0	0	8	12	24	120	0	730	0
Sheep A	10	0	0	1	1	2	8	8	200000	200000	0.96	0.75	1	0	0	0.15	0.030	0.59	0	7	8	14	56	0	1320	0
Sheep P	11	0	0	0.5	1	2	8	8	50000	50000	0.96	0.75	1	0	0	0.15	0.030	0.59	0	7	8	14	56	0	1320	0
Bovine A	12	0	0	1	2	2	9	14	168500	168500	0.9	0.9	1	0	0	0.057	0.045	0	0	21	119	12	1038	0	11000	0
Bovine P	13	0	0	0.5	2	2	9	14	168500	168500	0.9	0.9	1	0	0	0.057	0.045	0	0	21	119	12	1038	0	11000	0
Goat A	14	0	0	1	1	2	8	6	17000	17000	0.97	0.9	1	0	0	0.12	0.015	0.59	0	7	8	19	75	0	1400	0
Goat P	15	0	0	0.5	1	2	8	6	17000	17000	0.97	0.9	1	0	0	0.12	0.015	0.59	0	7	8	19	75	0	1400	0
Horse A	16	0	0	1	3	3	9	20	6600	6600	0.97	0.9	1	0	0	0.034	0.014	0	0	21	119	18	1782	0	12000	0
Horses P	17	0	0	0.5	3	3	9	20	6600	6600	0.97	0.9	1	0	0	0.034	0.014	0	0	21	119	18	1782	0	12000	0

Fig. 5. Constants used in the P system based model. Acronyms (A) and (P) mean *annual* and *periodical* respectively

Characterizing the Aperiodicity of Irreducible Markov Chains by Using P Systems

Mónica Cardona[1], M. Angels Colomer[1],
Mario J. Pérez-Jiménez[2]

[1] Dpt. of Mathematics, University of Lleida
Av. Alcalde Rovira Roure, 191. 25198 LLeida, Spain
{mcardona,colomer}@matematica.udl.es

[2] Research Group on Natural Computing
Dpt. of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
marper@us.es

Summary. 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 formal verification of this solution is presented and a comparative analysis with respect to another known solution is described.

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 the 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 designed in order to study the periodicity of that class. Section 4 shows a formal verification of the designed P system. In Section 5, the solution presented is compared with another solution obtained from [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:

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

Without loss of generality, we can suppose that the state space is the set of non-negative 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 $S = \{s_1, \dots, s_k\}$.

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 arrow 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 one class of equivalence by $\leftrightarrow$; 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 1. *Let $A = \{s_1, \ldots, s_r\}$ be a recurrent class. The period of A is*

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

That is, the period of A is the greatest common divisor of all times of recurrences of the states of that class, smaller than or equal to r.

Proof. By definition, given a state s_i $(1 \leq i \leq r)$ its period is

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

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 $d' = g.c.d.\{n \mid p_{ii}^{(n)} > 0;\ 1 \leq i, n \leq r\}$. Let us see that $d = d'$. For that, we will check that any trajectory from a state $s_i \in A$ to itself, with the length bigger than r, is the composition of trajectories with length smaller than or equal to r between the same states.

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 $p_{ii}^{(n)} \geq p_{ii_0}^{(n')} \cdot p_{i_0i_0}^{(n_0)} \cdot p_{i_0i}^{(n'')} > 0$, being $n = n' + n_0 + n''$. 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

$$p_{ii}^{(n)} \geq p_{ii_0}^{(n')} \cdot p_{i_0i_0}^{(n_0)} \cdot p_{i_1i_1}^{(n_1)} \cdots p_{i_ri_r}^{(n_r)} \cdot p_{i_ri}^{(n'')} > 0$$

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

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

Lemma 1. *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 2. *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 3. *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 2 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 2.

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 $\mathbf{\Pi} = \{\Pi(P_k) \mid k \in \mathbf{N}\}$, associating with P_k a P system with external output, such that:

- There exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(P_k)$ from P_k.
- 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:

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

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

  In the working alphabet the objects:
  - s_{ii} represents (at the initial configuration) the state s_i of the chain.
  - t_{ij} and τ_{ij} represent the elements p_{ij} of the Boolean matrix associated with the transition matrix of the Markov chain.
  - s_{ijr} represents the existence of a path of length r from the state s_i to state s_j.
  - T_r and p_r represent the existence of a recurrence time equal to r in different configurations.
  - τ_{ij} represents that the state s_j is reachable from state s_i.
- Membrane structure: $\mu(P_k) = [\ [\ [\ [\ ]_1\ ]_2\ ]_3]_4$.
- Initial multisets:
  $\mathcal{M}_1 = \{t_{ij}^{p_{ij}} \mid 1 \leq i,j \leq k\} \cup \{\beta_0\}$
  $\mathcal{M}_2 = \{s_{ii} \mid 1 \leq i \leq k\}$
  $\mathcal{M}_3 = \{b_i \mid 1 \leq i \leq k\} \cup \{d_0\}$
  $\mathcal{M}_4 = \emptyset$
- The set R of evolution rules consists of the following rules:

$$\begin{aligned} r_1(ij) &\equiv [t_{ij} \rightarrow \tau_{ij} t_{ij}^k]_1, \quad 1 \leq i,j \leq k \\ r_2(i) &\equiv [\beta_i \rightarrow \beta_{i+1}]_1, \quad 0 \leq i \leq k-2 \\ r_3 &\equiv [\beta_{k-1}]_1 \rightarrow c_0^k \\ r_4(rij) &\equiv [c_r s_{ij} \tau_{j1}^{p_{j1}} \ldots \tau_{jk}^{p_{jk}}]_2 \rightarrow [s_{i1}^{p_{j1}} \ldots s_{ik}^{p_{jk}} c_{r+1}^{\gamma_j}]_2 s_{i1r+1}^{p_{j1}} \ldots s_{ikr+1}^{p_{jk}} T_{r+1}^{p_{ji}}, \\ &\quad 1 \leq i,j \leq k, \quad 0 \leq r \leq \alpha - 1, \gamma_j = \textstyle\sum_{l=1}^{k} p_{jl} \end{aligned}$$

$$
\begin{aligned}
r_5 &\equiv [\sigma]_2 \rightarrow \sigma \\
r_6(jr) &\equiv [s_{1jr} \ldots s_{kjr}]_3 \rightarrow [\sigma]_2\ yes, \quad 1 \leq j \leq k,\ 1 \leq r \leq \alpha \\
r_7(r) &\equiv [T_r b_r \rightarrow p_r]_3, \quad 1 \leq r \leq k \\
r_8(il) &\equiv [p_i p_{i+l} \rightarrow p_i p_l]_3, \quad 1 \leq i \leq k,\ 1 \leq l \leq k-i \\
r_9(i) &\equiv [p_i^2 \rightarrow p_i]_3, \quad 1 \leq i \leq k \\
r_{10}(i) &\equiv [d_i \rightarrow d_{i+1}]_3, \quad 0 \leq i \leq \alpha - 1 \\
r_{11}(r) &\equiv [d_\alpha p_r]_3 \rightarrow p_r[\ ]_3, \quad 2 \leq r \leq k \\
r_{12} &\equiv [d_\alpha p_1]_3 \rightarrow yes[\ ]_3 \\
r_{13} &\equiv [yes]_4 \rightarrow YES[\ ]_4 \\
r_{14}(r) &\equiv [p_r]_4 \rightarrow p_r[\ ]_4, \quad 1 \leq r \leq k
\end{aligned}
$$

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 Formal Verification

Given a computation $\mathcal{C}$ of the P system $\Pi(P_k)$, for each $m \in \mathbf{N}$ we denote by $\mathcal{C}_m$ the configuration of the system obtained after the execution of m steps. For each label $l \in \{1,2,3\}$, we denote by $\mathcal{C}_m(l)$ the multiset of objects contained in membrane l in the configuration $\mathcal{C}_m$. Besides, we denote by $\mathcal{C}_m(env)$ the content of the environment of the system in the configuration $\mathcal{C}_m$.

Proposition 1. *(First stage) We have the following:*

(1) For each m, $1 \leq m \leq k-1$, we denote $\psi_m = 1 + k + k^2 + \ldots + k^m = (k^{m+1}-1)/(k-1)$. Then

$$\mathcal{C}_m(1) = \{\beta_m \; t_{ij}^{k^m \cdot p_{ij}} \; \tau_{ij}^{\psi_{m-1} \cdot p_{ij}}\}.$$

(2) $\mathcal{C}_k(2) = \{c_0^k, \; s_{ii}, \; \tau_{ij}^{\psi_{k-1} \cdot p_{ij}}, \; t_{ij}^{(k^k) \cdot p_{ij}} \mid 1 \leq i,j \leq k\}$

Proof. (1) By induction on m.

Let us see the result for $m = 1$. First, we notice that rule $r_2(0)$ is applicable to configuration $\mathcal{C}_0$, so $\beta_1 \in \mathcal{C}_1(1)$. Rule $r_1(ij)$ is applicable to configuration $\mathcal{C}_0$ if and only if $p_{ij} = 1$. Hence, $\mathcal{C}_1(1) = \{\beta_1 \; t_{ij}^{k \cdot p_{ij}} \; \tau_{ij}^{p_{ij}}\}$.

Let m be such that $1 \leq m < k-1$ and $\mathcal{C}_m(1) = \{\beta_m \; t_{ij}^{k^m \cdot p_{ij}} \; \tau_{ij}^{\psi_{m-1} \cdot p_{ij}}\}$. Then, rule $r_2(m)$ is applicable to configuration $\mathcal{C}_m$, so $\beta_{m+1} \in \mathcal{C}_{m+1}(1)$. Rule $r_1(ij)$ is applicable to configuration $\mathcal{C}_m$ if and only if $p_{ij} = 1$. Hence, $\mathcal{C}_{m+1}(1) = \{\beta_{m+1} \; t_{ij}^{k^m \cdot k \cdot p_{ij}} \; \tau_{ij}^{(\psi_{m-1}+k^m) \cdot p_{ij}}\}$.

(2) From (1), we have $\mathcal{C}_{k-1}(1) = \{\beta_{k-1} \; t_{ij}^{k^{k-1} \cdot p_{ij}} \; \tau_{ij}^{\psi_{k-2} \cdot p_{ij}}\}$. Next, rule $r_1(ij)$ produces k objects t_{ij} and an object τ_{ij} for each object $t_{ij} \in \mathcal{C}_{k-1}(1)$. Moreover, rule r_3 produces k copies of c_0 dissolving membrane 1. □

Remark: Let us notice that condition $\tau_{ij} \in \mathcal{C}_r(1)$, $1 \leq r \leq k-1$, means that state s_j is reachable from state s_i.

Lemma 2. *For each i, j, r $(1 \leq i,j,r \leq k)$ we have the following:*

- *The sum of the multiplicities of objects $s_{1j} \ldots s_{kj}$ in $\mathcal{C}_{k+r}(2)$ is, at most, k^r.*
- *There exists, at most, k^{r+1} objects c_r in $\mathcal{C}_{k+r}(2)$.*
- $\tau_{ij}^{(\psi_{k-1}-\psi_{r-1}) \cdot p_{ij}} \in \mathcal{C}_{k+r}(2)$.

Proof. By induction on r.

Let us suppose that $r = 1$, Let i, j be such that $1 \leq i, j \leq k$. From (2) in Proposition 1 we have $\mathcal{C}_k(2) = \{c_0^k,\ s_{ii},\ \tau_{ij}^{\psi_{k-1} \cdot p_{ij}}\} \subseteq \mathcal{C}_k(2)$. Then, rules $r_4(0,1,1), \ldots, r_4(0,k,k)$ must be applied.

If $p_{ij} = 1$, then an object s_{ij} (resp. an object τ_{ij}) is produced (resp. is consumed) by the application of rule $r_4(0,i,i)$ (besides, these objects can only be spent/produced by the application of that rules). Hence, the sum of multiplicities of objects $s_{1j}, \ldots, s_{kj}$ will be $p_{1j} + \ldots + p_{kj} \leq k$, there exists at most k^2 objects c_1 in $\mathcal{C}_{k+1}(2)$, and $\tau_{ij}^{(\psi_{k-1}-1)} \in \mathcal{C}_{k+1}(2)$.

Let $r \geq 1, r < k$, and let us suppose that the result holds for r. Let i, j be such that $1 \leq i, j \leq k$. By the induction hypothesis the sum of the multiplicities of objects $s_{1i}, \ldots, s_{ki}$ in $\mathcal{C}_{k+r}(2)$ is, at most, k^r, there exists, at most, k^{r+1} objects c_r in $\mathcal{C}_{k+r}(2)$, and $\tau_{ij}^{(\psi_{k-1}-\psi_{r-1}) \cdot p_{ij}} \in \mathcal{C}_{k+r}(2)$. For each i $(1 \leq i \leq k)$ the rules $r_4(r1i), \ldots, r_4(rki)$ will be applied to configuration $\mathcal{C}_{k+r}(2)$ at most k^r times, so at most k^r objects τ_{ij} will be spent and $k^r \cdot k$ objects c_{r+1} will be produced. Then, there exists at most k^{r+2} objects c_{r+1} in $\mathcal{C}_{k+r+1}(2)$, and $\tau_{ij}^{(\psi_{k-1}-\psi_{r-1}-k^r) \cdot p_{ij}} \in \mathcal{C}_{k+r+1}(2)$.

Moreover, each object s_{iq} $(1 \leq q \leq k)$ produces, at most, an object s_{ij} in $\mathcal{C}_{k+r+1}(2)$. Hence, the sum of multiplicities of $s_{1j}, \ldots, s_{kj}$ in $\mathcal{C}_{k+r+1}(2)$ will be, at most, $k^r + \ldots + k^r = k \cdot k^r = k^{r+1}$. □

Proposition 2. *(Second stage) For each i, j, r $(1 \leq i, j, r \leq k)$ we have:*

(1) Objects s_{ij} and c_r belong to $\mathcal{C}_{k+r}(2)$ if and only if there exists a trajectory from state s_i to state s_j with a length r.

(2) A state s of the Markov chain has a recurrence time r if and only if $T_r \in \mathcal{C}_{k+r}(3)$.

Proof. (1) By induction on r.

Let us suppose that $r = 1$. If $s_{ij}, c_1 \in \mathcal{C}_{k+1}(2)$, then rule $r_4(0,i,i)$ must be applied by using objects $c_0, s_{ii}, \tau_{ij} \in \mathcal{C}_k(2)$. Then, $p_{ij} = 1$, otherwise $p_{ij} = 0 \Rightarrow \tau_{ij} \notin \mathcal{C}_k(2)$ (from Proposition 1).

Let i_0, j_0 $(1 \leq i_0, j_0 \leq k)$ and let us suppose that there exists a trajectory from state s_{i_0} to state s_{j_0} with a length 1. Then, $p_{i_0 j_0} = 1$. From Proposition 1 we deduce that $\mathcal{C}_k(2) = \{c_0^k,\ s_{ii},\ \tau_{ij}^{\psi_{k-1} \cdot p_{ij}},\ t_{ij}^{(k^k) \cdot p_{ij}} \mid 1 \leq i, j \leq k\}$, so for each i $(1 \leq i \leq k)$ rule $r_4(0ii)$ is applied once to configuration $\mathcal{C}_k(2)$. Then, $\{c_1, s_{i_0 j_0}\} \subseteq \mathcal{C}_{k+1}(2)$.

Let $r \geq 1$, $r < k$, and let us suppose that the result holds for r. Let i, j $(1 \leq i, j \leq k)$ be such that $s_{ij}, c_{r+1} \in \mathcal{C}_{k+r+1}(2)$. On the one hand, rule $r_4(ril)$ has been applied, at least once, to configuration $\mathcal{C}_{k+r}$ by using objects $c_r,\ s_{il},\ \tau_{lj}$ (for some l, $1 \leq l \leq k$). So, $p_{lj} = 1$. On the other hand, $c_r,\ s_{il} \in \mathcal{C}_{k+r}(2)$. Then, by induction hypothesis we deduce that there exists a trajectory with a length r from state s_i to state s_l. Hence, there exists a trajectory with the length $r+1$ from state s_i to state s_j.

Let i_0, j_0 $(1 \leq i_0, j_0 \leq k)$ and let us suppose that there exists a trajectory from state s_{i_0} to state s_{j_0} with a length $r+1$. Then, there exists a trajectory from state s_{i_0} to state s_{n_0} with a length r (for some n_0, $1 \leq n_0 \leq k$) such that $p_{n_0 j_0} = 1$. From the induction hypothesis we have $s_{i_0 n_0},\ c_r \in \mathcal{C}_{k+r}(2)$, and from Lemma 2 we deduce that

$$\{\tau_{n_0 j}^{(\psi_{k-1} - \psi_{r-1}) \cdot p_{n_0 j}} \mid 1 \leq j \leq k\} \subseteq \mathcal{C}_{k+r}(2)$$

Then, by applying rule $r_4(r, i_0, n_0)$ once, we obtain $\{c_{r+1},\ s_{i_0 j}^{p_{n_0 j}} : 1 \leq j \leq k\} \subseteq \mathcal{C}_{k+r+1}(2)$. Hence, $s_{i_0 j_0} \in \mathcal{C}_{k+r+1}(2)$ because of $p_{n_0 j_0} = 1$.

(2) Let r $(1 \leq r \leq k)$ be the recurrence time of a state s_i. From (1), we deduce that $s_{ii}, c_r \in \mathcal{C}_{k+r}(2)$. Therefore, rule $r_4(rij)$ has been applied to configuration $\mathcal{C}_{k+r-1}$, for some j, $1 \leq j \leq k$, such that $p_{ji} = 1$, and some object $c_{r-1} \in \mathcal{C}_{k+r-1}(2)$. Then, $T_r^{p_{ji}} = T_r \in \mathcal{C}_{k+r}(3)$.
Let r $(1 \leq r \leq k)$ such that $T_r \in \mathcal{C}_{k+r}(3)$. Then rule $r_4(r-1, i, j)$ has been applied to configuration $\mathcal{C}_{k+r-1}$, for some objects $s_{ij},\ c_{r-1}$ such that $p_{ji} = 1$. From (1) there exists a trajectory with a length $r-1$ from state s_i to state s_j. Hence, there exists a trajectory with a length r from state s_i to state s_i. □

Theorem 4. *(Output of the system)*
Let S be an irreducible homogeneous Markov chain of order k. Let $\alpha = 3k + \lceil \frac{k}{2} \rceil$. We have the following:

(1) The class S is aperiodic if and only if there exists r $(1 \leq r \leq \alpha - k)$ such that configuration $\mathcal{C}_{k+r+2}$ of $\Pi(P_k)$ is a halting configuration and $\mathcal{C}_{k+r+2}(env) = \{YES\}$.
(2) The class S is periodic with period equal to $n > 1$ if and only if configuration $\mathcal{C}_{\alpha+2}$ of $\Pi(P_k)$ is a halting configuration and $\mathcal{C}_{\alpha+2}(env) = \{p_n\}$.

Proof. Let S be an irreducible homogeneous Markov chain.

(1) Let us suppose that S is aperiodic. From Theorem 3, there exists a state s_{j_0} and a natural number $q > 0$ such that $\forall i$ $(1 \leq i \leq k \Rightarrow p_{i j_0}^{(q)} > 0)$. Then, for each i, $1 \leq i \leq k$, there exists a trajectory with a length q from state s_i to state s_{j_0}.
- If $q \leq k$, from (1) Proposition 2 we deduce that $s_{1 j_0}, \ldots, s_{k j_0}, c_q \in \mathcal{C}_{k+q}(2)$. These objects have been produced by the application of rules $r_4(q-1, 1, j_1), \ldots, r_4(q-1, k, j_k)$ to configuration $\mathcal{C}_{k+q-1}$, for some $j_1, \ldots, j_k$ such that $p_{j_1, j_0} = \cdots = p_{j_k, j_0} = 1$. So, $s_{1 j_0 q}, \ldots, s_{k j_0 q} \in \mathcal{C}_{k+q}(3)$. So, by applying the rule $r_6(j_0, q)$ to configuration $\mathcal{C}_{k+q}$, we have $\{yes\} \in \mathcal{C}_{k+q+1}(4)$ and $\sigma \in \mathcal{C}_{k+q+1}(2)$. At the next step, rules r_5 and r_{13} are applied. Then, $\mathcal{C}_{k+q+2}(env) = \{YES\}$, membrane 2 is dissolved and the system halts.
- If $q > k$, then any rule of the type r_6 is not applicable. From Proposition 2 we have encoded the recurrence times (smaller than or equal to k) in membrane 3 by objects T. Then, some rule of the type r_7 produces objects p corresponding to objects T. Next, by applying suitable rules r_8 and r_9

we compute the greatest common divisor of these recurrence times (from Theorem 1 we know that g.c.d. is equal to the period of the class). From the aperiodicity of the class S, we deduce that object p_1 belongs to $\mathcal{C}_\alpha(3)$. Then, the rule r_{12} produces an object *yes* in $\mathcal{C}_{\alpha+1}(4)$, and we obtain that $\mathcal{C}_{\alpha+2}(env) = \{YES\}$ by applying the rule r_{13}. Then, the system halts.

Let us suppose that there exists r $(1 \leq r \leq \alpha - k)$ such that configuration $\mathcal{C}_{k+r+2}$ is a halting configuration and $\mathcal{C}_{k+r+2}(env) = \{YES\}$. Then, rule r_{13} has been applied to configuration $\mathcal{C}_{k+r+1}(4)$, and $yes \in \mathcal{C}_{k+r+1}(4)$.

- If $r \leq k$, then the rule $r_6(jr)$ (for some j, $1 \leq j \leq k$) has been applied to configuration $\mathcal{C}_{k+r}$, with $s_{1jr}, \ldots, s_{kjr} \in \mathcal{C}_{k+r}(3)$, and $s_{1j}, \ldots, s_{kj}, c_r \in \mathcal{C}_{k+r}(2)$. From (1) in Proposition 2, there exists a trajectory with a length r from state s_i to state s_j, for each i $(1 \leq i \leq k)$. From Theorem 3 we conclude that class S is aperiodic.
- If $r > k$, object *yes* has been sent to membrane 4 by applying the rule r_{12} using objects d_α and p_1. But object p_1 has been produced by the iterated application of rules r_8 and r_9. As these rules compute the greatest common divisor of recurrence times, we deduce that the period of S is equal to 1.

(2) Now, let us suppose that the period of S is $n > 1$. From Theorem 3 we deduce that for each j, $1 \leq j \leq k$, and for each $n' > 0$ there exists i, $1 \leq i \leq k$, such that $p_{ij}^{(n')} = 0$. From (1) in Proposition 2 we have $\{s_{ij}, c_{n'}\} \nsubseteq \mathcal{C}_{k+n'}(2)$. So, $s_{ijn'} \notin \mathcal{C}_{k+n'}(3)$ and any rule of the type r_6 is applicable to configuration $\mathcal{C}_{k+n'+1}$. Next, rules of type r_7, r_8, r_9 compute the g.c.d. of the recurrence times. Finally, object p_n is sent to the environment after $\alpha+2$ steps by applying rules r_{11} and r_{14}. Then, the system halts.

Let us suppose that configuration $\mathcal{C}_{\alpha+2}$ is a halting configuration and $\mathcal{C}_{\alpha+2}(env) = \{p_n\}$. Then, any rule of the type r_6 will not be applied, so rules r_7, r_8, and r_9 will be applied computing the g.c.d of the recurrence times (smaller than or equal to k) of states s_i. Hence, class S is periodic and its period is equal to n. □

5 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:

$$\begin{aligned}\Gamma'(P_k) = \{d_{ij},\ t_{ij} \mid 1 \le i,j \le k,\} \ \cup \{c_r \mid 0 \le r \le 2k+2\} \cup \\ \{t_{ijur} \mid 1 \le i,j,u \le k,\ 0 \le r \le k\} \ \cup \{\beta_i \mid 0 \le i \le \alpha+1\} \cup \\ \{s_{ijr} \mid 1 \le i,j \le k,\ 0 \le r \le k\} \ \cup \{A_{i1},\ R_{ij} \mid 1 \le i,j \le k\}\end{aligned}$$

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:
  $\mathcal{M}'_1 = \emptyset$; $\mathcal{M}'_2 = \{\beta_0\}$; $\mathcal{M}'_3 = \{c_0\}$;
  $\mathcal{M}'_4 = \{s_{ii0} \quad t_{ij}^{p_{ij}(k-1)} \mid 1 \le i,j \le k\}$.
- The set R of evolution rules consists of the following rules:
  – Rules in the skin membrane labeled by 1:
  $r_1 = \{d_{ip} \rightarrow (R_{ip}, out) \mid 1 \le i \le k,\ 1 < p \le k\}$
  $r_2 = \{d_{i1} \rightarrow (A_{i1}, out) \mid 1 \le i \le k\}$
  – Rules in the membrane labeled by 2:
  $r_3 = \{\beta_i \rightarrow \beta_{i+1} \mid 0 \le i \le \gamma\} \ \cup \{\beta_{\gamma+1} \rightarrow \lambda\}$.
  $r_4 = \{d_j^2 \rightarrow d_j \mid 1 \le j \le k\}$
  $r_5 = \{d_j d_{j+l} \rightarrow d_j d_l \mid 1 \le j \le k,\ 2 \le j+l \le k\}$
  – Rules in the membrane labeled by 3:
  $r_6 = \{t_{ijur} \rightarrow (t_{ij} s_{uj(r+1)}, in_4) \mid p_{ij} = 1, u \ne j, 1 \le i,j,u \le k, 0 \le r < 3(k-1)\}$
  $r_7 = \{t_{iju(3k-3)} \rightarrow (t_{ij}, in_4) \mid p_{ij} = 1, u \ne j, 1 \le i,j,u \le k\}$
  $r_8 = \{t_{ijjr} \rightarrow (t_{ij}, in_4)\ d_{r+1} \mid p_{ij} = 1, 1 \le i,j \le k, 0 \le r < 3(k-1)\}$
  $r_9 = \{t_{ijj(3k-3)} \rightarrow (t_{ij}, in_4) \mid p_{ij} = 1, 1 \le i,j \le k\}$
  $r_{10} = \{c_r \rightarrow c_{r+1} \mid 0 \le r \le 6(k-1)+1\} \ \cup \{c_{6(k-1)+2} \rightarrow \lambda\}$
  – Rules in the membrane labeled by 4:
  $r_{11} = \{s_{uir} t_{i1}^{p_{i1}} \ldots t_{ik}^{p_{ik}} \rightarrow (t_{i1ur}^{p_{i1}} \ldots t_{ikur}^{p_{ik}}, out) \mid 1 \le u,i \le k,\ 0 \le r \le 3(k-1)\}$.
- The partial order relation ρ' over R' consists of the following relations on the rules of R':
  – Priority relation in the skin membrane: $\emptyset$.
  – Priority relation in the membrane labeled by 2: $\{r_4 > r_5\}$
  – Priority relation in the membranes labeled by 3: $\emptyset$.
  – Priority relation in membrane 4: $\emptyset$.

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 firstly. Secondly, an analysis of the times of execution obtained on designed simulators for both P systems with some case studies is presented.

5.1 Computational Resources Required

The resources required initially to construct the systems $\Pi(P_k)$ and $\Pi'(P_k)$, and the number of steps taken for 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.

5.2 Case Studies

We have designed 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 3 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	$p_{i,i+1} = 1$	$1 \leq i \leq 99$	
	$p_{100,1} = 1$		
2	$p_{i,i+1} = 1$	$1 \leq i \leq 99$	
	$p_{i,1} = 1$	$1 \leq i \leq 100$	
3	$p_{10j+i,10j+i+1} = 1$	$1 \leq i \leq 9$	$0 \leq j \leq 9$
	$p_{10j,10j-9} = 1$	$1 \leq j \leq 10$	
	$p_{10j+1,10j+11} = 1$	$0 \leq j \leq 8$	
	$p_{91,1} = 1$		
4	$p_{10j+i,10j+i+1} = 1$	$1 \leq i \leq 9$	$0 \leq j \leq 9$
	$p_{10j,10j-9} = 1$	$1 \leq j \leq 10$	
	$p_{10j+1,10j+11} = 1$	$0 \leq j \leq 8$	
	$p_{91,1} = 1$		
	$p_{1,1} = 1$		
5	$p_{10j+i,10j+i+1} = 1$	$1 \leq i \leq 9$	$0 \leq j \leq 9$
	$p_{10j,10j-9} = 1$	$1 \leq j \leq 10$	
	$p_{10j+1,10j+11} = 1$	$0 \leq j \leq 8$	
	$p_{91,1} = 1$		
	$p_{2,2} = 1$		
6	$p_{5j+i,5j+i+1} = 1$	$1 \leq i \leq 4$	$0 \leq j \leq 19$
	$p_{5j,5j-4} = 1$	$1 \leq j \leq 20$	
	$p_{5j+1,5j+6} = 1$	$0 \leq j \leq 18$	
	$p_{96,1} = 1$		
7	$p_{i,i+1} = 1$	$1 \leq i \leq 100$	
	$p_{i+1,i} = 1$	$1 \leq i \leq 100$	
	$p_{1+3i,4+3i} = 1$	$0 \leq i \leq 32$	
8	$p_{i,i+1} = 1$	$1 \leq i \leq 100$	
	$p_{i+1,i} = 1$	$1 \leq i \leq 100$	
	$p_{1+3i,4+3i} = 1$	$0 \leq i \leq 32$	
	$p_{1,1} = 1$		

Table 1. Adjacency values of the examples

6 Conclusions

Markov chains have applications in different fields such as physics, economics, biology, statistics, social sciences... 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 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. A formal verification of P system using the methodology based on the search of invariant formulae has been presented.

Example	period	Previous	New
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

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 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.

References

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*, 4361, Springer, 2006, 266–27.
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.

On Very Simple P Colonies

Lucie Ciencialová[1], Erzsébet Csuhaj-Varjú[2,3],
Alica Kelemenová[1,4], György Vaszil[2]

[1] Institute of Computer Science
Faculty of Philosophy and Science
Silesian University in Opava
Bezručovo nám. 13, 74601 Opava, Czech Republic
{lucie.ciencialova, alica.kelemenova}@fpf.slu.cz
[2] Computer and Automation Research Institute
Hungarian Academy of Sciences
Kende utca 13-17, 1111 Budapest, Hungary
{csuhaj, vaszil}@sztaki.hu
[3] Department of Algorithms and Their Applications
Faculty of Informatics
Eötvös Loránd University
Pázmány Péter sétány 1/c, H-1117 Budapest, Hungary
[4] Department of Computer Science
Catholic University Ružomberok
Nám. A. Hlinku 56, 03401 Ružomberok, Slovakia

Summary. 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.

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.

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 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:

- $a \rightarrow b$ (internal point mutation), specifying that an object $a \in V$ inside the cell is changed to $b \in V$.
- $c \leftrightarrow d$ (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.
- $c \leftrightarrow d / c \leftrightarrow d'$ (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'.
- $c \leftrightarrow d / c \rightarrow d'$ (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.
- $\langle a, out; b \to cd\rangle$ with $a, b, c, d \in V$ (insertion programs), specifying that if ab is inside the cell, then a is sent out 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 $\Pi = (V, e, F, C_1, \dots, C_n)$ as above, a configuration can be formally written as an $(n + 1)$-tuple

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

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

A transition from a configuration to another is denoted as

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

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

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

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

where $(e^i, \dots, e^i, \varepsilon)$, $i \in \{1, 2\}$, is the initial configuration, $(v_1, \dots, v_n, v_E)$ 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 $j \ge 1$ cells of capacity $i \in \{1, 2\}$, having at most $k \ge 1$ 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

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

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,

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

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

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

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,

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

In the following we strengthen this result by showing that P colonies with six components generate all vectors even if checking rules are not used.

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

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:

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

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:

$$\begin{aligned} P_1 &\supset \{\langle e \to l_r'\rangle, \langle l_r' \leftrightarrow e\rangle, \langle e \to l_r''\rangle, \langle l_r'' \leftrightarrow e\rangle \mid l_r \in H\} \cup \\ &\quad\ \{\langle e \leftrightarrow D'\rangle, \langle D' \to l_0\rangle, \langle l_0 \leftrightarrow D\rangle\}, \\ P_2 &\supset \{\langle e \to D'\rangle, \langle D' \to D'\rangle, \langle D' \leftrightarrow l_1'\rangle, \langle l_1' \to D\rangle, \langle D \leftrightarrow l_1''\rangle\}. \end{aligned}$$

With these programs, from the configuration $(e, e, e, e, e, e; \varepsilon)$, we obtain $(D, l_1'', e, e, e, e; l_0 w)$ 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
$i_1 : \langle D \leftrightarrow a_{r,i} \rangle$	$i_1 : \langle e \leftrightarrow l_i \rangle$
$i_2 : \langle a_{r,i} \rightarrow a_r \rangle$	$i_2 : \langle l_i \rightarrow a_{r,i} \rangle$
$i_3 : \langle a_r \leftrightarrow K_j \rangle$	$i_3 : \langle a_{r,i} \leftrightarrow l_i' \rangle$
$i_4 : \langle a_r \leftrightarrow K_k \rangle$	$i_4 : \langle a_{r,i} \rightarrow t \rangle$
$i_5 : \langle K_j \rightarrow l_j \rangle$	$i_5 : \langle l_i' \rightarrow K_j \rangle$
$i_6 : \langle K_k \rightarrow l_k \rangle$	$i_6 : \langle l_i' \rightarrow K_k \rangle$
$i_7 : \langle l_j \leftrightarrow D \rangle$	$i_7 : \langle K_j \leftrightarrow e \rangle$
$i_8 : \langle l_k \leftrightarrow D \rangle$	$i_8 : \langle K_k \leftrightarrow e \rangle$
	$i_9 : \langle t \rightarrow t \rangle$

It is not difficult to follow how the interplay of these two cells produce the configuration $(D, l_1'', e, e, e, e; l_j a_r w')$ or $(D, l_1'', e, e, e, e; l_k a_r w')$ from a configuration $(D, l_1'', e, e, e, e; l_i w)$ where w, w' are multisets of l_i', l_i'' for $l_i \in H$ and a_r $1 \leq r \leq m$. 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
$f_1 : \langle D \leftrightarrow L_f \rangle$	$f_1 : \langle e \leftrightarrow l_f \rangle$	$f_1 : \langle e \leftrightarrow L_f' \rangle$	$f_1 : \langle e \leftrightarrow L_f'' \rangle$
$f_2 : \langle L_f \rightarrow E_f \rangle$	$f_2 : \langle l_f \rightarrow L_f \rangle$	$f_2 : \langle L_f' \rightarrow l_f' \rangle$	$f_2 : \langle L_f'' \rightarrow l_f' \rangle$
$f_3 : \langle E_f \rightarrow F_f \rangle$	$f_3 : \langle L_f \leftrightarrow l_f' \rangle$	$f_3 : \langle l_f' \leftrightarrow a_r \rangle$	$f_3 : \langle l_f' \leftrightarrow \$_f \rangle$
$f_4 : \langle F_f \rightarrow \$_f \rangle$	$f_4 : \langle l_f' \rightarrow L_f' \rangle$	$f_4 : \langle l_f' \leftrightarrow \$_f \rangle$	$f_4 : \langle \$_f \rightarrow l_g \rangle$
$f_5 : \langle \$_f \leftrightarrow D \rangle$	$f_5 : \langle L_f' \leftrightarrow l_f'' \rangle$	$f_5 : \langle \$_f \rightarrow \bar{l}_n \rangle$	$f_5 : \langle l_g \leftrightarrow e \rangle$
	$f_6 : \langle l_f'' \rightarrow L_f''' \rangle$	$f_6 : \langle a_r \rightarrow e \rangle$	$f_6 : \langle l_f' \leftrightarrow \bar{l}_n \rangle$
	$f_7 : \langle L_f''' \rightarrow L_f'' \rangle$	$f_7 : \langle \bar{l}_n \leftrightarrow e \rangle$	$f_7 : \langle \bar{l}_n \rightarrow l_n \rangle$
	$f_8 : \langle L_f'' \leftrightarrow e \rangle$		$f_8 : \langle l_n \leftrightarrow e \rangle$
	$f_9 : \langle L_f \rightarrow t \rangle$		
	$f_{10} : \langle L_f' \rightarrow t \rangle$		
	$f_{11} : \langle t \rightarrow t \rangle$		

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	$D a_r[\ldots]$	f_2	f_5	$-$	$-$
6.	E_f	l''_f	e	e	$L'_f D a_r[\ldots]$	f_3	f_6	f_1	$-$
7.	F_f	L'''_f	L'_f	e	$D a_r[\ldots]$	f_4	f_7	f_2	$-$
8.	$\$_f$	L''_f	l'_f	e	$D a_r[\ldots]$	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 we obtain from a configuration $(D, l''_1, e, e, e, e; l_f a_r[\ldots])$ a new one $(D, l''_1, e, e, e, e; l_g[\ldots])$ 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 $PCOL(\text{s-c}, i, j)$ 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 1. (a) Every sender cell in a P colony 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 accepting the Parikh vectors of the words in L, we use the programs

$$\langle e, out; e \to eS \rangle,\ \langle e, out; S \to aB \rangle$$

for each $S \to aB$ of P, 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) Every consumer cell in a P colony 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

$$\langle a, in; xq_i \to q_j\rangle, x \in T \cup \{e\}.$$

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

Proof. We simulate the computations of an m-register machine $M = (m, H, l_0, l_h, P)$, $m \geq 1$, 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:

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

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

$$(l_il_i, ee, ee; w) \Rightarrow (a_rl, ee, ee; l_iw) \Rightarrow (ll, ee, l_iT_1; a_rw)$$

where l is the label of the next instruction, that is, we either have $(l_jl_j, ee, l_iT_1; a_rw)$ or to $(l_kl_k, ee, l_iT_1; a_rw)$. The contents of cell C_3, l_iT_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 $(l_il_i, ee, \bar{l}_iT_4; \bar{\bar{l}}_iw)$ and we need the steps $(l_il_i, ee, \bar{l}_iT_4; \bar{\bar{l}}_iw) \Rightarrow (a_rl, ee, \bar{\bar{l}}_iT_5; l_iw) \Rightarrow (ll, ee, l_iT_1; a_rw)$ 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 $l_r \in H$, we have the following programs.

P_1	P_2	P_3
$j_1 : \langle l_j, out; l_j \to l'_jl'_j\rangle$	$j_1 : \langle l_j, in; ee \to e\rangle$	$j_1 : \langle l'_j, in; ee \to T_1\rangle$
$j_2 : \langle l'_j, out; l'_j \to l''_jl''_j\rangle$	$j_2 : \langle a_r, in; el_j \to e\rangle$	$j_2 : \langle e, in; l'_jT_1 \to T_2\rangle$
$j_3 : \langle l''_j, out; l''_j \to l'''_jl^{iv}_j\rangle$	$j_3 : \langle l''_j, in; el_j \to e\rangle$	$j_3 : \langle l''_j, in; eT_2 \to T_3\rangle$
$j_4 : \langle l'''_j, out; l^{iv}_j \to \bar{l}_k\bar{l}_k\rangle$	$j_4 : \langle l'''_j, in; a_re \to e\rangle$	$j_{4,r} : \langle \bar{l}_r, in; l''_jT_3 \to T_4\rangle$
$j_5 : \langle l^{iv}_j, out; l'''_j \to \bar{l}_l\bar{l}_l\rangle$	$j_5 : \langle e, in; l'''_je \to e\rangle$	$j_{5,r} : \langle \bar{l}_r, in; eT_2 \to T_4\rangle$
$j_6 : \langle \bar{l}_k, out; \bar{l}_k \to \bar{\bar{l}}_k\bar{\bar{l}}_k\rangle$	$j_6 : \langle l^{iv}_j, in; a_re \to K\rangle$	$j_{6,r} : \langle \bar{\bar{l}}_r, in; \bar{l}_rT_4 \to T_5\rangle$
$j_7 : \langle \bar{\bar{l}}_k, out; \bar{\bar{l}}_k \to l_kl_k\rangle$	$j_7 : \langle e, in; l^{iv}_jK \to K\rangle$	$j_{7,r} : \langle e, in; \bar{\bar{l}}_rT_5 \to e\rangle$
$j_8 : \langle \bar{l}_l, out; \bar{l}_l \to \bar{\bar{l}}_l\bar{\bar{l}}_l\rangle$	$j_8 : \langle e, in; eK \to K\rangle$	
$j_9 : \langle \bar{\bar{l}}_l, out; \bar{\bar{l}}_k \to l_ll_l\rangle$	$j_9 : \langle l'''_j, in; l''_je \to K\rangle$	
	$j_{10} : \langle e, in; l'''_jK \to K\rangle$	
	$j_{11} : \langle l^{iv}_j, in; l''_je \to e\rangle$	
	$j_{12} : \langle e, in; l^{iv}_je \to e\rangle$	

In the following table 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.

	configuration of Π				programs to be applied		
	C_1	C_2	C_3	Env	P_1	P_2	P_3
1.	l_jl_j	ee	?	a_rw'	j_1	$-$	?
2.	$l'_jl'_j$	ee	?	l_ja_rw''	j_2	j_1	?
3.	$l''_jl''_j$	l_je	ee	l'_ja_rw	j_3	j_2	j_1
4.	$l'''_jl^{iv}_j$	a_re	l'_jT_1	l''_jw	j_4/j_5	$-$	j_2
5.	$\bar{l}_k\bar{l}_k/\bar{l}_l\bar{l}_l$	a_re	eT_2	$(l'''_j/l^{iv}_j)l''_jw$	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'''_je/l^{iv}_jK	l''_jT_3	$(\bar{l}_k/\bar{l}_l)w$	j_7/j_9	j_5/j_7	$j_{4,k}/j_{4,l}$
7.	l_kl_k/l_ll_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.	$l'_kl'_k/l'_ll'_l$	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.	$l''_kl''_k/l''_ll''_l$	$(l_k/l_l)e/eK$	ee	$(l'_k/l'_l)w$	k_3/l_3	k_2/j_8	j_1

We see that starting with a configuration where C_1 contains the objects l_jl_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.

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.

	configuration of Π				rules to be applied		
	C_1	C_2	C_3	Env	P_1	P_2	P_3
1.	l_jl_j	ee	?	w	j_1	$-$	?
2.	$l'_jl'_j$	ee	?	l_jw	j_2	j_1	?
3.	$l''_jl''_j$	l_je	ee	l'_jw	j_3	$-$	j_1
4.	$l'''_jl^{iv}_j$	l_je	l'_jT_1	l''_jw	j_4/j_5	j_3	j_2
5.	$\bar{l}_k\bar{l}_k/\bar{l}_l\bar{l}_l$	l''_je	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'''_jK/l^{iv}_je	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_kl_k/l_ll_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.	$l'_kl'_k/l'_ll'_l$	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.	$l''_kl''_k/l''_ll''_l$	$eK/(l_k/l_l)e$	ee	$(l'_k/l'_l)w$	k_3/l_3	j_8/k_2	j_1

In this case, similarly to the previous one, we either get the objects l_kl_k in the cell C_1, or the computation will not be able to halt.

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_iT_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 $\bar{l}_jT_4$ and $\bar{\bar{l}}_jT_5$ in the first two rows where the programs $i_{6,j}$ and $i_{7,j}$ are applied. In this case $w'' = \bar{\bar{l}}_jw$, 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.

Acknowledgements

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 no. 1/0692/08 (A. Kelemenová).

References

1. L. Cienciala, L. Ciencialová, A. Kelemenová: On the number of agents in P colonies. *Membrane Computing. 8th International Workshop, WMC 2007. Thessaloniki, Greece, June 25-28, 2007. Revised Selected and Invited Papers* (G. Eleftherakis, P. Kefalas, Gh. Păun, G. Rozenberg, A. Salomaa eds.), LNCS 4860, Springer, 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 (2006), 201–215.
4. J. Kelemen, A. Kelemenová: A grammar-theoretic treatment of multi-agent systems. *Cybernetics and Systems*, 23 (1992), 621–633.
5. J. Kelemen, A. Kelemenová, Gh. Păun: Preview of P colonies: A biochemically inspired computing model. *Workshop and Tutorial Proceedings. Ninth International Conference on the Simulation and Synthesis of Living Systems (Alife IX)* (M. Bedau et al., eds.), 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, Berlin, 2002.

Cell-like Versus Tissue-like P Systems by Means of Sevilla Carpets

Daniel Díaz-Pernil[1], Pilar Gallego-Ortiz[2], Miguel A. Gutiérrez-Naranjo[2], Mario J. Pérez-Jiménez[2], Agustín Riscos-Núñez[2]

[1] Research Group on Computational Topology and Applied Mathematics
Departament of Applied Mathematics I
University of Sevilla, Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
sbdani@us.es

[2]Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla, Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
pigaor@gmail.com; {magutier,marper,ariscosn}@us.es

Summary. Sevilla Carpets are a handy tool for comparing computations performed by different systems solving the same problem. Such Sevilla Carpets provide on one hand quantitative information through parameters such as Weight, Surface and Average weight, and on the other hand they also provide a fast glimpse on the complexity of the computation thanks to their graphical representation.

Up to now, Sevilla Carpets were only used on Cell-like P systems. In this paper we present a first comparison by means of Sevilla Carpets of the computations of three P systems (designed within different models), all of them solving the same instance of the Subset Sum problem. Two of these solutions use Cell-like P systems with active membranes, while the third one uses Tissue-like P systems with cell division.

1 Introduction

Comparing two cellular designs that solve the same problem is not an easy task, as there are many ingredients to be taken into account. Moreover, in the case of P systems where the number of membranes increases along the computation, the problem of describing the complexity of the computational process becomes specially hard. The complexity in *time* (number of parallel cellular steps) of these solutions is polynomial, but it is clear that the time is not the unique variable that we need to consider in order to evaluate the complexity of the process. This fact has been observed previously in the literature of P systems. The first paper related to this problem was [1], where Ciobanu, Păun and Ştefănescu presented a new way to describe the complexity of a computation in a P system. The so-called

Sevilla Carpet was introduced as an extension of the notion of Szilard language from grammars to the case when several rules are used at the same time.

In [4], the problem was revisited, introducing new parameters for the study of the descriptive complexity of P systems. Besides, several examples of a graphical representation were provided, and the utility of these parameters for comparing different solutions to a given problem was discussed. In that paper two different solutions of the Subset Sum problem, running on the same instance, were compared by using these parameters.

In this paper we adapt Sevilla Carpets to tissue-like models, in order to describe the complexity of the computations.

Note that given two Sevilla Carpets corresponding to P systems from different models designed to solve a decision problem, we can obtain detailed information about two single computations, but this is not enough to compare the efficiency of the two models in general.

Nonetheless, the numerical parameters obtained from these two Sevilla Carpets can give us some hints to compare the corresponding designs of solutions to the problem.

The paper is organized as follows. In Section 2 we recall the definition of tissue-like P systems with cell division. Section 3 shows a solution of the Subset Sum problem in the model above presented. In Section 4 we revisit the definition of Sevilla Carpets and its associated parameters. Section 5 shows a comparison among two different solutions to the Subset Sum problem in the framework of P systems with active membranes and one solution designed with tissue-like P systems with cell division, all of them running on the same instance. Some final remarks are also provided.

2 Tissue-like P Systems with Cell Division

Tissue-like P systems with cell division is a well-established P system model presented by Gh. Păun *et al.* in [8]. In this section we briefly recall its main features. The biological inspiration for this model is that alive tissues are not *static* network of cells, since cells are duplicated via mitosis in a natural way.

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

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

where:

1. Γ is a finite *alphabet*, whose symbols will be called *objects*.
2. $\mathcal{E} \subseteq \Gamma$.
3. $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells in the initial configuration.
4. $\mathcal{R}$ is a finite set of rules of the following form:
   (a) *Communication rules*: $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$.

(b) *Division rules*: $[a]_i \to [b]_i[c]_i$, where $i \in \{1, 2, \dots, q\}$ and $a, b, c \in \Gamma$.

5. $i_0 \in \{0, 1, 2, \dots, q\}$.

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

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

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

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

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

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

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

2.1 Recognizer Tissue-like P Systems with Cell Division

Complexity classes within Membrane Computing have been usually studied in the framework of *decision problems*. Let us recall that a decision problem is a pair (I_X, θ_X) where I_X is a language over a finite alphabet (whose elements are called *instances*) and θ_X is a total boolean function over I_X.

In order to study the computational efficiency for solving **NP**-complete decision problems, a special class of tissue P systems with cell division is introduced in [8]: *recognizer tissue P systems*. The key idea of such recognizer systems is the same one as from recognizer P systems with cell-like structure.

Recognizer cell-like P systems were introduced in [10] and they are the natural framework to study and solve decision problems within Membrane Computing, since deciding whether an instance of a given problem has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizer cell-like P systems are associated with P systems with *input* in a natural way. The data encoding to an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation (`yes` or `no`) is sent to the environment, in the last step of the computation. In this way, cell-like P systems with input and external output are devices which can be seen as black boxes, in the sense that the user provides the data before the computation starts, and then waits *outside* the P system until it sends to the environment the output in the last step of the computation.

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

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

where

- $(\Gamma, \mathcal{E}, w_1, \dots, w_q, \mathcal{R}, i_0)$ is a tissue-like P system with cell division of degree $q \geq 1$ (as defined in the previous section), $i_0 = env$ and $w_1, \dots, w_q$ strings over $\Gamma \setminus \Sigma$.
- The working alphabet Γ has two distinguished objects `yes` and `no`, present in at least one copy in some initial multisets $w_1, \dots, w_q$, but not present in $\mathcal{E}$.
- Σ is an (input) alphabet strictly contained in Γ.
- $i_{in} \in \{1, \dots, q\}$ is the input cell.
- All computations halt.
- If $\mathcal{C}$ is a computation of Π, then either the object `yes` or the object `no` (but not both) must have been released into the environment, and only in the last step of the computation.

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

Definition 1. *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) \mid n \in \mathbb{N}\}$ of recognizer tissue-like P systems with cell division if the following holds:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$.*
- *There exists a pair (cod, s) of polynomial-time computable functions over I_X (called a polynomial encoding of I_X in $\mathbf{\Pi}$) such that:*
  - *for each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$;*
  - *the family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (X, cod, s), that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and, moreover, it performs at most $p(|u|)$ steps;*
  - *the family $\mathbf{\Pi}$ is* sound *with regard to (X, cod, s), that is, for each $u \in I_X$, if there exists an accepting computation of $\Pi(s(u))$ with input $cod(u)$, then $\theta_X(u) = 1$;*
  - *the family $\mathbf{\Pi}$ is* complete *with regard to (X, cod, s), that is, for each $u \in I_X$, if $\theta_X(u) = 1$, then every computation of $\Pi(s(u))$ with input $cod(u)$ is an accepting one.*

In the above definition we have defined every P system $\Pi(n)$ to be *confluent*, in the following sense: every computation of a system with the *same* input multiset must always give the *same* answer.

3 A Solution for the Subset Sum Problem

For the study of the Sevilla Carpet in tissue-like P systems with Cell division we take the computation of one P systems of the family presented in [2]. In such a paper a uniform family of tissue-like P systems with cell division solving the Subset Sum problem was presented.

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

The Subset Sum problem was solved in a linear time by a family of recognizer tissue P systems with cell division. The resolution was addressed via a brute force algorithm.

A tuple $(n, (w_1, \ldots, w_n), k)$ was used to represent an instance of the problem, where n stands for the size of $A = \{a_1, \ldots, a_n\}$, $w_i = w(a_i)$, and k is the constant given as input for the problem.

Let $A = \{a_1, \ldots, a_n\}$ be a finite set, $w : A \longrightarrow \mathbb{N}$ a weight function with $n = |A|$ and $k \in \mathbb{N}$. Let $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be a *function* defined by $g(n, k) = ((n + k)(n + k + 1/2)) + n$. This function is primitive recursive and bijective between $\mathbb{N}^2$ and

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

We will provide a family of tissue P systems where each P system solves all the instances of the `SUBSET SUM` problem with the same size. The weight function w of the concrete instance will be provided via an input multiset determined via the function $cod(u) = \{\{v_i^j : w(a_i) = j \wedge 1 \leq i \leq n\}\} \cup \{\{q^k\}\}$, where v_i^j (i.e., j copies of object v_i) represents that j is the weight of the element a_i.

Next, we will provide a family of recognizer tissue P systems with cell division which solve the `SUBSET SUM` problem in linear time. For each $(n, k) \in \mathbb{N}^2$ we will consider the system $\Pi(n, k) = (\Gamma, \Sigma, \omega_1, \omega_2, \mathcal{R}, \mathcal{E}, i_{in}, i_0)$, where

- $\Gamma = \Sigma(n) \cup \{A_i, B_i \mid 1 \leq i \leq n\}$
  $\cup\ \{a_i \mid 1 \leq i \leq n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 11\}$
  $\cup\ \{c_i \mid 1 \leq i \leq n+1\}$
  $\cup\ \{d_i \mid 1 \leq i \leq \lceil \log n \rceil + \lceil \log(k+1) \rceil + 4\}$
  $\cup\ \{e_i \mid 1 \leq i \leq \lceil \log n \rceil + 1\}$
  $\cup\ \{B_{ij} \mid 1 \leq i \leq n \wedge 1 \leq j \leq \lceil log(k+1) \rceil + 1\}$
  $\cup\ \{b, D, p, q, g_1, g_2, f_1, T, S, N, \texttt{yes}, \texttt{no}\}$
- $\Sigma = \{v_i \mid 1 \leq i \leq n\}$
- $\omega_1 = a_1\, b\, c_1\, \texttt{yes}\, \texttt{no}$
- $\omega_2 = D A_1 \cdots A_n$
- $\mathcal{R}$ is the following set of rules:
  1. *Division rules:*
     $r_{1,i} \equiv [A_i]_2 \rightarrow [B_i]_2[\lambda]_2$ for $i = 1, \ldots, n$
  2. *Communication rules:*
     $r_{2,i} \equiv (1, a_i/a_{i+1}, 0)$ for $i = 1, \ldots, n + \lceil \log n \rceil + \lceil \log(k+1) \rceil + 10$
     $r_{3,i} \equiv (1, c_i/c_{i+1}^2, 0)$ for $i = 1, \ldots, n$
     $r_4 \equiv (1, c_{n+1}/D, 2)$
     $r_5 \equiv (2, c_{n+1}/d_1 e_1, 0)$
     $r_{6,i} \equiv (2, e_i/e_{i+1}^2, 0)$ for $i = 1, \ldots, \lceil \log n \rceil$
     $r_{7,i} \equiv (2, d_i/d_{i+1}, 0)$ for $i = 1, \ldots, \lceil \log n \rceil + \lceil \log(k+1) \rceil + 3$
     $r_{8,i} \equiv (2, e_{\lceil \log n \rceil + 1} B_i/B_{i1}, 0)$ for $i = 1, \ldots, n$
     $r_{9,i,j} \equiv (2, B_{ij}/B_{ij+1}^2, 0)$ for $i = 1, \ldots, n$, $j = 1, \ldots, \lceil \log(k+1) \rceil$
     $r_{10,i} \equiv (2, B_{i\lceil \log(k+1) \rceil + 1} v_i/p, 0)$ for $i = 1, \ldots, n$
     $r_{11} \equiv (2, pq/\lambda, 0)$
     $r_{12} \equiv (2, d_{\lceil \log n \rceil + \lceil \log(k+1) \rceil + 4}/g_1 f_1, 0)$
     $r_{13} \equiv (2, f_1 p/\lambda, 0)$
     $r_{14} \equiv (2, f_1 q/\lambda, 0)$
     $r_{15} \equiv (2, g_1/g_2, 0)$
     $r_{16} \equiv (2, g_2 f_1/T, 0)$
     $r_{17} \equiv (2, T/\lambda, 1)$
     $r_{18} \equiv (1, bT/S, 0)$
     $r_{19} \equiv (1, S\texttt{yes}/\lambda, 0)$
     $r_{20} \equiv (1, a_{n+\lceil \log n \rceil + \lceil \log(k+1) \rceil + 11} b/N, 0)$

$r_{21} \equiv (1, N\mathtt{no}/\lambda, 0)$

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

An overview of the computation and more details of the design can be read in [2].

4 Sevilla Carpets

Sevilla Carpets were presented in [1] as an extension of the Szilard language, which consists of all strings of rule labels describing correct derivations in a given grammar (see e.g., [6, 7] or [11]). The Szilard language is usually defined for grammars in the Chomsky hierarchy where only a single rule is used in each derivation step, so a derivation can be represented as the string of the labels of the rules used in the derivation (the labeling is supposed to be one-to-one). Sevilla Carpets are a Szilard-way to describe a computation in a P system. The main difference is that a multiset of rules can be used in each evolution step of a P system. In [1] a bidimensional writing is proposed to describe a computation of a P system. The (Sevilla) Carpet associated with a computation of a P system is a table with the time on the horizontal axis and the rules explicitly mentioned along the vertical axis; then, for each rule, in each step, a piece of information is given. Depending on the amount of information given to describe the evolution, Ciobanu, Păun and Ştefănescu propose five variants for the Sevilla Carpets:

1. Specifying in each time unit for each membrane whether at least one rule was used in its region or not;
2. Specifying in each time unit for each rule whether it was used or not;
3. Mentioning in each time unit the number of applications of each rule; this is 0 when the rule is not used and can be arbitrarily large when the rules are dealing with arbitrarily large multisets;
4. We can also distinguish three cases: that a rule cannot be used, that a rule can be used but it is not because of the nondeterministic choice and that a rule is actually used;
5. A further possibility is to assign a cost to each rule, and to multiply the number of times a rule is used with its cost.

They also propose two parameters (*weight* and *surface*) to study Sevilla Carpets. In [4] two new parameters (*height* and *average weight*) were proposed.

4.1 Parameters for the Descriptive Complexity

Many times we will not be interested only in the number of cellular steps of the computation, but also in other type of resources required to perform the computation. Specially if we want to implement *in silico* a P system, we need to be careful with the number of times that a rule is applied, maybe with the number of membranes and/or the number of objects present in a given configuration.

In order to describe the complexity of the computation, the following parameters are proposed:

- **Weight:** It is defined in [1] as the sum of all the elements in the carpet, i.e., as the total number of applications of rules along the computation. The application of a rule has a cost and the weight measures the total cost of the computation.
- **Surface:** It is the multiplication of the number of steps by the total number of the rules used by the P system. It can be considered as the *potential size* of the computation. From a computational point of view we are not only interested on P systems which halt in a small number of steps, but in P systems which use a small amount of resources. The *surface* measures the resources used in the design of the P system. Graphically, it represents the surface where the Sevilla Carpet lies on.
- **Height:** It is the maximum number of applications of any rule in a step along the computation. Graphically, it represents the highest point reached by the Sevilla Carpet.
- **Average Weight:** It is calculated by dividing the *weight* to the *surface* of the Sevilla Carpet. This concept provides a relation between both parameters which gives an index on how the P system exploits its massive parallelism.

5 Comparing the Solutions

In [4], two uniform families of P systems with active membranes solving the Subset Sum were presented. In both solutions, a tuple $(n, (w_1, \ldots, w_n), k)$ is used to represent an instance of the problem, where n stands for the size of $A = \{a_1, \ldots, a_n\}$, $w_i = w(a_i)$, and k is the constant given as input for the problem. Both solutions are based on a brute force algorithm implemented in the framework of P systems with active membranes. The idea of the design can be divided into several stages:

- *Generation stage*: for every subset of A, a membrane is generated via membrane division.
- *Weight calculation stage*: in each membrane the weight of the associated subset is calculated. This stage will take place in parallel with the previous one.
- *Checking stage*: in each membrane it is checked whether or not the weight of its associated subset is exactly k. This stage cannot start in a membrane before the previous ones are over in that membrane.
- *Output stage*: when the previous stage has been completed in all membranes, the system sends out the answer to the environment.

The first family can be found in [9]. Let us recall that the instance $u = (n, (w_1, \ldots, w_n), k)$ is processed by the P system $\Pi_1(\langle n, k\rangle)$ with input the multiset $x_1^{w_1} x_2^{w_2} \ldots x_n^{w_n}$.

This design depends on the two constants that are given as input in the problem: n and k. It consists on $5n + 5k + 18$ evolution rules, and if an appropriate input multiset is introduced inside membrane e before starting the computation, the system will stop and output an answer in $2n + 2k + 6$ steps (if the answer is No) or in $2n + 2k + 5$ steps (if the answer is Yes).

The second family is inspired in the previous one. Some modifications were made following the design presented in [3]. In this solution the instance $u = (n, (w_1, \dots, w_n), k)$ is processed by the P system $\Pi_2(n)$ with input the multiset $x_1^{w_1} x_2^{w_2} \dots x_n^{w_n}$.

The above design depends only on one of the constants that are given as input in the problem: n. It is quite similar to the previous one, the difference lies in the checking stage and the answer stage. In this case we avoid the use of counters that require knowing the constant k.

The number of evolution rules is $5n + 41$, and the number of steps of the computation depends on the concrete instance that we need to solve, but it is linearly bounded.

We compare these solutions with the solution described in Section 3. As pointed out in [2], the number of rules for a set $A = \{a_1 \dots, a_n\}$ of size n is $\dots$ and the number of steps is ... if the answer is `Yes` and ... if the answer is `No`.

5.1 Descriptive Complexity

We present some detailed statistics about the previous designs, trying to compare them on a more general basis than just looking the number of steps that the computation performs. Following this scheme, we present the Sevilla Carpets associated with the computations of the three different solutions to the Subset Sum problem working on the same instance: $u = (5, (3, 5, 3, 2, 5), 9)$. That is, $n = 5$, $k = 9$ and the list of weights is $w_1 = 3, w_2 = 5, w_3 = 3, w_4 = 2, w_5 = 5$.

The P system $\Pi_1(\langle 5, 9\rangle)$ has 88 evolution rules, and all of them are applied with the exception of the rules: $[q_{19}]_e^- \to [\ ]_e^0 Yes$, $[q_3]_e^- \to [\ ]_e^- \#$, $[q_9]_e^- \to [\ ]_e^- \#$ and $[Yes]_s^- \to [\ ]_s^0 Yes$. The P system $\Pi_1(5, 9)$ stops at step 33 and sends an object No to the environment.

The weight of the Sevilla Carpet (the total number of rule applications along the computation) is 2179. The height of the Sevilla Carpet (the maximal number of times that a rule is applied in one evolution step) is 82 and it is reached at Step 9. The surface of the Sevilla Carpet is 2904. The average weight of the Sevilla Carpet is 0.749656

The P system $\Pi_2(5)$ has 65 evolution rules, and all of them are applied with the exception of the rules: $[q_3]_e^0 \to [\ ]_e^+ Yes$ and $[Yes]_s^- \to [\ ]_s^0 Yes$. The P system $\Pi_2(5)$ stops at step 38 and sends an object No to the environment.

The weight of the Sevilla Carpet is 3368. The height of the Sevilla Carpet is 108 and it is reached at step 10. The surface of the Sevilla Carpet is 2470. The average weight of the Sevilla Carpet is 1.36275

Finally, the solution with tissue-like P systems with cell division has 88 rules and 84 of them are applied in this computation. The P system stops at step 24.

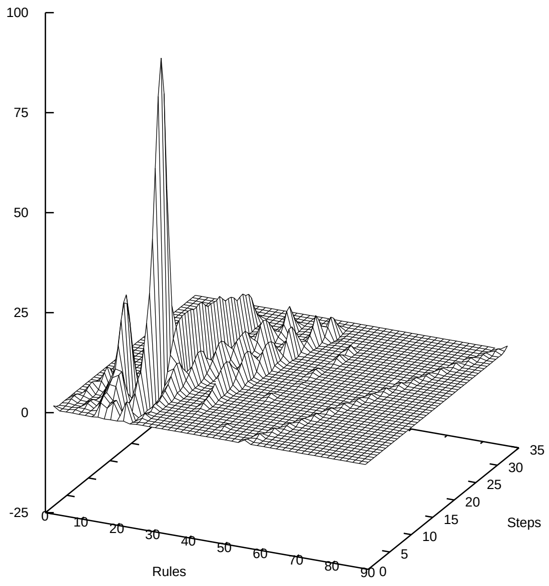


Fig. 1. Sevilla Carpet for solution 1

The surface of the Sevilla Carpet is 2112 and its weight is 2405. The height is 128 and it is reached at steps 10, 12, 13, 14 and 15. The average weight of the Sevilla Carpet is 1.13873.

The following table shows the parameters of both solutions:

	Sol. 1	Sol.2	Sol. 3
Rules	88	65	88
Steps	33	38	24
Surface	2904	2470	2112
Weight	2179	3368	2405
Height	82	108	128
Average Weight	0.749656	1.36275	1.13873

If we consider the number of steps as a complexity measure to compare the designs, then we conclude that the third solution is *better* than the other ones, since it needs less steps.

Moreover, concerning the weight of the Sevilla Carpet, solution 1 is *better* than the other ones, because it uses less resources during the computation.

Nonetheless, the key point of a design of a solution in Membrane Computing is the use of the massive parallelism. As pointed out in [5],

a bad design of a P system consists of a P system which does not exploit its parallelism, that is, working as a sequential machine: in each step only

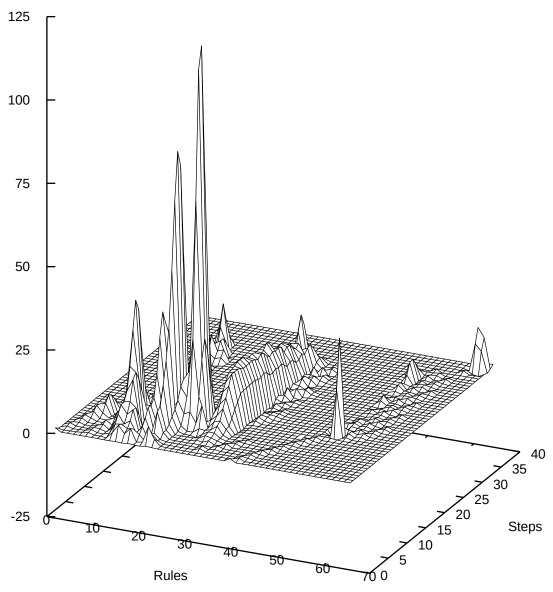


Fig. 2. Sevilla Carpet for solution 2

one object evolve in one membrane whereas the remaining objects do not evolve. On the other hand, a good design consists of a P system in which a huge amount of objects are evolving simultaneously in all membranes. If both P systems perform the same task, it is obvious that the second one is a better design that the first one.

In this line, the fact that the average weight of solution 2 is larger than the average weight of the other solutions can be interpreted saying that the second design makes a better use of the parallelism in P systems.

6 Conclusions and Future Work

It is important to remark that these are not asymptotical comparisons, as we focus only on the data corresponding to one instance. Indeed, due to the exponential number of membranes created during the generation stage, we believe that considering another instance with a greater size will stress the differences between the design based only on n and the other one, based on n and k. The bound on the size of the instances that can be studied is imposed by the necessity to use a P systems simulator to obtain the detailed description of the computation: number of rules, number of cellular steps, and number of times that the rules are applied in each step.

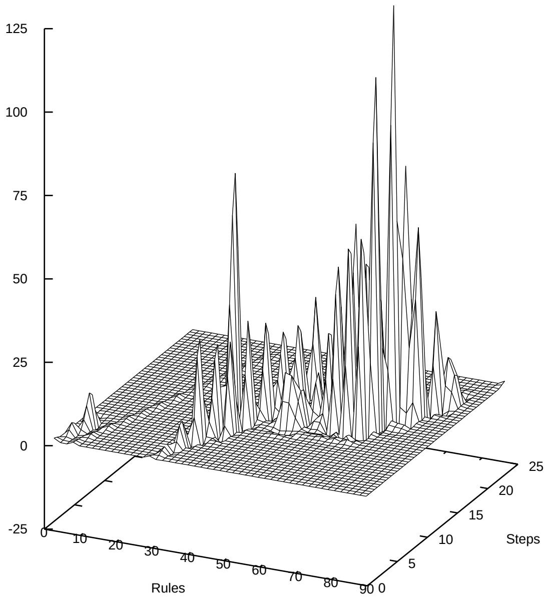


Fig. 3. Sevilla Carpet for solution 3

Acknowledgements

The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. G. Ciobanu, Gh. Păun, Gh. Ştefănescu: Sevilla Carpets Associated with P Systems. In M. Cavaliere, C. Martín–Vide and Gh. Păun (eds.), *Proceedings of the Brainstorming Week on Membrane Computing*, Tarragona, Spain, 2003, Report RGML 26/03, 135–140.
2. D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving Subset Sum in Linear Time by Using Tissue P Systems with Cell Division. In *Bio-inspired Modeling of cognitive tasks*, J. Mira and J.R. Alvarez, eds., LNCS 4527, Springer, 2007, 170–179.
3. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: Towards a programming language in cellular computing. *Proceedings of the 11th Workshop on Logic, Language, Information and Computation* (WoLLIC'2004), July 19-22, 2004, 1-16 Campus de Univ. Paris 12, Paris, France. A preliminary version in Gh. Păun, A. Riscos, A. Romero, F. Sancho, eds., *Proceedings of the Second Brainstorming Week on Membrane Computing*, Report RGNC 01/04, 2004, 247–257.

4. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: On Descriptive Complexity of P Systems. In G. Mauri, Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg, A. Salomaa, eds., *Membrane Computing.* LNCS 3365, Springer, 2005, 320–330.
5. M. A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: On the Degree of Parallelism in Membrane Systems. *Theoretical Computer Science*, 372, 2-3 (2007), 183–195.
6. E. Mäkinen: A Bibliography on Szilard Languages, Dept. of Computer and Information Sciences, University of Tampere, `http://www.cs.uta.fi/reports/pdf/Szilard.pdf`.
7. A. Mateescu, A. Salomaa: Aspects of Classical Language Theory. In G. Rozenberg, A. Salomaa, eds., *Handbook of Formal Languages* (vol. 1), Springer, Berlin, 1997.
8. Gh. Păun, M.J. Pérez-Jiménez, A. Riscos-Núñez: Tissue P System with cell division. In Gh. Păun, A. Riscos-Núñez, A. Romero-Jiménez, F. Sancho-Caparrini, eds., *Second Brainstorming Week on Membrane Computing*, Sevilla, Report RGNC 01/2004, 2004, 380–386.
9. M.J. Pérez-Jiménez, A. Riscos-Núñez: Solving the Subset Sum Problem by Active Membranes. *New Generation Computing*, 23, 4 (2005), 367–384.
10. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrini: A polynomial complexity class in P systems using membrane division. In E. Csuhaj-Varjú, C. Kintala, D. Wotschke, Gy. Vaszyl, eds., *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems, DCFS 2003*, 2003, 284–294.
11. A. Salomaa: *Formal Languages.* Academic Press, New York, 1973.

P Systems and Topology: Some Suggestions for Research

Pierluigi Frisco

School of Mathematical and Computer Sciences, Heriot-Watt University
EH14 4AS Edinburgh, UK
pier@macs.hw.ac.uk

Summary. Lately, some studies linked the computational power of abstract computing systems based on multiset rewriting to Petri nets and the computation power of these nets to their topology. In turn, the computational power of these abstract computing devices can be understood just looking at their *topology*, that is, information flow.

This line of research is very promising for several aspects:

its results are valid for a broad range of systems based on multiset rewriting;
it allows to know the computational power of abstract computing devices without tedious proofs based on simulations;
it links computational power to topology and, in this way, it opens a broad range of questions.

In this note we summarize the known result on this topic and we list a few suggestions for research together with the relevance of possible outcomes.

1 Introduction

Imagine that a computing device based on multiset rewriting is defined. Let us call this computing device S_1. This could be a P system with symport/antiport, or with catalysts, of with conformons, or it could be a definition of Diophantine equations, or some model of Brane calculi of whatever else you like.

What would you do in order to know the computing power of such a system?

Probably you would try to simulate with S_1 another computing system, say S_2 with known computational power. If this is possible, than you can say that S_1 can compute at least as much as S_2 can compute. Then you would probably try to simulate S_1 with S_2, if this is possible, then you know that the two systems have the same computational power. This is the standard way to know the computational power of a computing system.

There is another way to know the computational power of S_1. This new way is based on the fact that a computing system has a way to store information and a way to manipulate it. This other way looks at how such a system stores information and how it manipulates it and it deducts (in between other things) the computing power of S_1.

From a point of view this is not a new idea: this is a very well establish concept in formal grammars. If somebody gives you a formal grammar S_1 and asks what it can compute, probably you would not try to simulate with S_1 another grammar S_2 of known computing power. Instead, you would simply look at the productions of S_1 and from this (because of the Chomsky hierarchy) you would be able to deduct what it can be generated by S_1.

The approach that we are going to consider in this paper is about this: it shows how to know the computing power of a formal system based on multiset rewriting without running simulations but simply looking at the kind of operations that the formal system can perform.

The suggestions for research present in Section 7 have been in part inspired by conversations taking place during the 7^{th} Brainstorming Week on Membrane Computing (February 2 - 6, 2009, Seville, Spain). This note is not self contained, it has been written having in a mind readers knowledgeable in P systems and with a strong interest in Petri nets. Citation indicate where the used but not defined concepts can be found.

2 About Simulations

If you are familiar with formal language theory, then you know that productions in a formal grammar are all of the same form: $\alpha \rightarrow \beta$ with α and β strings over a certain alphabet. There is not much confusion about what such a production does and about the language generated by a grammar. On the other hand, formal systems based on multiset rewriting do not have 'standard' way to operate and do not have 'standard' ways to get the result of their computation. For this reason, if we want to 'reduce' the way these systems operate to just one way (on which we can analyze the topology), then we have to use a definition of simulation.

Let S and S' be two formal systems with O and O' their respective sets of operations and $\mathbb{C} = \{c_1, c_2, \ldots\}$ and $\mathbb{C}' = \{c'_1, c'_2, \ldots\}$ their respective sets of configurations. We denote with $\overset{\sigma}{\Rightarrow}$ ($\overset{\sigma'}{\pmb{\rightarrow}}$), σ multiset over O (σ' multiset over O'), the transition from one configuration to another in a computation of S (S') according to the application of the operations in σ (σ'). With $\overset{\sigma_1,\ldots,\sigma_n}{\Rightarrow^+}$ ($\overset{\sigma'_1,\ldots,\sigma'_n}{\pmb{\rightarrow}^+}$) we denote non-empty sequences of transitions from one configuration to another in a computation of S (S') according to the application of the operations in $\sigma_1, \ldots, \sigma_n$ ($\sigma'_1, \ldots, \sigma'_n$) in sequence. So, for instance, if $c_1 \overset{\sigma_1}{\Rightarrow} c_2 \overset{\sigma_2}{\Rightarrow} c_3$, then we can write $c_1 \overset{\sigma_1,\sigma_2}{\Rightarrow^+} c_3$.

It should be clear that, depending on the operational mode of S, the multiset σ can be a multiset of a specific kind. For instance, σ can be such that it returns at most 1.

Definition 1. *Let S and S' be two formal systems with O and O' their respective sets of operations, $\mathbb{C}$ and $\mathbb{C}'$ their respective sets of configurations and c_{init} and c'_{init} their respective initial configurations.*

We say that S simulates S' if there are two relations $\alpha \subseteq \mathbb{C} \times \mathbb{C}'$ and $\beta \subseteq O^n \times O'^m,\ n, m \in \mathbb{N}$, such that:

i) $(c_{init}, c'_{init}) \in \alpha$;

ii) for all $c_1, c_2 \in \mathbb{C},\ c'_1 \in \mathbb{C}'$ and $\sigma \in O^n$: if $c_1 \overset{\sigma}{\Rightarrow}^+ c_2$ and $(c_1, c'_1) \in \alpha$, then there is $c'_2 \in \mathbb{C}'$ such that $c'_1 \overset{\sigma'}{\Rightarrow}^+ c'_2$ with $(c_2, c'_2) \in \alpha$ and $(\sigma, \sigma') \in \beta$;

iii) for all $c'_1, c'_2 \in \mathbb{C}',\ c_1 \in \mathbb{C}$ and $\sigma' \in O^n$: if $c'_1 \overset{\sigma'}{\Rightarrow}^+ c'_2$ and $(c_1, c'_1) \in \alpha$, then there is $c_2 \in \mathbb{C}$ such that $c_1 \overset{\sigma}{\Rightarrow}^+ c_2$ with $(c_2, c'_2) \in \alpha$ and $(\sigma, \sigma') \in \beta$.

If S $\alpha\beta$ simulates S', then S is called the *simulating* system while S' is called the *simulated* system.

It is important to stress that in this paper the accepted or generated languages of simulations are related to the configurations, not to the labels associated to the operations (as in other definitions of simulation). Moreover, we have to point out that the just given definition of simulation differs substantially from the ones present in [11, 12] and from other similar definitions (specific to EN systems) as in [14].

3 Petri Nets

If we want to study how the topology of formal system based on multiset rewriting is related to their computational power, then we need one such system in which topology is clearly present. *Place/transition system* (*P/T systems*) are the ideal candidate. They are a type of Petri nets and, as such, are a (bipartite) graph. Similarly to grammars, one of the nice features of P/T systems is their simplicity: the operations that can be performed by them are very basic, no complexity hidden in the way they operate. In this way, the set of numbers that can be generated a P/T systems can be directly related to its topology, similarly to the way the language generated by a grammar can be directly related to its kind of productions.

Given an P/T system it is possible to ‘run’ it (that is, the way it fires) in different ways. This is different than other formal systems for which their way to run is embedded in their definition. For instance, in a grammar the productions are applied once per time (that is, given a sentential form at most one production is applied in order to pass to another sentential form) while in an L system productions are applied in parallel. So, if someone would give you a grammar, then you would also know the way it runs. This means that if together with a grammar you are also given a language, then it can either be that the grammar generates that language or not.

P/T systems (and Petri nets in general) do not have a unique way to run. If someone would give you a P/T system, then you could run it in different ways (each of these ways generates a *configuration graph* of a certain kind). This means that if together with the P/T system you are also given a set of numbers, then the P/T system can generate or not that set of numbers depending on the way it runs.

This separation between a P/T system and the way it runs is important to us as some of the results indicated in the following depend on a specific way to run while others are independent of the way to run.

For us it is important to say that we consider three types of Petri nets: *EN system*, *P/T systems* and *P/T′ systems* and we consider two ways they can run: SCG and $MSCG$. It is improper to refer to SCG and $MSCG$ as 'ways to run'. Here we use this improper language because we want to use a very simple terminology. The readers who are not familiar with these Petri nets concepts can find their definitions in [13, 5]. These concepts are important for understanding what in the following sections.

4 Building Blocks and Their Composition

Let us introduce the nets depicted in Fig. 1 and call them *building blocks*, *join* and *fork* in particular, as depicted in that figure. The places present in each building block are distinct.

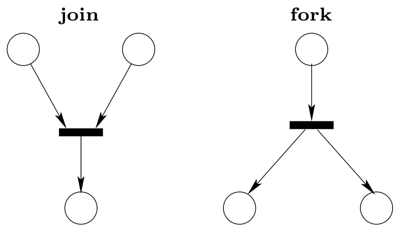


Fig. 1. Building blocks: *join* and *fork*. © With kind permission of Springer Science and Business Media [4].

Definition 2. *Let $x, y \in \{join, fork\}$ be building blocks and let $\bar{t}_x$ and $\hat{t}_y$ be the transitions present in x and y respectively.*

We say that y comes after x *(or* x is followed by y, *or* x comes before y *or* x and y are in sequence*) if $\bar{t}_x^{\bullet} \cap {}^{\bullet}\hat{t}_y \neq \emptyset$ and ${}^{\bullet}\bar{t}_x \cap {}^{\bullet}\hat{t}_y = \emptyset$. We say that* x and y are in parallel *if ${}^{\bullet}\bar{t}_x \cap {}^{\bullet}\hat{t}_y \neq \emptyset$ and $\bar{t}_x^{\bullet} \cap {}^{\bullet}\hat{t}_y = \emptyset$.*

We say that a net is composed by *building blocks (it is* composed by x*) if it can be defined by building blocks (it is defined by* x*) sharing places but not* *transitions. Consequently, we say that a Petri net is* composed by *building blocks (it is* composed by x*) if its underlying net is composed by building blocks (it is* composed by x*).*

In Fig. 2 a *join* and a *fork* are depicted in parallel, while in Fig. 3.a and Fig. 3.b *join* and *fork* are depicted in sequence.

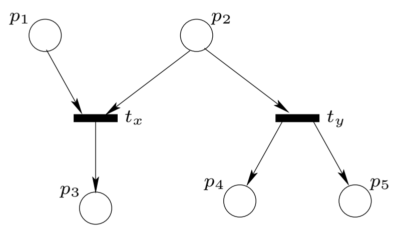


Fig. 2. A *join* and a *fork* in parallel.

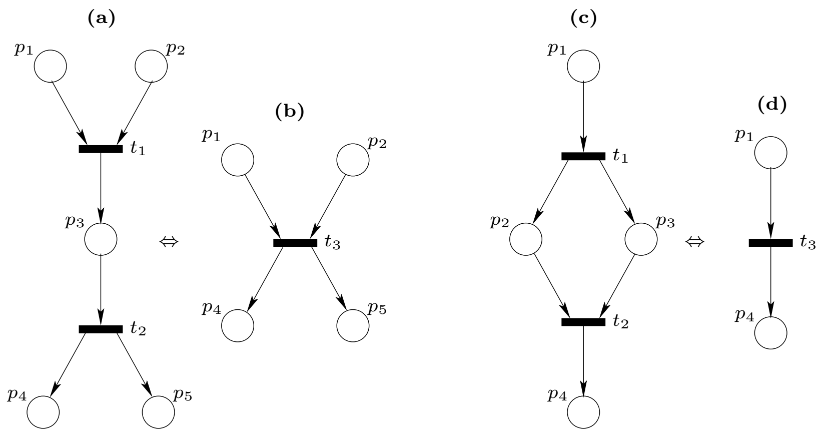


Fig. 3. **(a)** *join* and *fork* in sequence and **(b)** *fork* and *join* in sequence.

5 Known Results

Table 1 lists the known results linking topology to accepted/generated vectors of numbers.

n.	*system*	*build. blocks*	*composition*	*n. places*	*way to run*	*acc./gen.*	*class*
1	P/T	join, fork	any	finite	*SCG*	acc.	= part. blind r.m.
2	P/T	join, fork	as Fig. 3.a	finite	*MSCG*	acc.	= restricted r.m.
3	P/T	join, fork	any	finite	*MSCG*	acc./gen.	$= \mathbb{N}{\cdot}\mathsf{RE}$
4	EN	join	any	finite	*SCG*	acc.	$= \mathbb{N}{\cdot}\mathsf{FIN}$
5	P/T′	join	any	infinite	*SCG*	acc./gen.	$= \mathbb{N}{\cdot}\mathsf{RE}$
6	P/T	join	any	finite	*SCG*	acc.	$\mathbb{N}{\cdot}\mathsf{REG} \subset J \subset \mathbb{N}{\cdot}\mathsf{CS}$

Table 1. Summary of known results

In Table 1:

n. refers to the row in the table;
system indicates the kind of Petri net;
build. blocks indicated the kinds of building blocks present in the system;
composition indicates how the building blocks are composed in the system;
n. places indicates the number of places present in the system;
way to run indicates the way the system is run;
acc./gen. indicates if known results refers to the accepting or generating model;
class indicates the class of numbers accepted/generated by the considered system.

Moreover:

part. blind r.m. means *partially blind register machines* [8];
restricted r.m. means *restricted register machines* [9];
$\mathbb{N}{\cdot}\mathsf{FIN}, \mathbb{N}{\cdot}\mathsf{REG}, \mathbb{N}{\cdot}\mathsf{CS}, \mathbb{N}{\cdot}\mathsf{RE}$ denote classes of numbers [15].

Because of Theorem 2 in [5] row 3 in Table 1 holds true also for other ways to run the P/T system included P/T systems having an infinite number of places and running according to *SCG*.

6 Links with P Systems with Catalysts

In the introduction we hinted to a novel approach to study the computational power of abstract computing devices. This novel approach is based on the simulation (according to Definition 1) of *join* and *fork* and their composition. Once this is known, then the results listed in Table 1 can be used to deduct the computing power of the abstract computing device under investigation.

This has been performed for *catalytic P systems*. Here we list the results obtained using this novel approach for this model of P system. The following results can be found in [5] together with the definitions of the used notation and terminology.

Lemma 1. *The building blocks* **join** *and* **fork** *can be simulated by generating P systems with catalysts of degree 1 and with 2 catalysts.*

Lemma 2. *Generating P systems with 2 catalysts and one compartment can simulate the 0-test P/T system.*

From these two lemmas and the results in Table 1 the following holds true:

Theorem 1. $\mathbb{N}_{02} \cdot aCP(1,2) = \mathbb{N}_{02} \cdot aCatP(1,3) = \mathbb{N}_2 \cdot \mathsf{RE}$;
$\mathbb{N} \cdot gCP(2,2) = \mathbb{N} \cdot gCatP(2,3) = \mathbb{N} \cdot aCP_{-c}(1,2) = \mathbb{N} \cdot aCatP_{-c}(1,3) = \mathbb{N} \cdot \mathsf{RE}$.

Theorem 2. *The families* $\mathbb{N} \cdot gCP(2,2)$, $\mathbb{N} \cdot gCatP(2,3)$, $\mathbb{N}_{02} \cdot aCP(1,2)$, $\mathbb{N}_{02} \cdot aCatP(1,3)$, $\mathbb{N} \cdot aCP_{-c}(1,2)$, $\mathbb{N} \cdot aCatP_{-c}(1,3)$*, when maximal parallelism is not present, are the ones generated also by partially blind register machines.*

Corollary 1. *Accepting catalytic-P systems with only rules of the kind* $cx \rightarrow c$, $c \in C$ *and* $x \in V \setminus C$ *can accept only finite languages.*

Corollary 2. *Restricted P systems with catalysts of degree 2 and two catalysts and restricted catalytic-P systems of degree 2 and three catalysts can simulate restricted register machines.*

Moreover:

Corollary 3. *The class of numbers accepted by P systems with catalysts of degree 2 and 2 catalysts not using rules of the kind* $a \rightarrow b_1 b_2$ *is* J;
The class of numbers accepted by purely catalytic P systems of degree 2 and 3 catalysts not using rules of the kind $a \rightarrow b_1 b_2$ *is* J.

Where J is a class of numbers such that $\mathbb{N} \cdot \mathsf{REG} \subset J \subset \mathbb{N} \cdot \mathsf{CS}$ (that is, the one in row 6 of Table 1).

Some of the results in this section (as, for instance, Theorem 1) have previously been obtained using 'classical' (that is, simulating other devices of known computing power) methodologies [2, 10].

7 Suggestions for Research

The results in Section 6 show how far reaching the simulation of join and fork can be in respect to the direct simulation of a specific computational model. We say 'can be' and not 'is' because it is not always the case that some features considered for building blocks can be naturally translated in features of the simulating system (P systems with catalysts, in this case). Corollary 2 is an example of the difficulties in this 'translation'. How to translate in natural terms for P systems with catalysts the fact that *join* and *fork* are composed as in Fig. 3.a?

The definition of restricted P systems with catalysts considered for Corollary 2 is a very simple way to perform such a translation, but still it did not allow to say that the computational power of such P systems is equivalent to the one of restricted register machines. For this reason we formulate:

Suggestion for research 1 *Define a model of P systems with catalysts having computational power equivalent to the one of restricted register machines.*

Line 6 in Table 1 refers to the class of numbers accepted by P/T systems composed only by *join* and having a finite number of places when they run according to *SCG*. This class of number, called J in [7], is well defined: it is proved that this kind of P/T systems can only accept J. What it is not well known is how J relates to other classes of numbers.

Suggestion for research 2 *As indicated in Table 1, it is known that* $\mathbb{N}\cdot\mathsf{REG} \subset J \subset \mathbb{N}\cdot\mathsf{CS}$. *Restrict this (rather broad) interval.*

A solution to this suggestion does not necessarily mean to have a result of the kind: $A \subset J \subset B$ where A and B are classes of numbers. The indication of what part of $\mathbb{N}\cdot\mathsf{CS}$ is in J and what not, would be already of interest.

In the Introduction we said: “it shows how to know the computing power of a formal system based on multiset rewriting without running simulations but simply looking at the kind of operations that the formal system can perform”. What do we mean with ‘looking’?

In this case ‘looking’ means the way Definition 1 is implemented, that is what it is considered as configuration of the simulated system. Given a configuration of a formal system it is possible to ‘look’ at it in different ways. For instance, we could ignore some elements in the configuration, we could group different elements in the configuration, etc. This concept of ‘looking’ (observing) has been formalized (see, for instance, [1]). As a consequence of this, given a formal system, we could implement Definition 1 in different ways, leading to different models of Petri nets and, possibly, to different accepted/generated languages.

Suggestion for research 3 *Study how different implementations of Definition 1 in a given formal system change the class of numbers accepted/generated by it. Is it possible to link these results to the topology as presented in this paper?*

The previous suggestion for research can go beyond the study of a few formal systems ‘observed’ in different ways. It can include more general studies as the classification of systems that can be ‘observed’ in only one way, or in an unbounded number of ways, or that can accept/generate the same language independently of the way they are observed, etc. For instance, we could obtain results of the kind: the ‘observation’ of the systems in row 4 of Table 1 can only accept/generate finite classes of numbers, etc.

If we look at rows 2 and 3 in Table 1 you notice that the difference on the accepted class of numbers is due to the allowed composition. Here one could imagine that if the composition would be something in between what present in these two rows, then the accepted language would also be ‘in between’. We try to clarify this point. In row 2 we deal with systems in which *join* and *fork* can only be in sequence as in Fig. 3.a, while in row 3 we deal with systems in which the two building blocks can be composed in any way. What language can be accepted by

P/T systems having a finite number of *join* and *fork* in which only a few are in sequence as in Fig. 3.a?

This line of research could be called "language generated by pseudo-random Petri nets" where 'random' refers to the fact that the Petri net is created composing building blocks in a random way, while 'pseudo' refers to the fact that we impose some limitations to this randomness as, for instance, the fact that a few *join* and *fork* have to be in a specific sequence.

Suggestion for research 4 *Study languages generated by pseudo-random Petri nets.*

This line of research requires the creation of some computer programs (or the use of already available computer programs). Of course, a similar line of research can be pursued for P systems and any other kind of computing device that can be build in a pseudo-random way.

In Section 6 we indicated how the results in Table 1 have been used on P systems with catalysts. We tried to use similar results on other models of P systems (symport/antiport, conformons, etc., see [6]) and all went as we expected. This means that using Definition 1 and the results in Table 1 we obtained results similar, in terms of descriptional complexity, to the ones already known. The exception to this were *spiking P systems*. We did not succeed in defining a simulation (as in Definition 1) that let us re-obtain the known results on this model of P systems with the same descriptional complexity. The results we got needed more features (more compartments, the presence of forgetting rules, etc.) not present in direct proofs.

This fact is particularly intriguing because it could suggest some limits in the approach considered in this note. For this reason we propose:

Suggestion for research 5 *Use the approach considered in this paper on spiking P systems, analyzing advantages and limitations of it.*

Only row 2 in Table 1 considers P/T system in which the underlying net has one kind of limitations in its composition. What about other limitations in the arrangement?

Suggestion for research 6 *Study further the computational power of P/T systems whose relative arrangement of pairs of* fork *and* join *is limited.*

In particular one could focus on the limitations needed to generate semilinear sets.

The relevance of the following suggestion should be straightforward:

Suggestion for research 7 *Are* join *and* fork *the only building blocks that lead to the result in Table 1? Are there other building blocks leading to the same or different results?*

The overall aim of the approach considered in this paper is:

Suggestion for research 8 *Create a full hierarchies of accepting and generating computational processes in terms of sets of building blocks, compositions of elements in these sets and the functions W and K (present in the definition of P/T systems in [13, 5]).*

References

1. M. Cavaliere: Computing by observing: A brief survey. In A. Beckmann, C. Dimitracopoulos, B. Löwe, editors, *Logic and Theory of Algorithms, 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008, Proceedings*, volume 5028 of *Lecture Notes in Computer Science*. Springer, Berlin, 2008.
2. R. Freund, L. Kari, M. Oswald, P. Sosík: Computationally universal P systems without priorities: two catalysts are sufficient. *Theoretical Computer Science*, 330, 2 (2005), 251–266.
3. R. Freund, G. Lojka, M. Oswald, G. Păun, eds.: *Membrane Computing. 6th International Workshop, WMC 2005, Vienna, Austria, July 18-21, 2005, Revised Selected and Invited Papers*, volume 3850 of *Lecture Notes in Computer Science*. Springer, Berlin, 2006.
4. P. Frisco: P systems, Petri nets, and Program machines. In Freund et al. [3], 209–223.
5. P. Frisco: A hierarchy of computational processes. Technical report, Heriot-Watt University, 2008. HW-MACS-TR-0059 `http://www.macs.hw.ac.uk:8080/techreps/index.html`.
6. P. Frisco: *Computing with Cells. Advances in Membrane Computing.* Oxford University Press, 2009. to appear.
7. P. Frisco: On languages accepted by P/T systems composed by *join*. Technical report, Heriot-Watt University, 2009. HW-MACS-TR-0064 `http://www.macs.hw.ac.uk:8080/techreps/index.html`.
8. S.A. Greibach: Remarks on blind and partially blind one-way multicounter machines. *Theoretical Computer Science*, 7 (1978), 311–324.
9. O.H. Ibarra: On membrane hierarchy in P systems. *Theoretical Computer Science*, 334 (2005), 115–129.
10. O.H. Ibarra, H.-C. Yen: Deterministic catalytic systems are not universal. *Theoretical Computer Science*, 363, 2 (2006), 149–161.
11. R. Milner: *Communication and Concurrency.* Prentice-Hall, Englewood, N.J., 1989.
12. R. Milner: *Communicating and Mobile Systems: the π-calculus.* Cambridge University press, 1999.
13. W. Reisig, G. Rozenberg, eds.: *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer, Berlin, 1998.
14. G. Rozenberg, J. Engelfriet: *Elementary net systems*, pages 12–121. Volume 1491 of Reisig and Rozenberg [13], 1998.
15. G. Rozenberg, A. Salomaa: *Handbook of Formal Languages.* Springer, Berlin, 1996.

Modeling Reaction Kinetics in Low-dimensional Environments with Conformon P Systems: Comparison with Cellular Automata and New Rate Laws

Pierluigi Frisco[1], Ramon Grima[2]

[1] School of Mathematics and Computer Sciences
Heriot-Watt University, Edinburgh, UK
pier@macs.hw.ac.uk

[2] School of Biological Sciences
University of Edinburgh, UK
Ramon.grima@ed.ac.uk

Summary. Recently it has been shown that simulations of complex biological systems using conformon P systems and cellular automata do not necessarily give the same predictions. To further elucidate these differences we simulate a simple model of intracellular reactions involving a single bimolecular reaction occurring on a biological membrane using conformon P systems.

We find that the predictions broadly agree with results from both the theory of random walks in low-dimensional environments and with previously published simulations using cellular automata. Moreover, a re-analysis of the data enables us to deduce novel rate laws for the kinetics of reactions occurring on biological membranes.

1 Introduction

A recent publication [3] reported that simulations of HIV dynamics differ in their results according to the simulation platform used. In particular it is found that cellular automata (CA) models produce qualitatively correct dynamics only for a narrow range of rule probabilities and for particular initial conditions whereas conformon P (cP) models [2] derived from the CA model display significant robustness of qualitatively correct dynamics over a wide range of conditions.

Presently the reasons for these differences are not understood. The complexity of the system under study precludes a rigorous analysis of these discrepancies.

In this paper we consider a much simpler biological process at the base of an simpler model. For such model its rigorous analytical results are known for some cases.

The paper is divided as follows: in Section 2 we describe the biology behind the model and its implementation using CA and cP, in Section 3 we analyze the data generated by cP, compare with theory and CA results and also deduce some new biologically relevant kinetic laws and in Section 4 we draw some conclusion.

2 A Simple Biological Model: Implementation Using CA and cP

There is growing evidence of the importance of reaction kinetics for the structural organization of the intracellular environment, which is far from the homogeneous, well mixed solution typical of *in vitro* experiments. A high degree of molecular crowding as well as the presence of indigenous obstacles in cellular media have important consequences in the physico-chemistry of the cell. The consequences of this in some process are only now becoming to be more generally understood. One of these consequences is that it is not clear what are the rate laws governing reactions occurring *in vivo* [7]. To tackle this problem biochemists have been using various computational frameworks to extract rate laws or empirical reaction equations from direct numerical simulations. Among these approaches, simulations based on CA are the most popular (see, for example, [5]).

The biological process we considered in our investigations regards biochemical reactions occurring on cell membranes. It in known that about half of the proteins inside cells are membrane-associated [8] and thus biochemical reactions must necessarily function within the constraints imposed by the two-dimensional environment of the biological membrane. Prominent examples of such reactions are those involving enzymes called lipases which play key roles in fat metabolism and digestion and which occur on two-dimensional interfaces rather than in three-dimensional solution. The simplest model of such dimensionally-restricted reaction kinetics consists of two types of particles, denoted with A and B, which perform random walks on a two-dimensional plane and which upon encounter react with some probability and produce a single new inert particle C. This mathematical construct represents the physical process of the reaction of two molecules of two different types which normally perform Brownian motion (modelled by the random walks) and which react upon encounter to form some new product molecule [4]. Such elementary reactions form the backbone of all biochemical reaction networks, independent of their complexity and are particularly ideal for a comparison between CA and cP models because of the existence of rigorous analytical results from the theory of random walks in low-dimensions.

The biological process indicated above simplifies the biological membrane to a homogeneous quasi-two dimensional environment. In reality it is found that the heterogeneous micro-structure of the membrane significantly hinders the free diffusion of molecules on its surface. In particular it is known that transmembrane proteins (denoted with B in the above) impose relatively static barriers to the smaller and more mobile molecules (denoted with A in the above). This is due because transmembrane proteins are anchored to the cytoskeleton of the cell. These obstacles are incorporated in the models considered by us by making some parts of the plane inaccessible to particle motion.

A general CA model which describes both cases above (with and without obstacles to particle motion) has been described in [4]. The algorithm is the follows. Initially, particles of two different kinds A and B are uniformly distributed on a two-dimensional lattice with unit spacing and periodic boundary conditions (a torus). Particles A can move, while particles B are static. One A and one B particle can react when in the same location of the lattice and produce one particle C. Particles C are static and inert. Some of the tests considered another type of static and inert particle, an *obstacle*, uniformly distributed in the lattice in the initial configuration. At each time step, a particle of type A or B is randomly chosen and either moved or subject to a reaction according to the following:

the particle can move from the location in the lattice in which it is to a randomly chosen neighbor location only if the chosen neighbor location does not contain any other particle of the same kind or an obstacle;

if instead the chosen neighbor lattice location contains a particle of the other kind (that is, if A is subject to be moved, then B is the other kind; if B is subject to be moved, then A is the other kind), then the particle can react with the particle of the other kind with probability P. If this occurs both particles A and B are removed and a C particle is placed in the chosen neighbor location, otherwise nothing occurs.

It is important to note that the algorithm does not allow more than one particle of any type to be in the same location of the lattice, thus enforcing a hard-sphere molecular repulsion. The above two steps are repeated $n_{tot}(t)$ times, where $n_{tot}(t)$ is the current number of distinct particles on the lattice (excluding obstacles) at time t. After one such sequence the time is incremented by one. The simulations are performed with two different lattice types: square (von Neumann) and triangular neighborhoods.

Models of cP systems have been derived by the just described CA model. Particles of type A and B have been modeled with $[A, 1]$ and $[B, 1]$ conformons, respectively. Their eventual interaction (with probability P) creates $[B, 2]$ conformons representing the C particles. A lattice location of the CA has been modeled with a (membrane) compartment in the cP model. The presence of obstacles has been modeled with compartments in the lattice having no incoming edge (in this way no conformon could move in these compartments). The simulations have been performed using the cP simulator available from [9] modifying it in a way that interaction rules have priority on passage rules. Moreover, *ad hoc* programs to create the lattices and to analyze the data have been also used. These programs can be requested to the authors. The cP models and the simulations are such that more than one particle can be at the same time in one compartment.

3 Data Analysis

Data produced by the simulations consisted of the number of A and B particles as a function of time. For each set of parameter values, ten independent simulations were run on 300×300 lattices until the time (denoted by T) at which there remained only 1% of the minority particle species (that is, the one whose initial concentration is the smallest, which in our case are particles of type A [4]). The majority species is static, the minority species is mobile and their probability of interaction is P. We run the tests with two different values of P: 1 and 0.1. We consider two cases: presence and absence of *obstacles* uniformly distributed throughout the lattice so to occupy at most 0.4 of the available lattice locations (denoted by $[O] = 0.4$). The data from these ten copies were then averaged to reduce the inherent noise, yielding an array of values $[A(t)], [B(t)], t, t = 0...T$.

The simulation data for the absence of obstacles case can be directly compared to rigorous theoretical results from the theory of random walks [1]. Here it is shown that in a two-dimensional space where the majority particle species is immobile (B) and minority species is mobile (A) and the reaction probability is 1, then for sufficiently long time runs, the rate of change of the concentrations of mobile particles is described by the effective ordinary differential equation:

$$d[A(t)]/dt \sim -t^{-1/2}[A(t)][B(t)], \tag{1}$$

where $[A(t)]$ denotes the total number of particles of type A at time t divided by the total number of lattice locations defining the two-dimensional space. Thus the most basic test of our cP simulations is to use the data obtained for the case of no obstacles and $P = 1$ to extract the time exponent in the above equation.

The method used to obtain this exponent is the one reported in [4] where it is shown that for general differential equations of the type $d[A(t)]/dt \sim -t^{-(1-p)}[A(t)][B(t)]$, the exponent p is equal to the gradient of the graph of $G = Log[-Log(B_0[A(t)]/(A_0(B_0 - A_0 + [A(t)])))]$ versus $Log(t)$. Figure 1, bottom curve, shows the variation of the slope (that is, p) with time for the just indicated cP simulations. It is found that $p = 0.6$. This implies that the ordinary differential equation satisfied by the cP simulation data is $d[A(t)]/dt \sim -t^{-0.4}[A(t)][B(t)]$. This result is fairly close to the rigorous theoretical value given above and also agrees with previous CA simulations giving $p = 0.5$.

Figure 1, above curve, shows the results of the simulations for $P = 0.1$. No rigorous theory exists for this case, but CA simulations [4] give $p \sim 0.85$ whereas with cP simulations we obtain $p \sim 0.92$. Hence for the case of no obstacles, for both high and low values of reaction probability P, the results of CA and cP are in good quantitative agreement, though there is a consistent tendency of the exponent for cP to be slightly larger than that of CA. The latter discrepancy could be due to the fact that CA simulations impose the condition that only one particle is allowed at a site whereas cP simulations make no such assumption. The lack of such an assumption would necessarily imply a larger amount of "particle mixing" inside each spatial element of cP simulations which from physical considerations [4] would necessitate the exponent p to be closer to one, as observed.

To test this hypothesis we developed a cP model in which at most one particle per compartment is allowed. Figure 2 shows the curve comparing the data obtained by the cP models in which more than one particle and at most one particle per

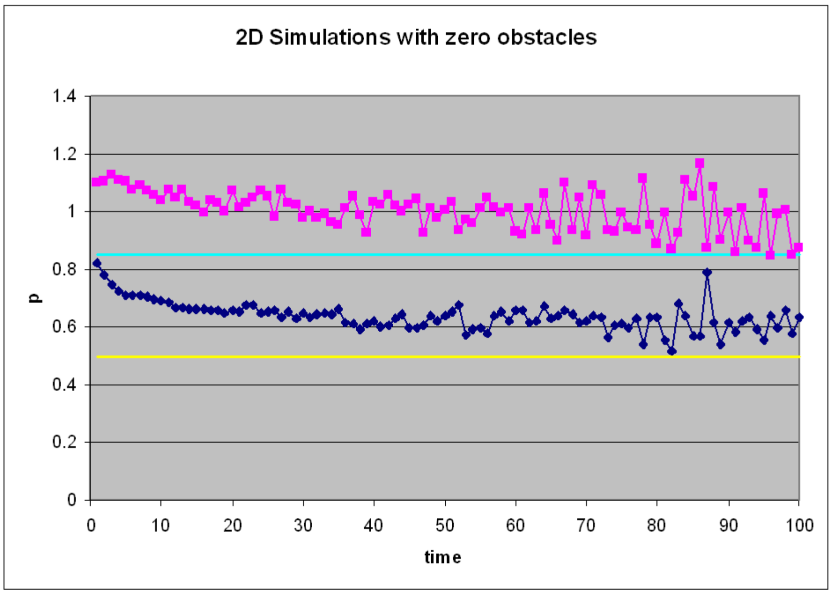


Fig. 1. Variation of the slope with no obstacles in the cP model

compartment is allowed. No discernible differences are observed between the two models. This implies that either the effect occurs only in lower dimensions or the possibility to have more than one particle in the same compartment is not the reason for the small discrepancies between CA and cP simulations.

There are no rigorous analytical results for the case in which obstacles are present. Anyhow, it has been traditionally assumed that the dynamics in this case would be captured by an effective ordinary differential equation as (1), but with a time exponent p which varies somewhere between 0 and 1. This is often referred to as fractal kinetics [5, 7]. However, in [4] it is shown that this is not the case. It is found from CA simulations that the slope is not constant but varies considerably with time and apparently does not approach a constant value in the limit of long time runs. Our cP simulations also confirm this result (Figure 3), once again showing no evidence of a discrepancy between CA and cP simulations.

In the present paper we go one step beyond the work reported in [4] and, for the case in which obstacles are present, we find a new effective ordinary equation which captures the dynamics of the reaction. It can be shown [4] that the solution of an ordinary differential equation of the type $d[A(t)]/dt \sim -k(t)[A(t)][B(t)]$ for long time runs is of the form:

$$[A(t)] \propto \exp\Big[-(B_0 - A_0)\int_0^t k(s)ds\Big]. \tag{2}$$

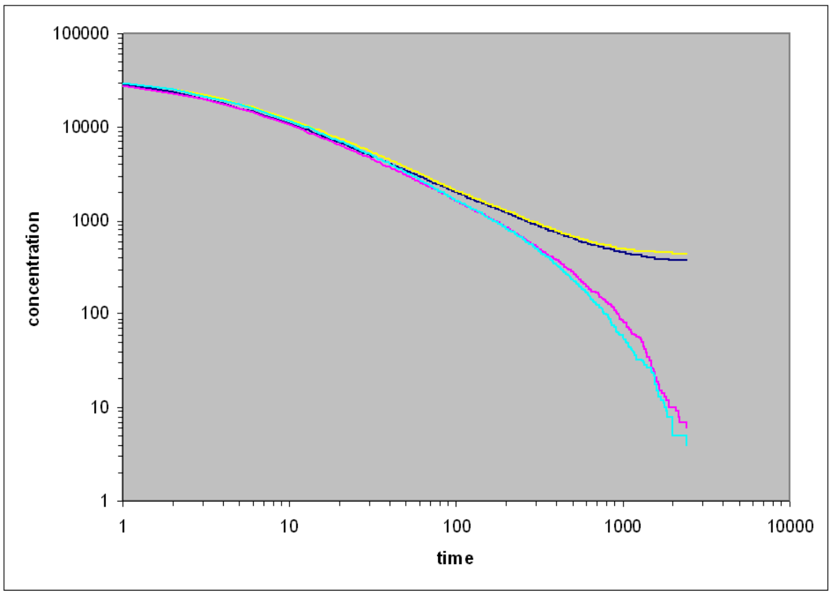


Fig. 2. Curves comparing results from cP simulations. Blue and pink curves for A and B when at most 1 particle per compartment is allowed. Yellow and cyan curves for A and B when more than one particle per compartment is allowed. The graph shows that the variation of particle concentration with time is independent of the one-particle constraint.

It is also known that $k(t) = dS/dt$ where S is the mean number of distinct lattice locations visited by a particle moving in a random walk [5]. It is found that $S \sim t^{d_s/2}$ for long time runs in a fractal space of spectral dimension d_s. This would imply (for long time runs) the dynamics to follow an effective equation of the form $d[A(t)]/dt \sim -t^{-(1-d_s/2)}[A(t)][B(t)]$ which implies a constant time exponent p. However note that to arrive at this conclusion one implicitly assumes that the long time regime is being observed. Actuality one may only observe the early and intermediate time regimes since the simulation halts after 99% of the particle A has been consumed.

Inspired by the theoretical results reported in [4], we surmise that the intermediate time scaling for S would be of the general form: $S \sim t^{\alpha} Ln(1/Ln(t^{\alpha}))$ where the exponent α is introduced to take into account the heterogeneity of space imposed by the presence of obstacles. Interestingly, it is found that the cP data is in good agreement with this conjectured law, see Figure 4.

Thus our simulations and data analysis suggest a new kinetic equation for describing bimolecular reactions in obstacle-ridden low dimensional media, namely $d[A(t)]/dt \sim -k(t)[A(t)][B(t)]$ with $k(t) = \partial/\partial t t^{\alpha} Ln(1/Ln(t^{\alpha}))$ instead of the customary $k(t) = t^{\alpha}$.

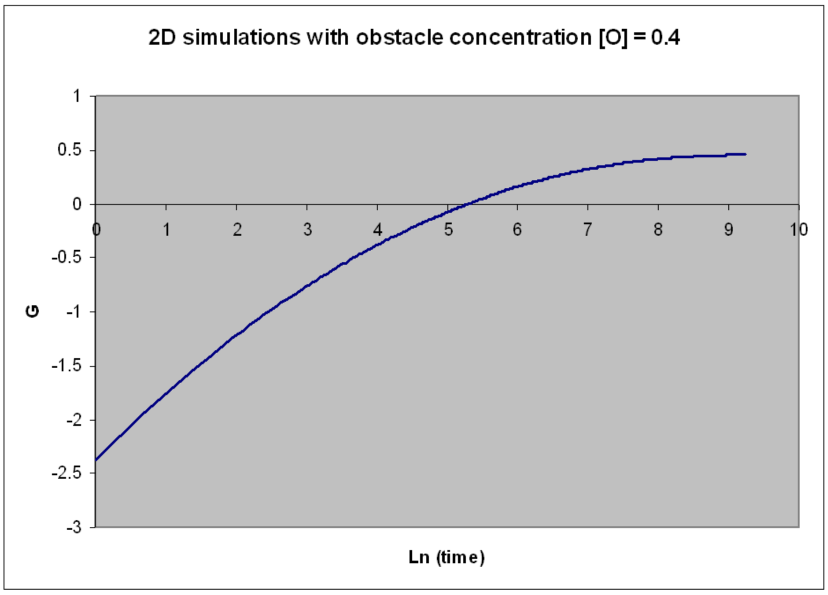


Fig. 3. *G* value for the cP tests with more than one particle per compartment and no obstacles

4 Conclusion

In this paper we report a preliminary study aiming to understand what kind of biological processes are better fit to be modeled with CA or with cP. In particular, we focused on the possibility offered by cP to model the presence of more than one particle in a compartment. From the tests we run we conclude that this possibility does not always make a difference in the obtained results.

Some differences between the results obtained by similar CA and cP models occurs only if obstacles, locations in the lattice limiting the mobility of the particles, are present. Anyhow, the found differences are not yet sufficient to draw general conclusions.

One line of further research is to compare the fluctuations from the average values in CA and cP simulations.

Acknowledgements: R. Grima gratefully acknowledges support from SULSA (Scottish Universities Life Sciences Alliance).

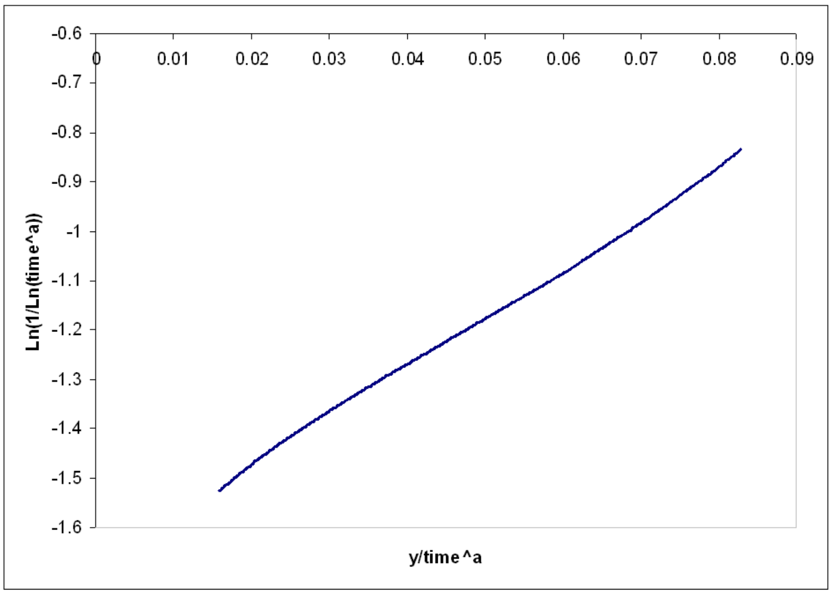


Fig. 4. Results for the cP tests with more than one particle per compartment and obstacles

References

1. M. Bramson, J.L. Lebowitz: Asymptotic behavior of densities in diffusion-dominated annihilation reactions. *Phys. Rev. Lett.*, 61 (1988), 2397.
2. P. Frisco: The conformon-P system: A molecular and cell biology-inspired computability model. *Theoretical Computer Science*, 312 (2004), 295.
3. P. Frisco, D.W. Corne: Dynamics of HIV infection studied with cellular automata and conformon-P systems. *Biosystems*, 91 (2008), 531.
4. R. Grima, S. Schnell: A systematic investigation of the rate laws valid in intracellular environments. *Biophys. Chem.*, 124 (2006), 1.
5. R. Kopelman: Rate-processes on fractals: theory, simulations and experiments. *J. Stat. Phys.*, 42 (1986), 185.
6. H. Larralde, P. Trunfio, S. Havlin, H.E. Stanley, G.H. Weiss: Number of distinct sites visited by N random walkers. *Phys. Rev. A*, 45 (1992), 7128 .
7. S. Schnell, T.E. Turner: Reaction kinetics in intracellular environments with macromolecular crowding: simulations and rate laws. *Prog. Biophys. Mol. Biol.*, 85 (2004), 235.
8. P. Reis, K. Holmberg, H. Watzke, M.E. Leser, R. Miller: Lipases at interfaces: A review. *Adv. Coll. Int. Sci.*, 2008, doi:10.1016/j.cis.2008.06.001.
9. The P Systems Webpage: `http://ppage.psystems.eu/`.

P-Lingua 2.0: New Features and First Applications

Manuel García-Quismondo, Rosa Gutiérrez-Escudero, Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
mangarfer2@alum.us.es, {rgutierrez,perezh,marper}@us.es

Summary. 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 two new applications based on P-Lingua 2.0: a tool for describing and simulating ecosystems and a framework (currently under development) for P systems design.

1 Introduction

Membrane computing (or cellular computing) is a branch of Natural Computing that was introduced by Gh. Păun [14]. The main idea is to consider biochemical processes taking place inside living cells from a computational point of view, in a way that provides a new nondeterministic model of computation.

The initial definition of this computing paradigm is very flexible, and many different models have been defined and investigated in the area: P systems with symport/antiport rules, with active membranes, with catalysts, with promoters/inhibitors, etc. There were some attempts to establish a common formalization covering most of the existing models (see e.g. [5]), but the membrane computing community is still using specific syntax and semantics depending on the model they work with.

1.1 Introduction to P-Lingua

When designing software simulators for membrane computing, one has to precisely define the P system that is to be simulated. This task is hard if we need to handle families of P systems where the set of rules, the alphabet, the initial contents and even the membrane structure depend on the value assigned to some initial parameters. Current software applications are usually focused on, and adapted for, particular cases, making it difficult to get interoperability.

In [3] it was introduced a programming language, called P-Lingua, whose programmes define active membrane P systems with division rules in a parametric and modular way. In this sense, it is possible to define a family of P systems with the use of parameters. After assigning values to the initial parameters, the compilation tool generates an XML document associated with the corresponding P system from the family, and furthermore it checks possible programming errors (both lexical/syntactical and semantical). Such documents can be integrated into other applications, thus guaranteeing interoperability by using the same P system definition in different software environments.

P-Lingua 2.0 is able to define P systems within different models, at this stage: active membrane P systems with membrane division rules or membrane creation rules, transition P systems, symport/antiport P systems, stochastic P systems and probabilistic P systems. Each model follows semantics restrictions, which define several constraints which rules in the model's P systems should follow (number of objects on each side, if membrane creation and/or membrane division are allowed, and so on) and the way rules are applied on configurations to evolve to other ones. Additional models can be added to the P-Lingua framework, but it is important to say that P-Lingua 2.0 supports only P systems whose configurations have a cell-like structure.

P-Lingua 2.0 defines several algorithms (from now on, simulators) to simulate P system configuration computations for each supported model, so every computation on a configuration whose P system belongs to a model can be performed by any simulator defined for the model.

P-Lingua 2.0 also supports different formats. For the purpose of this paper, a format is a way of representing P systems on a file. P-Lingua 2.0 supported formats range from XML and P-Lingua language to binary, and it provides a standard mechanism to add new formats, if needed.

1.2 An input standard for simulators

Each model displays characteristic semantic constraints entailing the rules applied, such as number of objects specified on the left-hand side, membrane creation, polarization, and so on. Hence, the need for 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, so that it is able to carry out computations. Nevertheless, this approach involves defining new input formats for each simulator, so designing simulators would take a great amount of effort as a new input format needs to be defined for each new developed simulator. 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. Besides, the former point of view would spare researchers from analyzing different models and P systems in order to extract common patterns out of them. On the other hand, although the latter approach involves analyzing different P systems and models to develop a standard format, it allows to use common simulators for different P systems. In this way, 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 and simulate computations by using simulators able to process it. Moreover, researches would not have to devise a new input format every time they specify a P system; they would use the standard format instead. In addition, researchers 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 on P-Lingua 2.0.

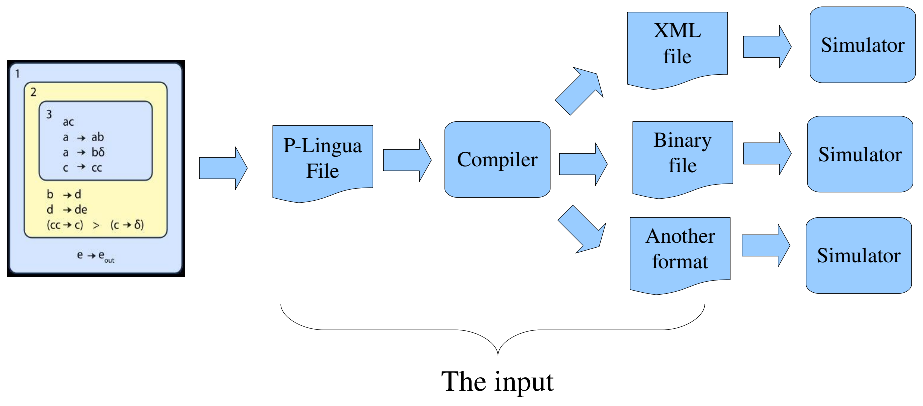


Fig. 1. The standard input format

2 Models

2.1 Contemplating new models

As mentioned, P-Lingua 1.0 provided support for active membrane P systems with division rules. However, as P-Lingua is intended to become a standard for P systems definition, it should contemplate other models. The supported models so far are enumerated below, but a standard mechanism for defining new models and simulators for each model has been defined on P-Lingua 2.0, easing those tasks. This mechanism has been used on all the existent models and simulators.

2.2 Transition P system model

The basic P systems were introduced in [14].

A *transition P system* of degree $q \geq 1$ is a tuple of the form

$$\Pi = (\Gamma, L, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, (R_1, \rho_1), \ldots, (R_q, \rho_q), i_o)$$

where:

- Γ is an alphabet whose elements are called *objects*.
- L is a finite set of labels.
- μ is a membrane structure consisting of q membranes with the membranes (and hence the regions, the space between a membrane and the immediately inner membranes, if any) injectively labeled with elements of L; as usual, we represent the membrane structures by strings of matching labeled parentheses.
- $\mathcal{M}_i$, $1 \leq i \leq q$, are strings which represent multisets over Γ associated with the q membranes of μ.
- R_i, $1 \leq i \leq q$, are finite sets of *evolution rules* over Γ, associated with the membranes of μ. An evolution rule is of the form $u \to v$, where u is a string over Γ and $v = v'$ or $v = v'\delta$, being v' a string over $\Gamma \times (\{here, out\} \cup \{in_j : 1 \leq j \leq q\})$.
- ρ_i, $1 \leq i \leq q$, are strict partial orders over R_i.
- i_o, $1 \leq i_o \leq q$, is the label of an elementary membrane (the *output membrane*).

The objects to evolve in a step and the rules by which they evolve are chosen in a non–deterministic manner, but in such a way that in each region we have a maximally parallel application of rules. This means that we assign objects to rules, non–deterministically choosing the rules and the objects assigned to each rule, but in such a way that after this assignation no further rule can be applied to the remaining objects.

2.3 Symport/antiport P system model

Symport/antiport rules were incorporated in the framework of P systems in [13].

A *P system with symport/antiport rules* of degree $q \geq 1$ is a tuple of the form

$$\Pi = (\Gamma, L, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, E, R_1, \ldots, R_q, i_o)$$

where:

- Γ is the alphabet of objects,
- L is the finite set of labels for membranes (in general, one uses natural numbers as labels), μ is the membrane structure (of degree $q \geq 1$, with the membranes labeled in a one-to-one manner with elements of L,
- $\mathcal{M}_1, \ldots, \mathcal{M}_q$ are strings over Γ representing the multisets of objects present in the q compartments of μ in the initial configuration of the system.
- $E \subseteq \Gamma$ is the set of objects supposed to appear in the environment in arbitrarily many copies.

- R_i, $1 \leq i \leq q$, are finite sets of rules associated with the q membranes of μ. The rules from R can be of two types (by Γ^+ we denote the set of all non-empty strings over Γ, with λ denoting the empty string):
  - *Symport rules*, of the form (x, in) or (x, out), where $x \in \Gamma^+$. When using such a rule, the objects specified by x enter or exit, respectively, the membrane with which the rule is associated. In this way, objects are sent to or imported from the surrounding region – which is the environment in the case of the skin membrane.
  - *Antiport rules*, of the form $(x, out; y, in)$, where $x, y \in \Gamma^+$. When using such a rule for a membrane i, the objects specified by x exit the membrane and those specified by y enter from the region surrounding membrane i; this is the environment in the case of the skin membrane.
- $i_o \in L$ is the label of a membrane of μ, which indicates the *output* region of the system.

The rules are used in the non-deterministic maximally parallel manner, standard in membrane computing.

2.4 Active membranes P system model

2.4.1 With membrane division rules

P systems with membrane division were introduced in [15], and in this model the number of membranes can increase exponentially in polynomial time. Next, we define P systems with active membranes using 2-division for elementary membranes, with polarizations, but without cooperation and without priorities (and without permitting the change of membrane labels by means of any rule).

A *P system with active membranes* using 2-division for elementary membranes of degree $q \geq 1$ is a tuple $\Pi = (\Gamma, L, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, R, i_o)$, where:

- Γ is an alphabet of symbol-objects.
- L is a finite set of labels for membranes.
- μ is a membrane structure, of m membranes, labeled (not necessarily in a one-to-one manner) with elements of L.
- $\mathcal{M}_1, \ldots, \mathcal{M}_q$ are strings over Γ, describing the initial multisets of objects placed in the m regions of μ.
- R is a finite set of rules, of the following forms:
  (a) $[\, a \rightarrow \omega \,]_h^{\alpha}$ for $h \in L$,$\alpha \in \{+, -, 0\}$, $a \in \Gamma$, $\omega \in \Gamma^*$: This is an object evolution rule, associated with a membrane labeled with h and depending on the polarization of that membrane, but not directly involving the membrane.
  (b) $a\,[\ ]_h^{\alpha_1} \rightarrow [\, b \,]_h^{\alpha_2}$ for $h \in L$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Gamma$: An object from the region immediately outside a membrane labeled with h is introduced in this membrane, possibly transformed into another object, and, simultaneously, the polarization of the membrane can be changed.

(c) $[\, a\,]_h^{\alpha_1} \to b\, [\ ]_h^{\alpha_2}$ for $h \in L$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $a, b \in \Gamma$: An object is sent out from membrane labeled with h to the region immediately outside, possibly transformed into another object, and, simultaneously, the polarity of the membrane can be changed.

(d) $[\, a\,]_h^{\alpha} \to b$ for $h \in L$, $\alpha \in \{+, -, 0\}$, $a, b \in \Gamma$: A membrane labeled with h is dissolved in reaction with an object. The skin is never dissolved.

(e) $[\, a\,]_h^{\alpha_1} \to [\, b\,]_h^{\alpha_2}\ [\, c\,]_h^{\alpha_3}$ for $h \in L$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $a, b, c \in \Gamma$: An elementary membrane can be divided into two membranes with the same label, possibly transforming some objects and their polarities.

- $i_o \in L$ is the label of a membrane of μ, which indicates the *output* region of the system.

These rules are applied according to the following principles:

- All the rules are applied in parallel and in a maximal manner. In one step, one object of a membrane can be used by only one rule (chosen in a non-deterministic way), but any object which can evolve by one rule of any form, must do it.
- If a membrane is dissolved, its content (multiset and internal membranes) is left free in the surrounding region.
- If at the same time a membrane labeled by h is divided by a rule of type (e) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that the evolution rules of type (a) are used, and before division is produced. Of course, this process takes only one step.
- The rules associated with membranes labeled by h are used for all copies of this membrane. At one step, a membrane can be the subject of *only one* rule of types (b)-(e).

2.4.2 With membrane creation rules

Membrane creation rules were first considered in [9, 10].

A *P system with membrane creation* of degree $q \geq 1$ is a tuple of the form

$$\Pi = (\Gamma, L, \mu, \mathcal{M}_1, \ldots, \mathcal{M}_q, R, i_o)$$

where:

- Γ is the alphabet of objects.
- L is a finite set of labels for membranes.
- μ is a membrane structure consisting of q membranes labeled (not necessarily in a one-to-one manner) with elements of L.
- $\mathcal{M}_1, \ldots, \mathcal{M}_q$ are strings over Γ, describing the initial multisets of objects placed in the q regions of μ.
- R is a finite set of rules of the following forms:

(a) $[a \to v]_h$ where $h \in L$, $a \in \Gamma$, and v is a string over Γ describing a multiset of objects. These are *object evolution rules* associated with membranes and depending only on the label of the membrane.
(b) $a[\,]_h \to [b]_h$ where $h \in L$, $a, b \in \Gamma$. These are *send-in communication rules.* An object is introduced in the membrane possibly modified.
(c) $[a]_h \to [\,]_h\, b$ where $h \in L$, $a, b \in \Gamma$. These are *send-out communication rules.* An object is sent out of the membrane possibly modified.
(d) $[a]_h \to b$ where $h \in L$, $a, b \in \Gamma$. These are *dissolution rules.* In reaction with an object, a membrane is dissolved, while the object specified in the rule can be modified.
(e) $[a \to [v]_{h_2}]_{h_1}$ where $h_1, h_2 \in L$, $a \in \Gamma$, and v is a string over Γ describing a multiset of objects. These are *creation rules.* In reaction with an object, a new membrane is created. This new membrane is placed inside of the membrane of the object which triggers the rule and has associated an initial multiset and a label.

- $i_o \in L$ is the label of a membrane of μ, which indicates the *output* region of the system.

Rules are applied according to the following principles:

- Rules from (a) to (d) are used as usual in the framework of membrane computing, that is, in a maximally parallel way. In one step, each object in a membrane can only be used for applying one rule (non-deterministically chosen when there are several possibilities), but any object which can evolve by a rule of any form must do it (with the restrictions below indicated).
- Rules of type (e) are used also in a maximally parallel way. Each object a in a membrane labeled with h_1 produces a new membrane with label h_2 placing in it the multiset of objects described by the string v.
- If a membrane is dissolved, its content (multiset and interior membranes) becomes part of the immediately external one. The skin membrane is never dissolved.
- All the elements which are not involved in any of the operations to be applied remain unchanged.
- The rules associated with the label h are used for all membranes with this label, independently of whether or not the membrane is an initial one or it was obtained by creation.
- Several rules can be applied to different objects in the same membrane simultaneously. The exception are the rules of type (d) since a membrane can be dissolved only once.

2.5 Probabilistic P system model

A probabilistic approach in the framework of P systems was first considered by A. Obtulowicz in [12].

A *probabilistic P system* of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, \mu, \mathcal{M}_1, \dots, \mathcal{M}_q, R, \{c_r\}_{r \in R}, i_o)$$

where:

- Γ is the alphabet (finite and nonempty) of objects (the working alphabet).
- μ is a membrane structure, consisting of q membranes, labeled $1, 2, \dots, q$. The skin membrane is labeled by 0. We also associate electrical charges with membranes from the set $\{0, +, -\}$, neutral and positive.
- $\mathcal{M}_1, \dots, \mathcal{M}_q$ are strings over Γ, describing the multisets of objects initially placed in the q regions of μ.
- R is a finite set of evolution rules. An evolution rule associated with the membrane labeled by i is of the form $r: \ u[\ v\ ]_i^{\alpha} \xrightarrow{c_r} u'[\ v'\ ]_i^{\beta}$, where u, v, u', v' are a multiset over Γ, $\alpha, \beta \in \{0, +, -\}$ and c_r is a real number between 0 and 1 associated with the rule such that:
  – for each $u, v \in M(\Gamma)$, $h \in H$ and $\alpha \in \{0, +\}$, if $r_1, \dots, r_t$ are the rules whose left–hand side is $u[\ v\ ]_h^{\alpha}$, then $\sum_{j=1}^{t} c_{r_j} = 1$
- $i_o \in L$ is the label of a membrane of μ, which indicates the *output* region of the system.

We assume that a global clock exists, marking the time for the whole system (for all compartments of the system); that is, all membranes and the application of all rules are synchronized.

The q-tuple of multisets of objects present at any moment in the q *regions* of the system constitutes the *configuration* of the system at that moment. The tuple $(\mathcal{M}_1, \dots, \mathcal{M}_q)$ is the initial configuration of the system.

We can pass from one configuration to another one by using the rules from R as follows: at each transition step, the rules to be applied are selected according to the probabilities assigned to them, all applicable rules are simultaneously applied, and all occurrences of the left–hand side of the rules are consumed, as usual. Rules with the same left–hand side and whose right–hand side has the same polarization can be applied simultaneously.

2.6 Stochastic P system model

The original motivation of P systems was not to provide a comprehensive and accurate model of the living cell, but to imitate the computational nature of operations that take place in cell membranes. Most P system models have been proved to be Turing complete and computationally efficient, in the sense that they can solve computationally hard problems in polynomial time, by trading time for space. Most research in P systems focus on complexity classes and computational power.

However, P systems have been used recently to model biological phenomena very successfully. Models of oscillatory systems [4], signal transduction [18], gene regulation control [16], quorum sensing [17] and metapopulations [19] have been presented.

We introduce in this section the specification of stochastic P systems, that constitute the framework for modeling biological phenomena.

A *stochastic P system* of degree $q \geq 1$ is a tuple

$$\Pi = (\Gamma, L, \mu, \mathcal{M}_1, \dots, \mathcal{M}_q, R_{l_1}, \dots, R_{l_m})$$

where:

- Γ is a finite alphabet of symbols representing objects.
- $L = \{l_1, \dots, l_m\}$ is a finite alphabet of symbols representing labels for the membranes.
- μ is a membrane structure containing $q \geq 1$ membranes identified in a one to one manner with values in $\{1, \dots, q\}$ and labeled with elements from L.
- $M_i = (l_i, w_i, s_i)$, for each $1 \leq i \leq q$, initial configuration of the membrane i, $l_i \in L$ is the label, $w_i \in \Gamma^*$ is a finite multiset of objects and s_i is a finite set of strings over Γ.
- $R_{l_t} = \{r_1^{l_t}, \dots, r_{k_{l_t}}^{l_t}\}$, for each $1 \leq t \leq m$, is a finite set of rewriting rules associated with membranes of label $l_t \in L$. Rules are of one of the following two forms:
  - Multiset rewriting rules:

    $$r_j^{l_t} : u[w]_l \xrightarrow{c_j^{l_t}} u'[w']_l$$

    with $u, w, u', w' \in \Gamma^*$ some finite multisets of objects and l a label from L. A multiset of objects, u is represented as $u = a_1 + \cdots + a_m$, with $a_1, \dots, a_m \in \Gamma$. The empty multiset will be denoted by λ and we will write o^n instead of $\overbrace{o + \cdots + o}^{n}$. The multiset u placed outside of the membrane labeled with l and the multiset w placed inside of that membrane are simultaneously replaced with a multiset u' and w' respectively.
  - String rewriting rules:

    $$r_j^{l_t} : [u_1 + s_1; \dots; u_p + s_p]_l \xrightarrow{c_j^{l_t}} [u'_1 + s'_{1,1} + \cdots + s'_{1,i_1}; \dots; u'_p + s'_{p,1} + \cdots + s'_{p,i_p}]$$

    A string s is represented as $s = \langle o_1.o_2. \cdots .o_j \rangle$, where $o_1, o_2, \dots, o_j \in \Gamma$. Each multiset of objects u_j and string s_j, $1 \leq j \leq p$, are replaced by a multiset of objects u'_j and strings $s'_{j,1}, \dots, s_{j,i_j}$.

  A constant $c_j^{l_t}$ is associated with each rule and will be referred to as *stochastic constant* and is needed to calculate the propensity of the rule according to the current context of the membrane to which this rule corresponds.

Rules in stochastic P systems model biochemical reactions. The *propensity* a_j of a reaction R_j is defined so that $a_j dt$ represents the probability that R_j will occur in the infinitesimal time interval $[t, t + dt]$ [7].

Applications of the rules and the semantics of stochastic P systems can vary, depending on which algorithm is used to simulate the model. At the present stage, two algorithms have been implemented and integrated as simulators within the pLinguaCore library. They will be discussed in Section 3.2.

3 Simulators

3.1 Contemplating new simulators

In P-Lingua 1.0, only one simulator was supported, since there was only one model to simulate. However, as new models have been included, new simulators have been developed, providing at least one simulator for each supported model. Furthermore, P-Lingua 2.0 provides translation and error detection services for the supported models.

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 counts on several simulation algorithms to choose, but P-Lingua 2.0 provides a mechanism to include new simulators for defined models.

3.2 Simulators for stochastic P systems

In the original approach to membrane computing 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 [14]). 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 modeled by rules, occur at a specific rate (determined by the propensity of the rule), therefore they can not be selected in an arbitrary and non-deterministic way. Second, in the classical approach all time step 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 [7]. As mentioned in Section 2.6, a constant $c_j^{l_t}$ is associated to each rule. This provides P systems with a stochastic extension. The constant $c_j^{l_t}$ depends on the physical properties of the molecules involved in the reaction modeled by the rule and other physical parameters of the system and it represents the probability per time unit that 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. The plugin-oriented architecture of P-Lingua allows easily to encode new simulators.

3.2.1 Multicompartmental Gillespie algorithm

The Gillespie [7] 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 [20] and it can be applied in the different regions defined by the compartmentalized 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 Multicompartmental 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.

3.2.2 Multicompartmental Next Reaction method

The Gillespie Algorithm is an exact numerical simulation method appropriate for systems with a small number of reactions, since it takes time proportional to the number of reactions (i.e., the number of rules). An exact algorithm which is also efficient is presented in [6], the Next Reaction Method. It uses only a single random number per simulation event (instead of two) and takes time proportional to the logarithm of the number of reactions. We have adapted this algorithm to make it compartmental.

The idea of this method is to be extremely sensitive in recalculating a_j and t_i, recalculate them only if they change. In order to do that, a data structure called *dependency graph* [6] 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 labeled 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 and $t = l$ 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 is executed and $t = l$ if $b \in v \vee b \in v'$ and $t = l'$ if $b \in u \vee b \in u'\}$.
  Generally, Affects$(r) = \{(b, l) \mid b \in v \vee b \in v'\} \cup \{(b, l') \mid b \in u \vee b \in u'\}$.

Definition 1. *Given a set of rules $R = R_{l_1} \cup \cdots \cup R_{l_m}$, the dependency graph is a directed graph $G = (V, E)$, with vertex set $V = R$ and edge set $E = \{(v_i, v_j) \mid$ Affects$(v_i) \cap$ 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 [6], 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 Multicompartmental 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 Multicompartmental Gillespie Algorithm and Multicompartmental Next Reaction Method are the core of the Direct Stochastic Simulator and Efficient Stochastic 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>`. See Section 4.2 for more details about the syntax.

3.3 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 said 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 give configuration, we proceed as follows:
   – It is computed the greatest number N so that u^N appears in the father 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 Formats

4.1 Contemplating new formats

As well as models and simulators, new formats have been included in P-Lingua 2.0. P-Lingua 1.0 provided a P-Lingua language format and an XML file format [3]. Those formats have been upgraded to allow representation of P systems which have cell-like structure, so any P system which corresponds to an existing model can be expressed via XML format or the P-Lingua 2.0 language. To accept the new models, P-Lingua's general syntax has changed, but it also supports backwards compatibility, so any P system accepted by P-Lingua 1.0 is recognized by P-Lingua 2.0, whether it is expressed in XML format or the P-Lingua language. A new format has been included as well: the binary format. This format is the input format for the incoming parallel simulator [11], so it is possible to define a P system in P-Lingua language and compile it to binary 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 pLinguaCore, a JAVA [23] library described in Section 6 of this paper. An output format is a file format which, if a P system is specified on a file by following that format, tat file can be generated by pLinguaCore. These concepts are similar to the source code and object code concepts [3].

For P-Lingua 2.0, P-Lingua language format 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 language files can be processed by pLinguaCore, binary files can be generated by pLinguaCore and XML files can be both processed and generated by pLinguaCore.

4.2 P-Lingua format

In the version of P-Lingua presented in [3] only P systems with active membranes and division rules were considered and therefore, possible to be defined in the P-Lingua language. New models have been added and consequently the syntax has been modified and extended, in order to support them. The syntax of the P-Lingua 2.0 language is defined as follows.

4.2.1 Valid identifiers

We say that a sequence of characters forms a `valid identifier` if it does not begin with a numeric character and it is composed by characters from the following:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 _
```

Valid identifiers are widely used in the language: to define module names, parameters, indexes, membrane labels, alphabet objects and strings.

The following text strings are reserved words in the language: `def, call, @mu, @ms, @model, @lambda, @d, let, @inf, @debug, main, -->, #` and they cannot be used as valid identifiers.

4.2.2 Variables

Four kind of variables are permitted in P-Lingua:

- `Global variables`
- `Local variables`
- `indexes`
- `Parameters`

Variables are used to store numeric values and their names are valid identifiers. We use 64 bits (signed) in double precision.

Global variables definition

Global variables must be declared out of any program module and they can be accesed from all of the program modules (see 4.2.10). The name of a global variable `global_variable_name` must be a valid identifier. The syntax to define a global variable is the following:

```
global_variable_name = numeric_expression;
```

Local variables definition

Local variables can only be accessed from the module in which they were declared and they must only be defined inside module definitions. The name of a local variable `local_variable_name` must be a valid identifier. The syntax to define a local variable is the following:

```
let local_variable_name = numeric_expression;
```

Indexes and parameters can be consider local variables used in 4.2.16 and 4.2.10 respectively.

4.2.3 Identifiers for electrical charges

In P-Lingua, we can consider electrical charges by using the `+` and `-` symbols for positive and negative charges respectively, and no one for neutral charge. It is worth mentioning that polarizationless P systems are included.

4.2.4 Membrane labels

There are three ways of writing membrane labels in P-Lingua: the first one is just a natural number; the second one is to denote the label as a valid identifier and the third one is by numeric expressions that represent natural numbers between brackets.

4.2.5 Numeric expressions

Numeric expressions can be written by using `*` (multiplication), `/` (division), `%` (module), `+` (addition), `-` (subtraction) operators with integer or real numbers and/or variables, along with the use of parentheses. It is possible to write numbers by using exponential notation. For example, $3 * 10^{-5}$ is written `3e-5`.

4.2.6 Objects

The objects of the alphabet of a P system are written using valid identifiers, and the inclusion of sub-indexes is permitted. For example, $x_{i,2n+1}$ and Yes are written as `x{i,2*n+1}` and `Yes` respectively.

The multiplicity of an object is represented by using the `*` operator. For example, x_i^{2n+1} is written as `x{i}*(2*n+1)`.

4.2.7 Strings

Strings are enclosed between `<` and `>` and made by concatenating valid identifiers with the character `.`, that is `<identifier1. ... .identifierN>`. For example, `<cap.RNAP.op>`.

4.2.8 Substrings

Substrings are used in string rewriting rules and the syntax is similar to strings, but it is possible to use the character `?` to represent any arbitrary sequence of valid identifiers concatenated by `.`. The empty sequence is included. For example, `<cap.?.NAP.op>` is a substring of the string `<cap.op.op.op.NAP.op>` and of the string `<cap.NAP.op>`.

4.2.9 Model specification

As this programming language supports more than one model, it is necessary to specify in the beginning of the file which is the model of the P system defined. Not each type of rule is allowed in every model, for example, membrane creation rules are not permitted in P systems with symport/antiport rules. The built-in compiler of P-Lingua detects such error. Models are specified by using `@model<model_name>` and at this stage, the allowed models are:

```
@model<membrane_division>

@model<membrane_creation>

@model<transition_psystem>

@model<probabilistic_psystem>

@model<stochastic_psystem>

@model<symport_antiport_psystem>
```

4.2.10 Modules definition

Similarities between various solutions to **NP**-complete numerical problems by using families of recognizing P systems are discussed in [8]. Also, a cellular programming language is proposed based on libraries of subroutines. Using these ideas, a P-Lingua program consists of a set of programming modules that can be used more times by the same, or other, programs.

The syntax to define a module is the following.

```
def module_name(param1,..., paramN)
{
  sentence0;
  sentence1;
  ...
  sentenceM;
}
```

The name of a module, `module_name`, must be a valid and unique identifier. The parameters must be valid identifiers and cannot appear repeated. It is possible to define a module without parameters. Parameters have a numerical value that is assigned at the module call (see below).

All programs written in P-Lingua must contain a `main` module without parameters. The compiler will look for it when generating the output file.

In P-Lingua there are sentences to define the membrane structure of a P system, to specify multisets, to define rules, to define variables and to call to other modules. Next, let us see how such sentences are written.

4.2.11 Module calls

In P-Lingua, modules are executed by using calls. The format of an sentence that calls a module for some specific values of its parameters is given next:

```
call module_name(value1, ..., valueN);
```

where `value`i is a numeric expression or a variable.

4.2.12 Definition of the initial membrane structure of a P system

In order to define the initial membrane structure of a P system, the following sentence must be written:

```
@mu = expr;
```

where `expr` is a sequence of matching square brackets representing the membrane structure, including some identifiers that specify the label and the electrical charge of each membrane.

Examples:

1. $[[\,]_2^0]_1^0 \equiv$ `@mu = [[]'2]'1`
2. $[[\,]_b^0[\,]_c^-]_a^+ \equiv$ `@mu = +[[]'b, -[]'c]'a`

4.2.13 Definition of multisets

The next sentence defines the initial multiset associated to the membrane labeled by `label`.

```
@ms(label) = list_of_objects;
```

where `label` is a membrane label and `list_of_objects` is a comma-separated list of objects. The character `#` is used to represent an empty multiset.

If a stochastic P system is being defined (that is, the file starts with `@model<stochastic>`), strings are also permitted in the initial content of a membrane:

```
@ms(label) = list_of_objects_and_strings;
```

`list_of_objects_and_strings` is a comma-separated list of objects and/or strings.

4.2.14 Union of multisets

P-Lingua allows to define the union of two multisets (recall that the input multiset is "added" to the initial multiset of the input membrane) by using a sentence with the following format.

```
@ms(label) += list_of_objects;
```

For stochastic P systems, it would be

```
@ms(label) += list_of_objects_and_strings;
```

4.2.15 Definition of rules

The definition of rules has been significantly extended in this version of P-Lingua. A general rule is defined as follow (most elements are optional):

$$u[v[w_1]_{h_1}^{\alpha_1}\dots[w_n]_{h_n}^{\alpha_n}]_h^{\alpha} \xrightarrow{k} x[y[z_1]_{h_1}^{\beta_1}\dots[z_n]_{h_n}^{\beta_n}]_h^{\beta}[s]_h^{\gamma}$$

where $u, v, w_1, \dots, w_n, x, y, z_1, \dots, z_n$ are multisets of objects or strings, $h, h_1, \dots, h_n$ are labels, $\alpha, \alpha_1, \dots, \alpha_n, \beta, \beta_1, \dots, \beta_n, \gamma$ are electrical charges and k is a numerical value.

The P-Lingua syntax for such a rule is:

uα`[v`α_1`[w1]'h1`$\dots\alpha_n$`[wN]'hN]'h --> x`β`[y`β_1`[z1]'h1`$\dots\beta_n$`[zN]'hN]'h `γ`[s]'h :: k` where `u, v, w1...wN, x, y, z1...zN, s` are comma-separated list of objects or strings (it is possible to use the character `#` in order to represent the empty multiset), `h,h1,..., hN` are labels, $\alpha, \alpha_1, \ldots, \alpha_n, \beta, \beta_1, \ldots, \beta_n, \gamma$ are identifiers for electrical charges and `k` is a numeric expression.

As mentioned before, not each type of rule is permitted in every model. Below we enumerate the possible types of rules, classified by the model in which they are allowed.

`@model<mebrane_division>`

1. The format to define evolution rules of type $[a \rightarrow v]_h^\alpha$ is given next:

   α`[a --> v]'h`

2. The format to define send-in communication rules of type $a\,[\,]_h^\alpha \rightarrow [b]_h^\beta$ is given next:

   `a`α`[]'h -->`β`[b]`

3. The format to define send-out communication rules of type $[a]_h^\alpha \rightarrow b[\,]_h^\beta$ is given next:

   α`[a]'h -->` β`[]b`

4. The format to define division rules of type $[a\,]_h^\alpha \rightarrow [b]_h^\beta[c]_h^\gamma$ is given next:

   α`[a]'h -->`β`[b]`γ`[c]`

5. The format to define dissolution rules of type $[a]_h^\alpha \rightarrow b$ is given next:

   α`[a]'h --> b`

`@model<membrane_creation>`

1. Rules 1, 2, 3 and 5 of `@model<membrane_division>` can be defined in this model, with the same format.
2. The format to define membrane creation rules of type $[a]_h^\alpha \rightarrow [[b]_{h_1}^\beta]_h^\alpha$ is given next:

   α`[a]'h -->` α`[`β`[b]'h1]'h`

`@model<transition_psystem>`

1. The format to define evolution rules of type $[u[u_1]_{h_1}, \ldots, [u_N]_{h_N} \rightarrow v[v_1]_{h_1}, \ldots, [v_N]_{h_N}, \lambda]_h$ is given next:

   ```
   [u [u1]'h1 ... [uN]'hN --> v [v1]'h1, ... [vN]'hN, @d]'h
   ```

   `@d` is a new keyword representing the containing membrane is marked to dissolved.

`@model<symport_antiport_psystem>`

1. The format to define symmetric communication rules of type $a[b]_h^\alpha \rightarrow b[a]_h^\alpha$ is given next:

   α`a[b]'h -->` β`b[a]'h`

`@model<probabilistic_psystem>`

1. The format to define rules of type $u[v]_h^\alpha \xrightarrow{p} u_1[v_1]_h^\beta$ is given next:

   `u`α`[v]'h --> u1`β`[v1]'h::p`

`@model<stochastic_psystem>`

1. The format to define multiset rewriting rules of type $u[v]_h \xrightarrow{c} u_1[v_1]_h$ is given next:

   `u[v]'h --> u1[v1]'h::c`

2. The format to define string rewriting rules of type $[u+s]_h \xrightarrow{c} [v+r]_h$ is given next:

   `[u,s]'h --> [v,r]'h::c`

- α, β and γ are identifiers for electrical charges.
- `a`, `b` and `c` are objects of the alphabet.
- `u, u1, v, v1, ..., vN` are comma-separated lists of objects that represents a multiset.
- `s` and `r` are comma-separated lists of substrings.
- `h`, `h1, ..., hN` are labels.
- `p` and `c` are real numeric expressions. The result of evaluating `p` must be between 0 and 1, and the result of evaluating `c` must be greater or equal than 0.

Some examples:

- $[x_{i,1} \rightarrow r_{i,1}^4]_2^+ \equiv$ `+[x{i,1} --> r{i,1}*4]'2`
- $d_k[\,]_2^0 \rightarrow [d_{k+1}]_2^0 \equiv$ `d{k}[]'2 --> [d{k+1}]`
- $[d_k]_2^+ \rightarrow [\,]_2^0 d_k \equiv$ `+[d{k}]'2 --> []d{k}`
- $[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- \equiv$ `[d{k}]'2 --> +[d{k}]-[d{k}]`
- $[a]_2^- \rightarrow b \equiv$ `-[a]'2 --> b`
- $Y_{i,j}[\,]_2 \xrightarrow{k_{i,8}} [B^{k_{i,12}}]_2 \equiv$ `Y{i,j}[]'2 --> [B*k{i,12}]'2::k{i,8}`
- $[RNAP+ < cap.\omega.op >]_m \xrightarrow{c} [< cap.\omega.RNAP.op >]_m \equiv$
  `[RNAP,<cap.?.op>]'m --> [<cap.?.RNAP.op>]'m::c`

4.2.16 Parametric sentences

In P-Lingua, it is possible to define parametric sentences by using the following format:

```
sentence : range1, ..., rangeN;
```

where `sentence` is a sentence of the language, or a sequence of sentences in brackets, and `range1, ..., rangeN` is a comma-separated list of ranges with the format:

```
min_value <= index <= max_value
```

where `min_value` and `max_value` are numeric expressions, integer numbers or variables, and `index` is a variable that can be used in the context of the sentence. It is possible to use the operator < instead of <=.

The sentence will be repeated for each possible values of each `index`.

Some examples of parametric sentences:

1. $[d_k]_2^0 \rightarrow [d_k]_2^+[d_k]_2^- \;:\; 1 \leq k \leq n \equiv$
   `[d{k}]'2 --> +[d{k}]-[d{k}] : 1<= k <= n;`
2. $[x_{i,j} \rightarrow x_{i,j-1}]_2^+ \;:\; 1 \leq i \leq m, 2 \leq j \leq n \equiv$
   `+[x{i,j} --> x{i,j-1}]'2 : 1<=i<=m,2<=j<=n;`

4.2.17 Inclusion of comments

The programs in P-Lingua can be commented by writing phrases into the text strings `/*` and `*/`.

4.2.18 Inclusion of debug information

Each rule sentence can optionally include a debug message which will be presented every time the rule is executed by the simulator. The syntax to write a debug message associated to a rule definition is defined as follows:

```
rule_definition @debug "debug message"
```

5 Command-line Tools

5.1 Command-line tools changes

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.

5.2 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` reports the system to compile the P system specified on a file to a file specified on another, whereas the file `input_file` contains the program 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`
  - `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`
  - `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].

5.3 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` reports 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, by default it is 3.
- 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 pLinguaCore

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. And, if the P system is defined in P-Lingua language, locates the error on 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, which define a P system, between formats, for instance, from P-Lingua language format to binary format. For more information and library documentation, please browse the P-Lingua website [25], currently under development. This library is free software published under LGPL license [22], so everyone who is interested can change and distribute this library respecting the license restrictions.

7 Tools 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 (see figure 2) 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, such as new species and a more complex behavior for the animals. In this sense, a second version of the model is presented in [2]

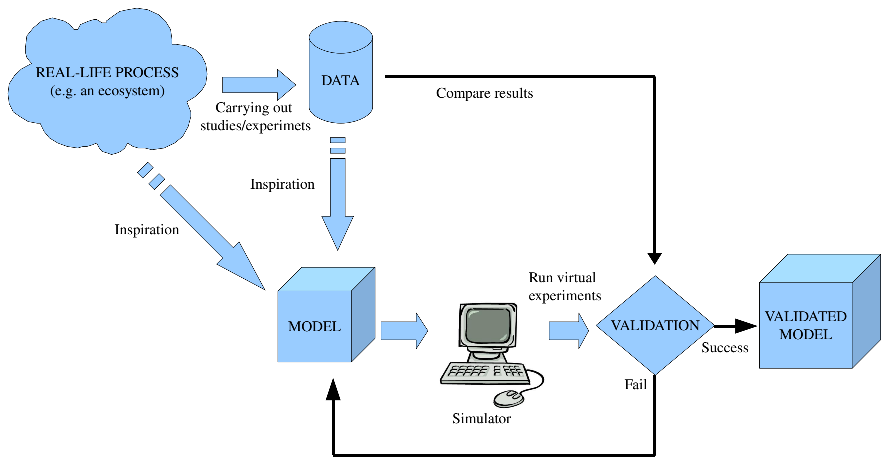


Fig. 2. Validation process

A new GPL [21] licensed JAVA application with a friendly user-interface sitting on the pLinguaCore 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 modeled ecosystem in order to make the virtual experiments suggested by experts (see figure 3). These experiments will provide results that can be interpreted in terms of hypotheses. Finally, some of these hypotheses will be selected by the experts in order to be checked in real experiments.

The current version of this application is a prototype (Figure 4), and we will publish more information as soon as possible on the P-Lingua website.

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 membrane P systems with division or creation rules, probabilistic P systems and stochastic P systems.

We have developed a JAVA library (pLinguaCore) that implements several simulators for each mentioned model and defines different formats to encode P

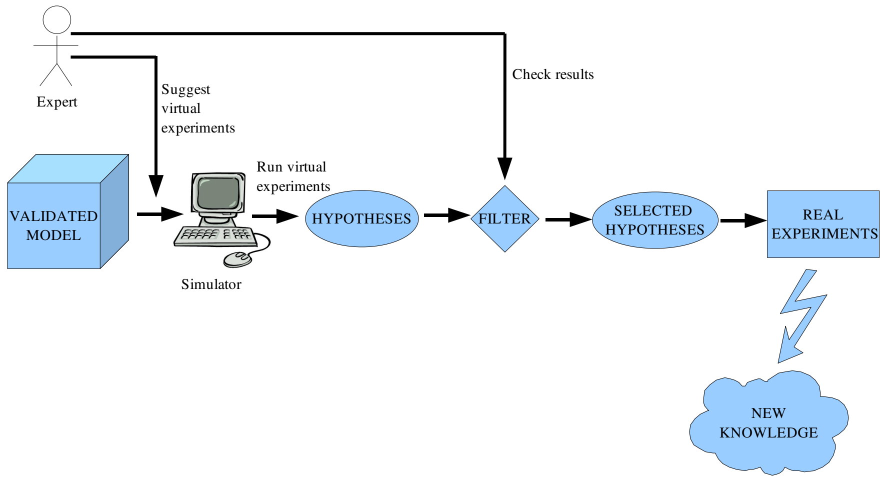
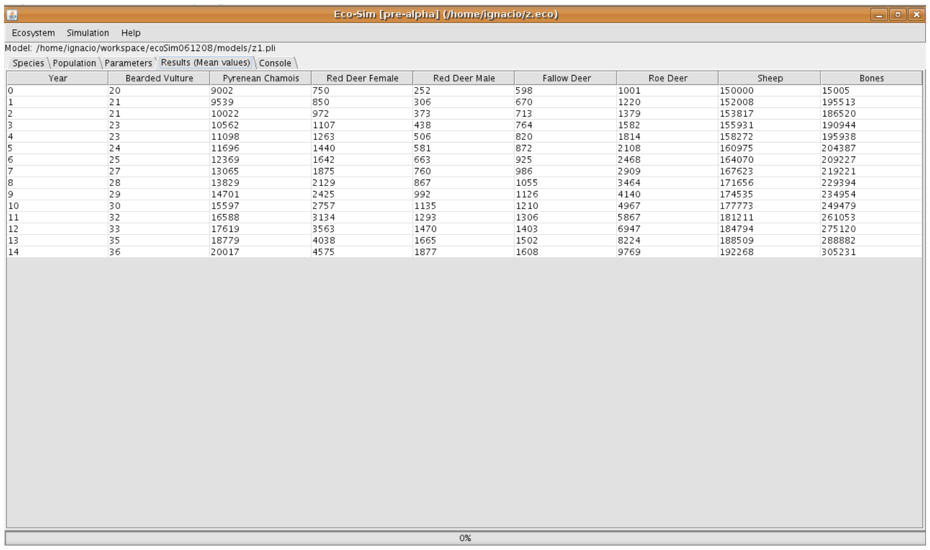


Fig. 3. Virtual experimentation



Model: /home/ignacio/workspace/ecoSim061208/models/z1.pli

Year	Bearded Vulture	Pyrenean Chamois	Red Deer Female	Red Deer Male	Fallow Deer	Roe Deer	Sheep	Bones
0	20	9002	750	252	598	1001	150000	15005
1	21	9539	850	306	670	1220	152008	195513
2	21	10022	972	373	713	1379	153817	186520
3	23	10562	1107	438	764	1582	155931	190944
4	23	11098	1263	506	820	1814	158272	195938
5	24	11696	1440	581	872	2108	160975	204387
6	25	12369	1642	663	925	2468	164070	209227
7	27	13065	1875	760	986	2909	167623	219221
8	28	13829	2129	867	1055	3464	171656	229394
9	29	14701	2425	992	1126	4140	174535	234954
10	30	15597	2757	1135	1210	4967	177773	249479
11	32	16588	3134	1293	1306	5867	181211	261053
12	33	17619	3563	1470	1403	6947	184794	275120
13	35	18779	4038	1665	1502	8224	188509	288882
14	36	20017	4575	1877	1608	9769	192268	305231

Fig. 4. A tool for simulating ecosystems

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 [25], which is currently under development, will be available to download the applications, libraries and source-code, as well as provide information about the P-Lingua project. In addition, this site aims to be a meeting point for users and developers through the use of web-tools such as forums.

The syntax of P-Lingua language is standard enough for specifying several different models of cell–like P systems. However, a new version of the language is necessary in order to specify tissue P systems and this will be 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 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 is being developed. This one is integrated into the Eclipse platform [24], 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.

Acknowledgement

The authors acknowledge the support of the project TIN2006–13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. M. Cardona, M.A. Colomer, M.J. Pérez–Jiménez, D. Sanuy and 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. *In this volume.*
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. F. Fontana, L. Bianco and V. Manca. P Systems and the Modelling fo Biochemical Oscillations, Membrane Computing, Sixth international Workshop, WMC6, Vienna, Austria, *Lecture Notes in Computer Sience*, 3850, 199–208, 2005.
5. R. Freund, S. Verlan. A Formal Framework for Static (Tissue) P Systems, *Lecture Notes in Computer Science*, 4860, 271–284, 2007.
6. 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.

7. D.T. Gillespie. Exact stochastic simulation of coupled chemical reactions, *J. Phys. Chem.*, 81, 2340–2361, 1977.
8. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez. Towards a programming language in cellular computing. *Electronic Notes in Theoretical Computer Science*, 123, 93–110 2005.
9. M. Ito, C. Martín–Vide, Gh. Păun. A characterization of Parikh sets of ET0L languages in terms of P systems. In *Words, semigroups and transducers* (M- Ito, Gh. Păun, S. Yu, eds.), 239–254, Word Scientific, Singapore 2001.
10. M. Madhu, K. Krithivasan. P systems with membrane creation: Universality and efficiency. *Lecture Notes in Computer Science*, 2055, 276–287, 2001.
11. M.A. Martínez–del–Amor, I. Pérez–Hurtado, M.J. Pérez–Jiménez, J.M. Cecilia, G.D. Guerrero, J.M. García. Simulation of Recognizer P Systems by using Manycore GPUs. *In this volume.*
12. A. Obtulowicz. Probabilistic P systems. *Lecture Notes in Computer Science*, 2597, 377–387, 2002.
13. A. Păun, Gh. Păun. The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3, 295–305, 2002.
14. Gh. Păun. Computing with Membranes, *Journal of Computer and System Sciences* 61(1) 108–143, 2000.
15. Gh. Păun. P systems with active membranes. *Journal of Automata, Languages and Combinatorics*, 6, 1, 75–90, 2001.
16. M.J. Pérez–Jiménez, F.J. Romero–Campero. Modelling Gene Expression Control using P systems: The Lac Operon, a case study. *BioSystems*, 91, 438–457, 2008.
17. M.J. Pérez–Jiménez, F.J. Romero–Campero. A model of the Quorum Sensing System in Vibrio Fischeri using P systems. *Artificial Life*, 14, 95–109, 2008.
18. M.J. Pérez–Jiménez, F.J. Romero–Campero. P Systems, a new computational modelling tool for systems biology. *Transactions on Computational Systems Biology VI*, LNBI, 4220, 176–197, 2006.
19. D. Pescini, D. Besozzi, G. Mauri and C. Zandron. Dynamical probabilistic P systems. *International Journal of Foundations of Computer Science*, 17(1), 183–195, 2006.
20. 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.
21. The GNU General Public License: `http://www.gnu.org/copyleft/gpl.html`
22. The GNU Lesser General Public License: `http://www.gnu.org/copyleft/lgpl.html`
23. Java web page: `http://www.java.com/`
24. The Eclipse Project: `http://www.eclipse.org`
25. The P-Lingua website: `http://www.p-lingua.org`

Characterizing Tractability by Tissue-Like P Systems

Rosa Gutiérrez–Escudero[1], Mario J. Pérez–Jiménez[1], Miquel Rius–Font[2]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla, Spain
`rgutierrez,marper@us.es`

[2] Department of Applied Mathematics IV
Universitat Politécnica de Catalunya
Edifici C3, Despatx 016, Av. del Canal Olímpic, s/n
08860 Castelldefels, Spain
`mrius@ma4.upc.edu`

Summary. In the framework of cell–like membrane systems it is well known that the construction of exponential number of objects in polynomial time is not enough to efficiently solve **NP**–complete problems. Nonetheless, it may be sufficient to create an exponential number of membranes in polynomial time. In the framework of recognizer polarizationless P systems with active membranes, the construction of an exponential workspace expressed in terms of number of membranes and objects may not suffice to efficiently solve computationally hard problems.

In this paper we study the computational efficiency of recognizer tissue P systems with communication (symport/antiport) rules and division rules. Some results have been already obtained in this direction: (a) using communication rules and forbidding division rules, only tractable problems can be efficiently solved; (b) using communication rules with length three and division rules, **NP**–complete problems can be efficiently solved. In this paper we show that the allowed length of communication rules plays a relevant role from the efficiency point of view of the systems.

1 Introduction

Membrane Computing is a branch of Natural Computing and starts from the assumption that the processes taking place within the compartmental structure of a living cell can be interpreted as computations [9]. The computational devices in Membrane Computing are called *P systems*. Roughly speaking, a P system consists of a membrane structure. In the compartments of this structure are multisets of objects which evolve according to given rules in a synchronous, non–deterministic, maximally parallel manner[3].

In recent years, many different models of P systems have been proposed and proved to be computationally universal. The most studied variants are characterized by a *cell-like* membrane structure, where the communication happens between a membrane and the surrounding one. In this model the membrane structure is hierarchical and the graph of the neighborhood relation between compartments is a tree.

We shall focus here on another type of P systems, the so-called (because of their membrane structure) *tissue P Systems.* Instead of considering a hierarchical arrangement, membranes are modeled as nodes of an undirected graph. This variant has two biological inspirations: intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication between cells is based on symport/antiport rules[4]. Symport rules move a number of objects across a membrane together in the same direction, whereas antiport rules move objects across a membrane in opposite directions.

Since the initial definition of tissue P systems several research lines have been developed and other variants have arisen. One of the most interesting variants of tissue P systems was presented in [12] where the definition of tissue P systems is combined with the corresponding one of P systems with active membranes, yielding the model of *tissue P systems with cell division.*

This model has been studied in depth in [1], where the importance of the cell division rules regarding the computational power of the model is shown. Working with tissue P systems without division rules it is not possible to solve computationally hard problems [2] (unless **P**=**NP**). We focus now on the influence of the length of communication rules on the computational power of tissue P systems with cell division. In particular, when limiting this length to 1, only tractable problems can be efficiently solved. A proof of this result is presented here.

The paper is organized as follows. In Section 2 we recall some definitions related to tissue P systems (further information can be found in the literature, see [15]). Section 3 is devoted to formalizing the concept of polynomial solvability of decision problems by recognizer tissue P systems. In Section 4 we introduce a dependency graph for tissue P systems and use this technique to prove the main result of the paper. Finally, the last section contains some remarks and raises open questions and future work directions.

[3] An informal overview can be found in [11] and further bibliography at [15].

[4] This method of communication for P systems was introduced in [8].

2 Recognizer Tissue P Systems

Firstly, the concept of *tissue P system of degree* $q \geq 1$ *with cell division* is introduced.

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

$$\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$$

where:

1. Γ *is a finite alphabet* (*called working alphabet*) *whose elements are called objects;*
2. Σ *is a finite alphabet* (*called input alphabet*) *strictly contained in* Γ *;*
3. $\Omega \subseteq \Gamma \setminus \Sigma$ *is a finite alphabet, describing the set of objects present in the environment in arbitrarily many copies each;*
4. $\mathcal{M}_1, \dots, \mathcal{M}_q$ *are strings over* Γ*, describing the multisets of objects placed in the* q *cells of the system;*
5. R *is a finite set of rules, of the following forms:*
   a) $(i, u/v, j)$*, for* $i, j \in \{0, 1, 2, \dots, q\}, i \neq j$*, and* $u, v \in \Gamma^*$*;*
   communication rules; $1, 2, \dots, q$ *identify the cells of the system, 0 is the environment; when applying a rule* $(i, u/v, j)$*, the objects of the multiset represented by* u *are sent from region* i *to region* j *and simultaneously the objects of the multiset* v *are sent from region* j *to region* i *(we say that the sum of the lengths of* u *and* v *is the length of the rule);*
   b) $[\ a\ ]_i \rightarrow [\ b\ ]_i [\ c\ ]_i$*, where* $i \in \{1, 2, \dots, q\}$ *and* $a, b, c \in \Gamma$*;*
   division rules; under the influence of object a*, the cell with label* i *is divided in two cells with the same label; in the first copy the object* a *is replaced by* b*, in the second copy the object* a *is replaced by* c*; all other objects are replicated and copies of them are placed in the two new cells.*
6. $i_{in} \in \{1, \dots, q\}$ *is the input cell, and* $i_{out} \in \{0, 1, \dots, q\}$ *is the output cell.*

The rules of such a system are applied in a non-deterministic maximally parallel manner as is customary in membrane computing. In each step, all cells which can evolve must evolve in a maximally parallel way (in each step we apply a multiset of rules which is maximal, no further rule can be added), with the following important remark: if a cell divides, then the division rule is the only one which is applied for that cell in that step, its objects do not evolve by means of communication rules. In other words, before division a cell interrupts all its communication channels with the other cells and with the environment; the new cells resulting from division will interact with other cells or with the environment only in the next step – providing that they do not divide once again. A cell's label precisely identifies the rules which can be applied to it.

A configuration of Π is a tuple $C = (M_0, M_1, \dots, M_q)$, where M_0 is a multiset of objects over $\Gamma \setminus \Omega$ (the objects in the environment which are in finitely many copies), and $M_1, \dots, M_q$ are multisets of objects over Γ (the objects in each cell of the system). For two configurations C_1, C_2 of Π we write $C_1 \Rightarrow_\Pi C_2$, and we say that we have a *transition* from C_1 to C_2, if we can pass from C_1 to C_2 by applying the rules from $\mathcal{R}$.

The initial configuration of the system is $(\emptyset, \mathcal{M}_1, \ldots, \mathcal{M}_q)$. For each multiset m over the input alphabet, the initial configuration of the system associated with it is $(\emptyset, \mathcal{M}_1, \ldots, \mathcal{M}_{i_{in}} \cup m, \ldots, \mathcal{M}_q)$. Then, m is an *input multiset* of every computation $\mathcal{C} = \{C_i\}_{i<r}$ such that C_0 is the initial configuration of Π associated with m.

All computations start from an initial configuration and proceeds as stated above; only halting computations give a result, which is encoded by the number of objects in the output cell i_{out} in the last configuration. From now on, we will consider that the output is collected in the environment (that is, $i_{out} = 0$, and thus we will omit i_{out} in the definition of tissue P systems). This way, if Π is a tissue P system and $\mathcal{C} = \{C_i\}_{i<r}$ is a halting computation of Π, with $C_i = (M_{i,0}, M_{i,1}, \ldots, M_{i,q})$, then the answer of the computation $\mathcal{C}$ is

$$Output(\mathcal{C}) = \Psi_{\Gamma \setminus \Omega}(M_{r-1,0})$$

where Ψ is the Parikh function.

Let us recall that **NP**–completeness has been usually studied in the framework of *decision problems*, that is problems whose solution is either *yes* or *no*. More formally, a decision problem is a pair (I_X, θ_X) where I_X is a language over a finite alphabet whose elements are called *instances*, and θ_X is a total Boolean function over I_X.

Each decision problem $X = (I_X, \theta_X)$ has a language L_X over the alphabet of I_X associated with it, defined as follows: $L_X = \{a \in I_X \mid \theta_X(a) = 1\}$. Reciprocally, each language L over an alphabet Σ has a decision problem, X_L associated with it as follows: $I_{X_L} = \Sigma^*$, and $\theta_{X_L} = \{(x, 1) \mid x \in L\} \cup \{(x, 0) \mid x \notin L\}$.

Recognizer cell-like P systems were introduced in [14] and they are the natural framework to study and solve decision problems within Membrane Computing, since deciding whether an instance of a given problem has an affirmative or negative answer is equivalent to deciding if a string belongs or not to the language associated with the problem.

In the literature, recognizer cell-like P systems are associated with P systems with *input* in a natural way. The data encoding an instance of the decision problem has to be provided to the P system in order to compute the appropriate answer. This is done by codifying each instance as a multiset placed in an *input membrane*. The output of the computation (`yes` or `no`) is sent to the environment in the last step of the computation. In this way, cell-like P systems with input and external output are devices which can be seen as black boxes, in the sense that the user provides the data before the computation starts, and then waits *outside* the P system until it sends to the environment the output in the last step of the computation.

In order to use these computational devices for solving decision problems, *recognizer tissue P systems* are introduced.

Definition 2. *A tissue P system with cell division of degree $q \geq 1$*

$$\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in})$$

is a recognizer system if the following holds:

1. *The working alphabet Γ has two distinguished objects* `yes` *and* `no`*, present in at least one copy in some initial multisets $\mathcal{M}_1, \ldots, \mathcal{M}_q$, but not present in Ω.*
2. *All computations halt.*
3. *If $\mathcal{C} = \{C_i\}_{i<r}$ is a computation of Π, then either the object* `yes` *or the object* `no` *(but not both) must have been released into the environment, and only in the last step of the computation.*

Given a recognizer tissue P system with cell division, and a computation $\mathcal{C} = \{C_i\}_{i<r}$ of Π ($r \in \mathbf{N}$), we define the result of $\mathcal{C}$ as follows:

$$Output(\mathcal{C}) = \begin{cases} \texttt{yes}, \text{ if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r-1,0}) = (1,0) \\ \qquad \wedge\, \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{k,0}) \quad = (0,0) \text{ for } k = 0, \ldots, r-2 \\ \texttt{no}, \;\; \text{if } \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{r-1,0}) = (0,1) \\ \qquad \wedge\, \Psi_{\{\texttt{yes},\texttt{no}\}}(M_{k,0}) \quad = (0,0) \text{ for } k = 0, \ldots, r-2 \end{cases}$$

That is, $\mathcal{C}$ is an accepting computation (respectively, rejecting computation) if the object `yes` (respectively, `no`) appears in the environment (only) in the halting configuration of $\mathcal{C}$.

3 Polynomial Solvability by Recognizer Tissue P systems

In this section, the definition of polynomial (uniform) solvability of decision problems by a family of cell–like P systems is extended to solvability by a family of tissue P systems.

Definition 3. *We say that a decision problem $X = (I_X, \theta_X)$ is solvable in polynomial time by a family $\mathbf{\Pi} = \{\Pi(n) : n \in \mathbb{N}\}$ of recognizer tissue P systems with cell division if the following hold:*

- *The family $\mathbf{\Pi}$ is* polynomially uniform *by Turing machines, that is, there exists a deterministic Turing machine working in polynomial time which constructs the system $\Pi(n)$ from $n \in \mathbb{N}$.*
- *There exists a pair (cod, s) of polynomial-time computable functions over I_X (called a polynomial encoding of I_X in $\mathbf{\Pi}$) such that:*
  - *For each instance $u \in I_X$, $s(u)$ is a natural number and $cod(u)$ is an input multiset of the system $\Pi(s(u))$.*
  - *The family $\mathbf{\Pi}$ is* polynomially bounded *with regard to (X, cod, s); that is, there exists a polynomial function p, such that for each $u \in I_X$ every computation of $\Pi(s(u))$ with input $cod(u)$ is halting and, moreover, it performs at most $p(|u|)$ steps.*

- *The family* $\mathbf{\Pi}$ *is* sound *with regard to* (X, cod, s)*; that is, for each* $u \in I_X$*, if there exists an accepting computation of* $\Pi(s(u))$ *with input* $cod(u)$*, then* $\theta_X(u) = 1$.
- *The family* $\mathbf{\Pi}$ *is* complete *with regard to* (X, cod, s)*; that is, for each* $u \in I_X$*, if* $\theta_X(u) = 1$*, then every computation of* $\Pi(s(u))$ *with input* $cod(u)$ *is an accepting one.*

From the soundness and completeness conditions above we deduce that every P system $\Pi(n)$ is *confluent*, in the following sense: every computation of a system with the *same* input multiset must always give the *same* answer.

We denote by $\mathbf{PMC}_{TD}$ the set of all decision problems which can be solved by means of recognizer tissue P systems with cell division in polynomial time. This class is closed under polynomial–time reduction and under complement (see [13] for a similar result for cell-like P systems). We also denote by $\mathbf{PMC}_{TD(k)}$ the set of all decision problems which can be solved by means of recognizer tissue P systems with cell division in polynomial time, by using communication rules whose length is, at most, k.

4 Dependency Graph Associated with Tissue P Systems

Let Π be a tissue P system with cell division and let all communication rules be of length 1. In this case, each rule of the system can be activated by a single object. Hence, there exists in a certain sense, a *dependency* between the object triggering the rule and the object or objects produced by its application. This dependency allows to adapt the ideas developed in [5] for cell-like P systems with active membranes to tissue P systems with cell division and communication rules of length 1.

We can consider a general pattern $(a, i) \rightarrow (b_1, j) \dots (b_s, j)$ where $i, j \in \{0, 1, 2, \dots, q\}, i \neq j$, and $a, b \in \Gamma$. Communication rules correspond to the case $s = 1$ and $b_1 = a$, and division rules correspond to the case $s = 2$ and $j = i \neq 0$. The above pattern can be interpreted as follows: from the object a in the cell (or in the environment) labeled with i we can *reach* the objects $b_1, \dots, b_s$ in the cell (or in the environment) labeled with j.

Without loss of generality we can assume that all communication rules in the system obey the syntax $(i, a/\lambda, j)$, since every rule of the form $(j, \lambda/a, i)$ can be rewritten to follow the above syntax, with equivalent semantics.

Next, we formalize these ideas in the following definition.

Definition 4. *Let* $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{in})$ *be a tissue P system of degree* $q \geq 1$ *with cell division. Let* $H = \{0, 1, \dots, q\}$*. The dependency graph associated with* Π *is the directed graph* $G_\Pi = (V_\Pi, E_\Pi)$ *defined as follows:*

$$V_\Pi = \{(a, i) \in \Gamma \times H : \ \exists j \in H \ ((i, a/\lambda, j) \in R \ \vee \ (j, a/\lambda, i) \in R) \ \vee$$
$$\exists b, c \in \Gamma \ ([a]_i \rightarrow [b]_i[c]_i \ \in R \ \vee \ [b]_i \rightarrow [a]_i[c]_i \ \in R)\},$$

$$E_\Pi = \{((a,i),(b,j)) : (a = b \ \wedge \ (i, a/\lambda, j) \in R) \vee$$
$$\exists c \in \Gamma \ ([a]_i \rightarrow [b]_i[c]_i \in R \wedge j = i)\}.$$

Proposition 1. *Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{in})$ be a tissue P system with cell division, in which the length of all communication rules is 1. Let $H = \{0, 1, \dots, q\}$. There exists a deterministic Turing machine that constructs the dependency graph G_Π associated with Π, in polynomial time (that is, in a time bounded by a polynomial function depending on the total number of rules).*

Proof. A deterministic algorithm that, given a P system Π with the set R of rules, constructs the corresponding dependency graph, is the following:

```
Input: $\Pi$ (with $R$ as its set of rules)
$V_\Pi \leftarrow \emptyset$; $E_\Pi \leftarrow \emptyset$
for each rule $r \in R$ of $\Pi$ do
  if $r = (i, a/\lambda, j)$ then
    $V_\Pi \leftarrow V_\Pi \cup \{(a,i),(a,j)\}$; $E_\Pi \leftarrow E_\Pi \cup \{((a,i),(a,j))\}$
  if $r = [a]_i \rightarrow [b]_i[c]_i$ then
    $V_\Pi \leftarrow V_\Pi \cup \{(a,i),(b,i),(c,i)\}$;
    $E_\Pi \leftarrow E_\Pi \cup \{((a,i),(b,i)),((a,i),(c,i))\}$
```

The running time of this algorithm is bounded by $O(|R|)$. □

Proposition 2. *Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \dots, \mathcal{M}_q, \mathcal{R}, i_{in})$ be a tissue P system with cell division, in which the length of all communication rules is 1. Let $H = \{0, 1, \dots, q\}$. Let Δ_Π be defined as follows:*

$$\Delta_\Pi = \{(a,i) \in \Gamma \times H : \textit{there exists a path (within the dependency graph) from } (a,i) \textit{ to } (\texttt{yes}, 0)\}.$$

Then, there exists a Turing machine that constructs the set Δ_Π in polynomial time (that is, in a time bounded by a polynomial function depending on the total number of rules).

Proof. We can construct the set Δ_Π from Π as follows:

- We construct the dependency graph G_Π associated with Π.
- Then we consider the following algorithm:

```
Input: $G_\Pi = (V_\Pi, E_\Pi)$
  $\Delta_\Pi \leftarrow \emptyset$
  for each $(a,i) \in V_\Pi$ do
    if **reachability** $(G_\Pi, (a,i), (\texttt{yes}, 0)) = \texttt{yes}$ then
      $\Delta_\Pi \leftarrow \Delta_\Pi \cup \{(a,i)\}$
```

The running time of this algorithm is of order $O(|V_\Pi| \cdot |V_\Pi|^2)$, hence[5] it is of order $O(|\Gamma|^3 \cdot |H|^3)$. □

Notation: Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in}, i_{out})$ be a tissue P system with cell division. Let m be a multiset over Σ. Then we denote $\mathcal{M}_j^* = \{(a, j) : a \in \mathcal{M}_j\}$, for $1 \le j \le q$, and $m^* = \{(a, i_{in}) : a \in m\}$.

Below we characterize accepting computations of a recognizer tissue P system with cell division and communication rules of length 1 by distinguished paths in the associated dependency graph.

Lemma 1. *Let $\Pi = (\Gamma, \Sigma, \Omega, \mathcal{M}_1, \ldots, \mathcal{M}_q, \mathcal{R}, i_{in})$ be a recognizer confluent tissue P system with cell division in which the length of all communication rules is 1. The following assertions are equivalent:*

(1) There exists an accepting computation of Π.
(2) There exists $(a_0, i_0) \in \bigcup_{j=1}^q \mathcal{M}_j^*$ and a path in the dependency graph associated with Π, from (a_0, i_0) to $(\mathtt{yes}, 0)$.

Proof. (1) $\Rightarrow$ (2) First, we show that for each accepting computation $\mathcal{C}$ of Π there exists $(a_0, i_0) \in \bigcup_{j=1}^q \mathcal{M}_j^*$ and a path $\gamma_\mathcal{C}$ in the dependency graph associated with Π from (a_0, i_0) to $(\mathtt{yes}, 0)$. By induction on the length n of $\mathcal{C}$.

If $n = 1$, a single step is performed in $\mathcal{C}$ from C_0 to C_1. A rule of the form $(j, \mathtt{yes}/\lambda, 0)$, with $a \in \Gamma, j \neq 0$, has been applied in that step. Then, $(\mathtt{yes}, j) \in \mathcal{M}_j^*$, for some $j = 1, \ldots, q$. Hence, $((\mathtt{yes}, j), (\mathtt{yes}, 0))$ is a path in the dependency graph associated with Π.

Let us suppose that the result holds for n. Let $\mathcal{C} = (C_0, C_1, \ldots, C_n, C_{n+1})$ be an accepting computation of Π. Then $\mathcal{C}' = (C_1, \ldots, C_n, C_{n+1})$ is an accepting computation of the system $\Pi' = (\Gamma, \Sigma, \Omega, \mathcal{M}_1', \ldots, \mathcal{M}_q', \mathcal{R}, i_{in})$, being $\mathcal{M}_j'$ the contents of cell j in configuration C_1, for $1 \le j \le q$. By induction hypothesis there exists an object b_0 in a cell i_0 from C_1, and a path in the dependency graph associated with Π' from (b_0, i_0) to $(\mathtt{yes}, 0)$. If (b_0, i_0) is an element of configuration C_0 (that means that in the first step a division rule has been applied to cell i_0), then the result holds. Otherwise, there is an element (a_0, j_0) in C_0 producing (b_0, i_0). So, there exists a path $\gamma_\mathcal{C}$ in the dependency graph associated with Π from (a_0, j_0) to $(\mathtt{yes}, 0)$.

[5] The Reachability Problem is the following: *given a (directed or undirected) graph, G, and two nodes a, b, determine whether or not the node b is reachable from a, that is, whether or not there exists a path in the graph from a to b.* It is easy to design an algorithm running in polynomial time solving this problem. For example, given a (directed or undirected) graph, G, and two nodes a, b, we consider a depth–first–search with source a, and we check if b is in the tree of the computation forest whose root is a. The total running time of this algorithm is $O(|V| + |E|)$, that is, in the worst case is quadratic in the number of nodes. Moreover, this algorithm needs to store a linear number of items (it can be proved that there exists another polynomial time algorithm which uses $O(\log^2(|V|))$ space).

(2) $\Rightarrow$ (1). Let us see that for each $(a_0, i_0) \in \bigcup_{j=1}^{q} \mathcal{M}_j^*$ and for each path in the dependency graph associated with Π from (a_0, i_0) to $(\texttt{yes}, 0)$, there exists an accepting computation of Π. By induction on the length n of the path.

If $n = 1$, we have a path $((a_0, i_0), (\texttt{yes}, 0))$. Then, $a_0 = \texttt{yes}$ and the computation $\mathcal{C} = (C_0, C_1)$ where the rule $(i_0, \texttt{yes}/\lambda, 0)$ belongs to a multiset of rules m_0 that produces configuration C_1 from C_0 is an accepting computation of Π.

Let us suppose that the result holds for n. Let

$$((a_0, i_0), (a_1, i_1), \ldots (a_n, i_n), (\texttt{yes}, 0))$$

be a path in the dependency graph of length $n + 1$. If $(a_0, i_0) = (a_1, i_1)$, then the result holds by induction hypothesis. Otherwise, let C_1 be the configuration of Π reached from C_0 by the application of a multiset of rules containing the rule that produces (a_1, i_1) from (a_0, i_0). Then $((a_1, i_1), \ldots (a_n, i_n), (\texttt{yes}, 0))$ is a path of length n in the dependency graph associated with the system

$$\Pi' = (\Gamma, \Sigma, \Omega, \mathcal{M}'_1, \ldots, \mathcal{M}'_q, \mathcal{R}, i_{in})$$

being $\mathcal{M}'_j$ the content of cell j in configuration C_1, for $1 \leq j \leq q$. By induction hypothesis, there exists an accepting computation $\mathcal{C}' = (C_1, \ldots, C_t)$ of Π'. Hence, $\mathcal{C} = (C_0, C_1, \ldots, C_t)$ is an accepting computation of Π. □

Next, given a family $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ of recognizer tissue P system with cell division in which the length of all communication rules is 1, solving a decision problem, we will characterize the acceptance of an instance of the problem, w, using the set $\Delta_{\Pi(s(w))}$ associated with the system $\Pi(s(w))$, that processes the given instance w. More precisely, the instance is accepted by the system if and only if there is an object in the initial configuration of the system $\Pi(s(w))$ with input $cod(w)$ such that there exists a path in the associated dependency graph starting from that object and reaching the object $\texttt{yes}$ in the environment.

Proposition 3. *Let $X = (I_X, \theta_X)$ be a decision problem. Let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer tissue P system with cell division in which the length of all communication rules is 1 solving X, according to Definition 3. Let (cod, s) be the polynomial encoding associated with that solution. Then, for each instance w of the problem X the following assertions are equivalent:*

(a) $\theta_X(w) = 1$ *(that is, the answer to the problem is* $\texttt{yes}$ *for* w*).*

(b) $\Delta_{\Pi(s(w))} \cap (\,(cod(w))^* \cup \bigcup_{j=1}^{p} \mathcal{M}_j^*\,) \neq \emptyset$*, where* $\mathcal{M}_1, \ldots, \mathcal{M}_p$ *are the initial multisets of the system* $\Pi(s(w))$*.*

Proof. Let $w \in I_X$. Then $\theta_X(w) = 1$ if and only if there exists an accepting computation of the system $\Pi(s(w))$ with input multiset $cod(w)$. From Lemma 1 this condition is equivalent to the following: in the initial configuration of $\Pi(s(w))$ with input multiset $cod(w)$ there exists at least one object $a \in \Gamma$ in a cell labeled with i such that in the dependency graph the node $(\texttt{yes}, 0)$ is reachable from (a, i).

Hence, $\theta_X(w) = 1$ if and only if $\Delta_{\Pi(s(w))} \cap \mathcal{M}_j^* \neq \emptyset$ for some $j \in \{1, \ldots, p\}$, or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$. □

Theorem 1. $\mathbf{P} = \boldsymbol{PMC}_{TD(1)}$

Proof. We have $\mathbf{P} \subseteq \mathbf{PMC}_{TD(1)}$ because the class $\mathbf{PMC}_{TD(1)}$ is closed under polynomial time reduction. Next, we show that $\mathbf{PMC}_{TD(1)} \subseteq \mathbf{P}$. Let $X \in \mathbf{PMC}_{TD(1)}$ and let $\mathbf{\Pi} = (\Pi(n))_{n \in \mathbf{N}}$ be a family of recognizer tissue P systems with cell division solving X, according to Definition 3. Let (cod, s) be the polynomial encoding associated with that solution.

We consider the following deterministic algorithm:

```
Input: An instance $w$ of $X$
  - Construct the system $\Pi(s(w))$ with input multiset $cod(w)$.
  - Construct the dependency graph $G_{\Pi(s(w))}$ associated with $\Pi(s(w))$.
  - Construct the set $\Delta_{\Pi(s(w))}$ as indicated in Proposition 2
    $answer \leftarrow$ no; $j \leftarrow 1$
    while $j \leq p \;\wedge\; answer =$ no do
        if $\Delta_{\Pi(s(w))} \cap \mathcal{M}_j^* \neq \emptyset$ then
            $answer \leftarrow$ yes
        $j \leftarrow j + 1$
    endwhile
    if $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$ then
        $answer \leftarrow$ yes
```

On one hand, the answer of this algorithm is yes if and only if there exists a pair (a, i) belonging to $\Delta_{\Pi(s(w))}$ such that the symbol a appears in the cell labeled with i in the initial configuration (with input the multiset $cod(w)$).

On the other hand, a pair (a, i) belongs to $\Delta_{\Pi(s(w))}$ if and only if there exists a path from (a, i) to $(\texttt{yes}, 0)$, that is, if and only if we can obtain an accepting computation of $\Pi(s(w))$ with input $cod(w)$. Hence, the algorithm above described solves the problem X.

The cost to determine whether or not $\Delta_{\Pi(s(w))} \cap \mathcal{M}_j^* \neq \emptyset$ (or $\Delta_{\Pi(s(w))} \cap (cod(w))^* \neq \emptyset$) is of order $O(|\Gamma|^2 \cdot |H|^2)$.

Hence, the running time of this algorithm can be bounded by $f(|w|) + O(|R|) + O(q \cdot |\Gamma|^2 \cdot n^2)$, where f is the (total) cost of a polynomial encoding from X to $\mathbf{\Pi}$, R is the set of rules of $\Pi(s(w))$, and q is the number of (initial) cells of $\Pi(s(w))$. But from Definition 3 we have that all involved parameters are polynomials in $|w|$. That is, the algorithm is polynomial in the size $|w|$ of the input. □

In [3] a polynomial time solution of the Vertex Cover problem was given by using a family of recognizer tissue P systems with cell division and communication rules of length at most 3. Then $\mathbf{NP} \cup \mathbf{co-NP} \subseteq \mathbf{PMC}_{\mathcal{TD}(3)}$.

Hence, in the framework of recognizer tissue P systems with cell division the length of the communication rules provides a borderline between efficiency and non-efficiency. Specifically, a frontier is obtained when we pass from length 1 to length 3.

5 Final Remarks and Future Work

It is well known [2] that tissue P systems with communication rules and without division rules can efficiently solve only tractable problems. It is also well known that by adding division rules we can efficiently solve **NP**–complete problems in linear time by using communication rules with length at most 3 [3].

In order to obtain new borderlines between tractability and intractability of problems, we study the possibility to restrict the length of communication rules to 1, allowing division rules. By using the dependency graph technique of cell–like P systems, we have shown that only tractable problems can be efficiently solved in that scenario.

Several questions regarding the role of the length remain open, for example:

- What happens if we consider tissue P systems using communication rules of length at most 2?
- In the solution provided in [3], antiport rules of length at most 3 were used. Would it be possible to provide another solution in which all rules of length 3 were symport?

Other open issues related to tissue P systems that may be interesting are:

- Analyzing a new role for the environment. More specifically, consider in the initial configuration only permitting objects with finite multiplicity in the environment . It seems that this new scenario would be equivalent to tissue P systems without environment, with a new distinct cell with no division rules associated. Is it still possible to solve **NP**–complete problems in polynomial time in this new framework, permitting division rules?
- Considering variations in the semantics of division rules, for example, dispensing with replication or with evolution. Division rules without replication would obey the syntax $[\ a\ ]_i \to [\ \ ]_i[\ u\ ]_i$, where $i \in \{1, 2, \ldots, q\}$, $a \in \Gamma$ and $u \in \Gamma^*$, meaning that under the influence of object a, the cell with label i is divided in two cells with the same label. The first copy contains all objects of the original cell except for a and in the second copy the content of the original cell is replaced by the multiset u. Division rules without evolution would be either of the form $[\ a\ ]_i \to [\ \ ]_i[\ \ ]_i$ or $[\ a\ ]_i \to [\ a\ ]_i[\ a\ ]_i$, where $i \in \{1, 2, \ldots, q\}$ and $a \in \Gamma$. In both cases, under the influence of object a, the cell with label i is divided in two cells. All objects are replicated and copies of them are placed in the two new cells, except for a in the first case.

Acknowledgement

The authors acknowledge the support of the project TIN2006–13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. D. Díaz–Pernil: *Sistemas celulares de tejidos: Formalización y eficiencia computacional.* Ph D. Thesis, University of Sevilla, 2008.
2. D. Díaz–Pernil, M.J. Pérez–Jiménez, A. Romero–Jiménez: Efficient simulation of tissue-like P systems by transition cell-like P systems. *Natural Computing*, online version (http://dx.doi.org/10.1007/s11047-008-9102-z).
3. D. Díaz–Pernil, M.J. Pérez–Jiménez, A. Riscos–Núñez, A. Romero–Jiménez: Computational efficiency of cellular division in tissue-like membrane systems. *Romanian Journal of Information Science and Technology*, 11, 3 (2008), 229–241.
4. P. Frisco, H.J. Hoogeboom: Simulating counter automata by P systems with symport/antiport. *Lecture Notes in Computer Science*, 2597 (2003), 288–301.
5. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez, F.J. Romero–Campero: On the power of dissolution in P systems with active membranes. *Lecture Notes in Computer Science*, 3850 (2006), 224–240.
6. M.A. Gutiérrez–Naranjo, M.J. Pérez–Jiménez, A. Riscos–Núñez, F.J. Romero–Campero, A. Romero–Jiménez: Characterizing tractability by cell-like membrane systems. In K.G. Subramanian, K. Rangarajan, M. Mukund, eds., *Formal models, languages and applications*, World Scientific, 2006, chapter 9, 137–154.
7. C. Martín–Vide, J. Pazos, Gh. Păun, A. Rodríguez–Patón: Tissue P systems. *Theoretical Computer Science*, 296 (2003), 295–326.
8. A. Păun, Gh. Păun: The power of communication: P systems with symport/antiport. *New Generation Computing*, 20, 3 (2002), 295–305.
9. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61, 1 (2000), 108–143.
10. Gh. Păun: *Membrane Computing. An Introduction.* Springer–Verlag, Berlin, 2002.
11. Gh. Păun, M.J. Pérez–Jiménez: Recent computing models inspired from biology: DNA and membrane computing. *Theoria*, 18, 46 (2003), 72–84.
12. Gh. Păun, M.J. Pérez–Jiménez, A. Riscos–Núñez: Tissue P Systems with cell division. In Gh. Păun, A. Riscos–Núñez, A. Romero–Jiménez and F. Sancho–Caparrini, eds., *Second Brainstorming Week on Membrane Computing*, Sevilla, Report RGNC 01/2004, (2004), 380–386.
13. M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini: Complexity classes in cellular computing with membranes, *Natural Computing*, 2, 3 (2003), 265–285.
14. M.J. Pérez–Jiménez, A. Romero–Jiménez, F. Sancho–Caparrini: A polynomial complexity class in P systems using membrane division, *Journal of Automata, Languages and Combinatorics*, 11, 4 (2006), 423–434.
15. P systems website: `http://ppage.psystems.eu/`

Performing Arithmetic Operations with Spiking Neural P Systems

Miguel A. Gutiérrez-Naranjo[1], Alberto Leporati[2]

[1] Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
`magutier@us.es`

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

Summary. 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.

1 Introduction

Spiking neural P systems (SN P systems, for short) have been introduced in [5] as a new class of distributed and parallel computing devices. They were inspired by *membrane systems* (also known as *P systems*) [9, 10, 12], in particular by tissue–like P systems [8], and are based on the neurophysiological behavior of neurons sending electrical impulses (*spikes*) along axons 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 consists of a number of copies of a single object type, namely the *spike*. Neurons may also contain *firing* and/or *forgetting* rules. The *firing rules* allow a neuron to send information to other neurons in the form of electrical impulses (also called spikes) which are accumulated at the target cell. The application of the rules depends on the contents of the neuron; in the general case, applicability is determined by checking the contents of the neuron against a regular set associated with the rule. As inspired from biology, when a neuron sends out spikes it becomes "closed" (inactive) for a specified period of time, that reflects the refractory period of biological neurons. During this period, the neuron does not accept new inputs and cannot "fire" (that is, emit spikes). Another important feature of biological neurons is that the length of the axon may cause a time delay before a spike arrives at the target. In SN P systems this delay is modeled by associating a delay parameter to each rule which occurs in the system. If no firing rule can be applied in a neuron, there may be the possibility to apply a *forgetting rule*, that removes from the neuron a predefined number of spikes.

Formally, an SN P system of degree $m \geq 1$, as defined in [6], is a construct of the form

$$\Pi = (O, \sigma_1, \sigma_2, \ldots, \sigma_m, syn, in, out), \text{ where:}$$

1. $O = \{a\}$ is the singleton alphabet (a is called *spike*);
2. $\sigma_1, \sigma_2, \ldots, \sigma_m$ are *neurons*, of the form $\sigma_i = (n_i, R_i)$, with $1 \leq i \leq m$, where:
   a) $n_i \geq 0$ is the *initial number of spikes* contained in σ_i;
   b) R_i is a finite set of *rules* of the following two forms:
      (1) $E/a^c \to a; d$, where E is a regular expression over a, and $c \geq 1$, $d \geq 0$ are integer numbers. If $E = a^c$, then it is usually written in the following simplified form: $a^c \to a; d$; similarly, if a rule $E/a^c \to a; d$ has $d = 0$, then we can simply write it as $E/a^c \to a$. Hence, if a rule $E/a^c \to a; d$ has $E = a^c$ and $d = 0$, then we can write $a^c \to a$;
      (2) $a^s \to \lambda$, for $s \geq 1$, with the restriction that for each rule $E/a^c \to a; d$ of type (1) from R_i, we have $a^s \notin L(E)$ (where $L(E)$ denotes the regular language defined by E);
3. $syn \subseteq \{1, 2, \ldots, m\} \times \{1, 2, \ldots, m\}$, with $(i, i) \notin syn$ for $1 \leq i \leq m$, is the directed graph of *synapses* between neurons;
4. $in, out \in \{1, 2, \ldots, m\}$ indicate the *input* and the *output* neurons of Π.

The rules of type (1) are called *firing* (also *spiking*) *rules*, and they are applied as follows. If the neuron σ_i contains $k \geq c$ spikes, and $a^k \in L(E)$, then the rule $E/a^c \to a; d \in R_i$ can be applied. The execution of this rule removes c spikes from σ_i (thus leaving $k - c$ spikes), and prepares one spike to be delivered to all the neurons σ_j such that $(i, j) \in syn$. If $d = 0$, then the spike is immediately emitted, 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. Note that, by definition, if a firing rule is applicable then no forgetting rule is applicable, and vice versa.

In each time unit, if a neuron σ_i can use one of its rules, then a rule from R_i must be used. Since two firing rules, $E_1 : a^{c_1} \to a; d_1$ and $E_2 : a^{c_1} \to a; d_2$, can have $L(E_1) \cap L(E_2) \neq \emptyset$, it is possible that two or more rules can be applied in a neuron. In such a case, only one of them is 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 in input to a specified *input neuron*. This number may be encoded in many different ways, for example as the interval of time steps elapsed between the insertion of two spikes into the neuron (note that this is a unary encoding). Other possible encodings are discussed below. To pass from a configuration to another one, for each neuron a rule is chosen among the set of applicable rules, and is executed. The computation proceeds in a sequential way into each neuron, and in parallel among different neurons. Generally, a computation may not halt. However, in any case the output of the system is 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 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 $\mathbb{N}$.

As discussed in [6], there are other possibilities to encode natural numbers read from and/or emitted to the environment by SN P systems; for example, we can consider the number of spikes contained in the input and in the output neuron, respectively, or the number of spikes read/produced in a given interval of time. Also, an alternative way to compute a function $f : \mathbb{N}^k \to \mathbb{N}$ is to introduce k natural numbers $n_1, n_2, \ldots, n_k$ in the system by "reading" from the environment a binary sequence $z = 0^b 1 0^{n_1} 1 0^{n_2} 1 \ldots 1 0^{n_k} 1 0^g$, for some $b, g \geq 0$; this means that the input neuron of Π receives a spike in each step corresponding to a digit 1 from the string z. Note that we input exactly $k+1$ spikes, and that this is again a unary encoding. Sometimes we may need to impose that the system outputs exactly two spikes and halts (sometimes after the second spike) hence producing a spike train of the form $0^{b'} 1 0^r 1 0^{g'}$, for some $b', g' \geq 0$ and with $r = f(n_1, n_2, \ldots, n_k)$. In what follows we will also consider systems which have k input neurons. For these systems, the input values $n_1, n_2, \ldots, n_k$ will arrive *simultaneously* to the system, each one entering through the corresponding input neuron. Moreover, the input numbers will be sometimes encoded in *binary* form, using the same number of bits in order to synchronize the different parts of the systems: the sequence of bits that encodes a natural number will be represented as a spike train such that, at each time step, the presence of a spike denotes 1 in the corresponding position, whereas the absence of a spike denotes 0. For further details, we refer the reader to the next sections.

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 the distance between the first two spikes of the output neuron (or the number of spikes, etc.) is the result of the computation. Note that generative SN P systems are inherently nondeterministic, otherwise they would always reproduce the same sequence of computation steps, and hence the same output. Dually, we can neglect the output neuron and use SN P systems in the *accepting* mode; for $k \geq 1$, the natural number $n_1, n_2, \ldots, n_k$ are read in input and, if the computation halts, then the numbers are accepted.

In [5] it was shown that generative SN P systems are universal, that is, can generate any recursively enumerable set of natural numbers. Moreover, a characterization of semilinear sets was obtained by spiking neural P systems with a bounded number of spikes in the neurons. SN P systems are universal also in their computing version (that is, when they compute functions $f : \mathbb{N} \to \mathbb{N}$), as it can be easily shown by simulating register machines [6]. These results can also be obtained with even more restricted forms of spiking P systems; for example, [4] shows that at least one of these features can be avoided while keeping universality: time delay (refractory period) greater than 0, forgetting rules, outdegree of the synapse graph greater than 2, and regular expressions of complex form. These results have been further extended in [3], where it is shown that universality is kept even if we remove some combinations of two of the above features. Finally, in [11] the behavior of spiking neural P systems on infinite strings and the generation of infinite sequences of 0 and 1 was investigated, whereas in [1] spiking neural P systems were studied as language generators (over the binary alphabet $\{0, 1\}$).

Spiking neural P systems have also been used to solve decision problems, both in a *semi–uniform* and in a *uniform* way [7]. When solving a problem $\mathcal{Q}$ in the *semi–uniform* setting, for each specified instance $\mathcal{I}$ of $\mathcal{Q}$ we build in a polynomial time (with respect to the size of $\mathcal{I}$) an SN P system $\Pi_{\mathcal{Q},\mathcal{I}}$, whose structure and initial configuration depend upon $\mathcal{I}$, that halts (or emits a specified number of spikes in a given interval of time) if and only if $\mathcal{I}$ is a positive instance of $\mathcal{Q}$. On the other hand, a *uniform* solution of $\mathcal{Q}$ consists of a family $\{\Pi_{\mathcal{Q}}(n)\}_{n \in \mathbb{N}}$ of SN P systems such that, when having an instance $\mathcal{I} \in \mathcal{Q}$ of size n, we introduce a polynomial (in n) number of spikes in a designated (set of) input neuron(s) of $\Pi_{\mathcal{Q}}(n)$ and the computation halts (or, alternatively, a specified number of spikes is emitted in a given interval of time) if and only if $\mathcal{I}$ is a positive instance. The preference for uniform solutions over semi–uniform ones is given by the fact that they are more strictly related to the structure of the problem, rather than to specific instances. Indeed, in the semi–uniform setting we do not even need any input neuron, as the instance of the problem is embedded into the structure (number of spikes, graph of neurons and synapses, rules) from the very beginning.

In this paper, we consider SN P systems in a completely different way. We will view SN P systems as components of a restricted Arithmetic Logic Unit in which one or more natural numbers are provided in binary form, some arithmetic operation is performed and the result is sent out (to the environment) also in bi-

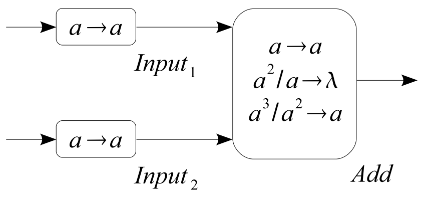


Fig. 1. An SN P system that performs the addition among two natural numbers expressed in binary form

nary 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 spike will be supplied to the 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 organized 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 amount 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 binary form. As explained above, no spike in the sequence at a given time instant means 0 in the corresponding position of the binary expansion, 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.

Formally, the SN P system for 2-addition is defined as a structure:

$$\Pi_{Add} = (O, \sigma_{Input_1}, \sigma_{Input_2}, \sigma_{Add}, syn, in_1, in_2, out)$$

where:

- $O = \{a\}$;
- $\sigma_{Input_1} = (0, R_{Input_1})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Input_2} = (0, R_{Input_2})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Add} = (0, R_{Add})$, with $R_{Add} = \{a \to a, a^2/a \to \lambda, a^3/a^2 \to a\}$;
- $syn = \{(Input_1, Add), (Input_2, Add)\}$;
- $in_1 = Input_1$, $in_2 = Input_2$;
- $out = Add$.

The following theorem holds.

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. Let t denote the current time step. In the initial configuration ($t = 0$), the system does not contain any spike. At $t = 1$, a binary digit has been provided to each of the input neurons, in the form of absence or presence of a spike. Such a digit is associated with the power 2^0 in the binary representation of the input numbers. At $t = 2$ these spikes are placed in neuron σ_{Add}. We can now divide the future behavior of σ_{Add} in three cases, depending upon the number of spikes it contains, that may be 0, 1 or 2.

- 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. This means that no spike is sent out, which can be interpreted as a 0 in the binary form of the output. Note that only one spike is consumed in the application of the rule. This means that in the next computation step the spikes in the addition neuron will be the spikes provided from the input neuron plus the one which has not been consumed. This encodes $1 + 1 = 10$.

In the general case, we observe that the spikes in the addition neuron either come from the input neurons or remain in the neuron from the previous step. Since only one spike can remain, the number of spikes contained in σ_{Add} can be 0, 1, 2 or 3 at each computation step. The cases for 0, 1, or 2 spikes are treated as described above. If there are 3 spikes, then the rule $a^3/a^2 \to a$ is applied. One spike is sent out, two of them are consumed and one remains for the next step. This encodes the operation $1 + 1 + 1 = 11$. $\square$

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

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 $11100_2 + 10101_2 = 110001_2$

the environment, at each time step during the computation. The input and the output sequences are written in bold. Note that the first instant of time for which the output is valid is $t = 3$, due to the time needed for the first input bits to reach the output neuron and to be processed.

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 second input number (the *subtrahend*) is supplied in binary form to neuron σ_{Input_2}. Neuron σ_{Input_1} is linked to three auxiliary neurons, called σ_{aux_1}, σ_{aux_2} and σ_{aux_3}, whereas σ_{Input_2} is connected with another auxiliary neuron called σ_{aux_4}. 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 neuron called σ_{gen}, which is connected with σ_{aux_flow} and σ_{aux_5}. These latter neurons are also mutually connected by two synapses. The target of the subsystem built by the neurons σ_{gen}, σ_{aux_flow} and σ_{aux_5} 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 $1 \le i \le 5$, 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.

Formally, the subtraction SN P system is defined as a structure:

$$\Pi_{Sub} = (O, \sigma_{Input_1}, \sigma_{Input_2}, \sigma_{aux_1}, \sigma_{aux_2}, \sigma_{aux_3}, \sigma_{aux_4}, \sigma_{aux_5}, \sigma_{gen},$$
$$\sigma_{aux\_flow}, \sigma_{Sub}, syn, in_1, in_2, out)$$

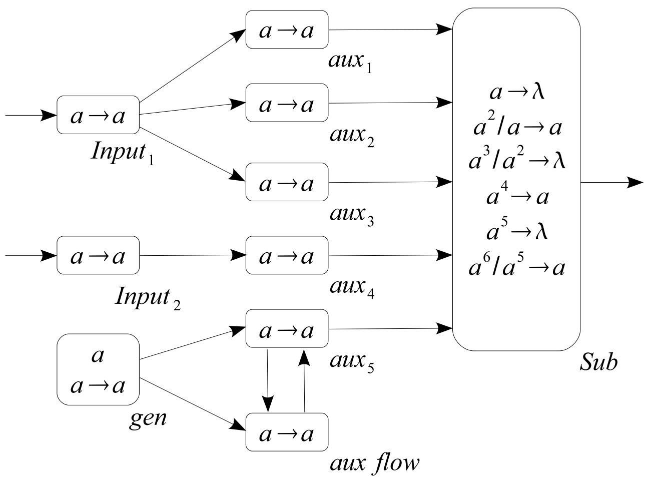


Fig. 2. An SN P system that performs the subtraction among two natural numbers expressed in binary form

where:

- $O = \{a\}$;
- $\sigma_{Input_1} = (0, R_{Input_1})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Input_2} = (0, R_{Input_2})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{aux_i} = (0, R_{aux_i})$, with $R_{aux_i} = \{a \to a\}$, for all $1 \le i \le 5$;
- $\sigma_{aux\_flow} = (0, R_{aux\_flow})$, with $R_{aux\_flow} = \{a \to a\}$;
- $\sigma_{gen} = (1, R_{gen})$, with $R_{gen} = \{a \to a\}$;
- $\sigma_{Sub} = (0, R_{Sub})$, with $R_{Sub} = \{a \to \lambda, a^2/a \to a, a^3/a^2 \to \lambda, a^4 \to a, a^5 \to \lambda, a^6/a^5 \to a\}$;
- $syn = \{(Input_1, aux_1), (Input_1, aux_2), (Input_1, aux_3), (Input_2, aux_4), (gen, aux_5), (gen, aux\_flow), (aux_5, aux\_flow), (aux\_flow, aux_5), (aux_1, Sub), (aux_2, Sub), (aux_3, Sub), (aux_4, Sub), (aux_5, Sub)\}$;
- $in_1 = Input_1$, $in_2 = Input_2$;
- $out = Sub$.

The following theorem holds.

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).*

Proof. At the beginning of the computation (time $t = 0$) all the neurons of the system are empty but neuron σ_{gen}, that contains one spike. Let us focus first on the subsystem composed of the neurons σ_{gen}, σ_{aux_flow} and σ_{aux_5}. At time $t = 1$, one spike is placed in each of the neurons σ_{aux_flow} and σ_{aux_5}, whereas neuron σ_{gen} contains no spikes. Since this neuron has no incoming synapses, it will not receive spikes and thus it will be empty along all the computation. At time $t = 2$, σ_{aux_5} has sent one spike to σ_{Sub} and another one to σ_{aux_flow}. Meanwhile, σ_{aux_flow} has sent a spike to σ_{aux_5}. This means that at $t = 2$ the spikes in both neurons σ_{aux_flow} and σ_{aux_5} are the same as those at time $t = 1$, and one spike has been sent to the subtraction neuron. Therefore, starting from $t = 2$, in each time unit neuron σ_{Sub} receives one spike from σ_{aux_5}.

Now let us focus on the input neurons. At the beginning of the computation they are empty. At time $t = 3$, neuron σ_{Sub} can receive 0 or 3 spikes from σ_{Input_1} (through σ_{aux_1}, σ_{aux_2} and σ_{aux_3}), and 0 or 1 spike from σ_{Input_2} (through σ_{aux_4}). This means that, at time $t = 3$, σ_{Sub} may contain 1, 2, 4 or 5 spikes. We can thus consider the following four cases.

- If σ_{Sub} contains 1 spike, then it comes from σ_{aux_5}. The rule $a \to \lambda$ is triggered, so that the spike is consumed and no spike is sent out. This encodes the operation $0 - 0 = 0$.
- If σ_{Sub} contains 2 spikes, then one of them comes from σ_{aux_5} and the other one from σ_{aux_4}. The rule $a^2/a \to a$ is triggered; as a consequence, one spike is sent out and one spike is consumed, so one spike remains in σ_{Sub} for the next step. This encodes $x0 - y1 = z1$ where x, y and z are numbers in binary form such that $x - (y + 1) = z$.
- If σ_{Sub} contains 4 spikes, then one of them comes from σ_{aux_5} and the other three from σ_{aux_1}, σ_{aux_2} and σ_{aux_3}. The rule $a^4 \to a$ is triggered, so that all the spikes are consumed and one spike is sent out. This encodes $1 - 0 = 1$.
- If σ_{Sub} contains 5 spikes, then each of them comes from σ_{aux_1} to σ_{aux_5}. The rule $a^5 \to \lambda$ is triggered; as a consequence, all the spikes are consumed and no spike is sent out. This encodes the operation $1 - 1 = 0$.

In the general case, we consider that the spikes in neuron σ_{Sub} come from the input neurons, or remains in the neuron from the previous step. Since only one spike can remain at each computation step, the number of spikes can be 1, 2, 3, 4, 5 or 6.

- If σ_{Sub} contains 1 spike, then it comes from σ_{aux_5}. No spike has remained from the previous step, nor comes from the input neurons. The rule $a \to \lambda$ is applied. As a result, the spike is consumed and no spike is sent out. This encodes $0 - 0 = 0$.
- If σ_{Sub} contains 2 spikes, then one of them comes from σ_{aux_5} and the other one either comes from σ_{aux_4} or remains from the previous step (no both cases may occur at the same time). The case in which the second spike comes from σ_{aux_4} has already been considered above. If it remains from the previous step, then it comes from the operation in the second digit in $x00 - y01 = z11$ where $x - (y + 1) = z$. The rule $a^2/a \to a$ is triggered; as a consequence, one spike is sent out and one spike is consumed, thus leaving one spike in the neuron for the next step.
- If σ_{Sub} contains 3 spikes, then one of them comes from σ_{aux_5}, the other one from σ_{aux_4} and the last one remains from the previous step. This situation

comes from the operation on the second digit of $x00 - y11 = z01$, where $x - (y + 1) = z1$. The rule $a^3/a^2 \to \lambda$ is applied. This means that two spikes are consumed, no spike is sent out and one spike remains in the neuron for the next step.

- If σ_{Sub} contains 4 spikes, then one of them comes from σ_{aux_5} and the other three come from σ_{aux_1}, σ_{aux_2} and σ_{aux_3}. This case has been already considered above: the rule $a^4 \to a$ is triggered, which consumes all the spikes and sends out one spike. This encodes $1 - 0 = 1$.
- If σ_{Sub} contains 5 spikes, then one of them comes from σ_{aux_5} and three of them come from σ_{aux_1}, σ_{aux_2} and σ_{aux_3}. The fifth spike can come from σ_{aux_4} or can remain from the previous step (the two events cannot occur both at the same time). The case in which it comes from σ_{aux_4} has already been considered above. If it remains from the previous step, then the situation comes from $x10 - y01 = z01$, where $x - y = z1$. The rule $a^5 \to \lambda$ is applied, which consumes all the spikes. No spike is sent out and no spike remains for the next step.
- If σ_{Sub} contains 6 spikes, then one of them remains from the previous step and the others come from σ_{aux_1}, σ_{aux_5}, σ_{aux_2} and σ_{aux_3}. The situation comes from $x_10 - y11 = z11$ where $x - (y + 1) = z$. The rule $a^6/a^5 \to a$ is triggered; as a consequence, five spikes are consumed, one spike is sent out and one spike remains for the next step. □

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 $1101100_2 - 110011_2 = 111001_2$

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. At time $t = 0$, the eight neurons of Π_{Sub} are empty and the system has not yet emitted any spike to the environment. At time $t = 1$, the minuend and the subtrahend start to be supplied to σ_{Input_1} and σ_{Input_2}, respectively. From this moment, σ_{aux_5} always contains one spike. At $t = 4$ the first digit of the output is emitted to the environment. The computation continues until the binary sequence 111001_2, which is the binary representation of 57, has been emitted.

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 equal. 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 $a \to a$.

Formally, the SN P system for checking equality is defined as a structure:

$$\Pi_{Comp} = (O, \sigma_{Input_1}, \sigma_{Input_2}, \sigma_{Comp}, syn, in_1, in_2, out)$$

where:

- $O = \{a\}$;
- $\sigma_{Input_1} = (0, R_{Input_1})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Input_2} = (0, R_{Input_2})$, with $R_{Input_1} = \{a \to a\}$;
- $\sigma_{Comp} = (0, R_{Comp})$, with $R_{Comp} = \{a \to a, a^2 \to \lambda\}$;
- $syn = \{(Input_1, Comp), (Input_2, Comp)\}$;
- $in_1 = Input_1$, $in_2 = Input_2$;
- $out = Comp$.

The system is illustrated in Figure 3. Due to its simplicity, we just give an informal justification of its working and correctness.

Both σ_{Input_1} and σ_{Input_2} send the information of the corresponding input digits simultaneously. Such information consist of 0 or 1 spike for each of the two inputs, so at each computation step there can be 0, 1 or 2 spikes in neuron σ_{Comp}. If there are 0 or 2 spikes, then no difference has been found and no spike is sent to the

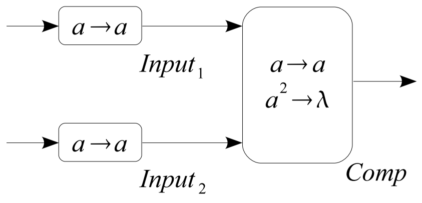


Fig. 3. An SN P system that compares two natural numbers of any length, expressed in binary form

environment. A spike is emitted only when a single spike is placed in σ_{Comp}. As explained above, this will be considered as a mark of inequality of the two binary numbers given as input, independent of any following bits.

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 such algorithm needs some kind of memory. Other algorithms for multiplication, such as Booth's algorithm (see, for example, [2]) 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 $\{\Pi_{Add}(n)\}_{n \geq 2}$ of SN P systems which allows to add numbers expressed in binary form. Precisely, for any integer $n \geq 2$ the

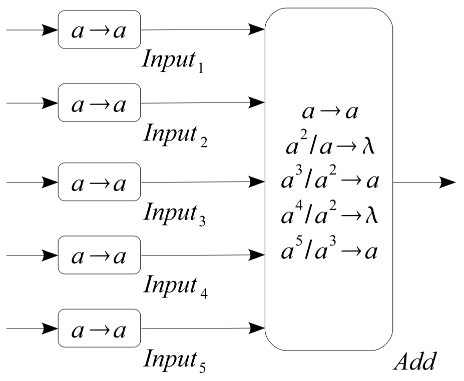


Fig. 4. An SN P system that performs the addition among five natural numbers expressed in binary form

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, $i \in \{1, \dots, n\}$, which are defined as follows:

$$\begin{array}{ll} r_i \equiv a^i/a^{k+1} \to a & \text{if } i \text{ is odd and } i = 2k+1 \\ r_i \equiv a^i/a^k \to \lambda & \text{if } i \text{ is even and } i = 2k \end{array}$$

Formally, the SN P system for n-addition is defined as a structure:

$$\Pi_{Add}(n) = (O, \sigma_{Input_1}, \dots, \sigma_{Input_n}, \sigma_{Add}, syn, in_1, \dots, in_n, out)$$

where:

- $O = \{a\}$;
- $\sigma_{Input_i} = (0, R_{Input_i})$, with $R_{Input_i} = \{a \to a\}$ for all $i \in \{1, 2, \dots, n\}$;
- $\sigma_{Add} = (0, R_{Add})$, with $R_{Add} = \bigcup_{i=1}^{n} \{r_i\}$, where:
  - $r_i \equiv a^i/a^{k+1} \to a$ if i is odd and $i = 2k+1$;
  - $r_i \equiv a^i/a^k \to \lambda$ if i is even and $i = 2k$;
- $syn = \bigcup_{i=1}^{n} \{(Input_i, Add)\}$;
- $in_i = Input_i$, for all $i \in \{1, 2, \dots, n\}$;
- $out = Add$.

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

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 $\sigma_{Input\_1}, \dots, \sigma_{Input\_n}$ 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 $U = 2k + \alpha$ ($\alpha = 1$ is U is odd and $\alpha = 0$ if U is even). The addition of $A_1, \ldots, A_n$ can be written as:

$$\begin{aligned}\sum_{i=1}^{n} A_i &= \sum_{i=1}^{n}\sum_{k=0}^{p} a_i^k 2^k = \left(\sum_{i=1}^{n}\sum_{k=1}^{p} a_i^k 2^k\right) + \sum_{i=1}^{n} a_i^0 \\ &= 2\left(\sum_{i=1}^{n}\sum_{k=1}^{p} a_i^k 2^{k-1}\right) + 2k + \alpha = 2\left(\sum_{i=1}^{n} A'_i + k\right) + \alpha\end{aligned}$$

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 the t-th digit b_t of the output, dividing the problem in the following two 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 for the next step.
- 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 ($b_t = 1$) 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. Note that the first instant of time for which the output is valid is $t = 3$. 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

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 $11_2 + 100_2 + 10_2 + 111_2 + 1_2 = 10001_2$

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 described as follows. It consists of one *input* neuron, σ_{Input}, linked to k neurons $\sigma_{aux_{11}}, \dots, \sigma_{aux_{k1}}$, where k is the number of occurrences of the digit 1 in the binary representation of n. For each $i \in \{1, \dots, k\}$, neuron $\sigma_{aux_{i1}}$ is connected with a new neuron $\sigma_{aux_{i2}}$, which is connected with $\sigma_{aux_{i3}}$, 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.

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.*

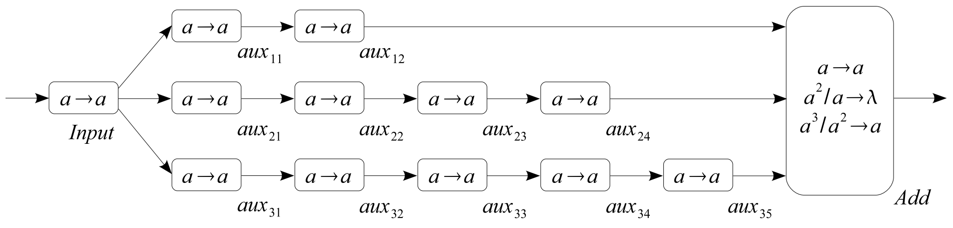


Fig. 5. An SN P system that computes the product among the natural number given as input (in binary form) and the fixed multiplier $26 = 11010_2$, encoded in the structure of the system

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 $n = \sum_{j=0}^{q} n_j 2^j$. Then we can write:

$$
\begin{aligned}
m \cdot n = m \cdot \left(\sum_{j=0}^{q} n_j 2^j \right) &= \sum_{j=0}^{q} \left(m \cdot 2^j \right) n_j \\
&= \sum \left\{ m \cdot 2^j \mid j \in \{0, \ldots, q\} \wedge n_j = 1 \right\}
\end{aligned}
$$

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 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. $\square$

We conclude this section with an example of multiplication. We will take $n = 26$ as the multiplier (hence the system $\Pi_{Mult}(26)$ illustrated in Figure 5) and we will supply the number $29 = 11101_2$ as input. Table 4 reports the number of spikes contained in neurons $\sigma_{aux_{12}}$, $\sigma_{aux_{24}}$, $\sigma_{aux_{35}}$ and σ_{Add} of $\Pi_{Mult}(26)$ during the computation, as well as the number of spikes sent to the environment. According to this computation, the output of the multiplication is 1011110010_2, which is the binary representation of 754.

6 Conclusion and Future Work

In this paper we have presented some simple SN P systems that perform the following operations on natural numbers: addition, multiple addition, comparison, and multiplication by a fixed factor. All the numbers given as inputs to these

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 $11101_2 \cdot 11010_2 = 1011110010_2$

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, that we leave as two directions for future research, are:

- Simulating by means of SN P systems the widely known Boolean circuits that perform the desired arithmetical and logical operations. However, so doing we need one system for each possible input size: as an example, we need one system to add 2-bit numbers, another one to add 3-bit numbers, and so on. On the contrary, the systems presented in this paper are able to process inputs of any length;
- Simulating by means of SN P systems the programs of register machines that perform the desired arithmetical and logical operations. This solution would overcome the need to have one system for each possible input size. But, on the other hand, the simulating SN P systems would probably be much larger than those presented in this paper.

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 be designed.

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).

References

1. H. Chen, R. Freund, M. Ionescu, Gh. Păun, M.J. Pérez-Jiménez: On string languages generated by spiking neural P systems. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Riscos-Núñez, F.J. Romero-Campero, eds., *Fourth Brainstorming Week on Membrane Computing*, Vol. I RGCN Report 02/2006, Research Group on Natural Computing, Sevilla University, Fénix Editora, 169–194.
2. M.J. Flynn: *Advanced Computer Arithmetic Design.* John Wiley Publisher, 2001.
3. M. García-Arnau, D. Peréz, A. Rodríguez-Patón, P. Sosík: Spiking neural P systems: stronger normal forms. In M.A. Gutiérrez-Naranjo, Gh. Păun, A. Romero-Jiménez, A. Riscos-Núñez, eds., *Fifth Brainstorming Week on Membrane Computing*, RGCN Report 01/2007, Research Group on Natural Computing, Sevilla University, Fénix Editora, 157–178.
4. O.H. Ibarra, A. Păun, Gh. Păun, A. Rodríguez-Patón, P. Sosík, S. Woodworth: Normal forms for spiking neural P systems. *Theoretical Computer Science*, 372, 2-3 (2007), 196–217.
5. M. Ionescu, Gh. Păun, T. Yokomori: Spiking neural P systems. *Fundamenta Informaticae*, 71, 2-3 (2006), 279–308.
6. M. Ionescu, A. Păun, Gh. Păun, M.J. Pérez-Jiménez: Computing with spiking neural P systems: Traces and small universal systems. In *DNA Computing,* 12th *International Meeting on DNA Computing (DNA12)*, Revised Selected Papers, LNCS 4287, Springer, 2006, 1–16.
7. A. Leporati, C. Zandron, C. Ferretti, G. Mauri: Solving numerical **NP**-complete problems with spiking neural P systems. In *Membrane Computing*, LNCS 4860, Springer, 2007, 336–352.
8. C. Martín-Vide, J. Pazos, Gh. Păun, A. Rodríguez-Patón: A new class of symbolic abstract neural nets: Tissue P systems. In *Proceedings of COCOON 2002*, LNCS 2387, Springer, 2002, 290–299.
9. Gh. Păun: Computing with membranes. *Journal of Computer and System Sciences*, 61 (2000), 108–143.
10. Gh. Păun: *Membrane Computing. An Introduction.* Springer–Verlag, 2002.
11. Gh. Păun, M.J. Pérez-Jiménez, G. Rozenberg: Spike trains in spiking neural P systems. *Intern. J. Foundations of Computer Science*, 17, 4 (2006), 975–1002
12. The P systems website: `http://ppage.psystems.eu/`

Solving the N-Queens Puzzle with P Systems

Miguel A. Gutiérrez-Naranjo, Miguel A. Martínez-del-Amor,
Ignacio Pérez-Hurtado, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
`magutier@us.es, mdelamor@us.es, perezh@us.es, marper@us.es`

Summary. The N-queens puzzle consists on placing N queens on an $N \times N$grid in such way that no two queens are on the same row, column or diagonal line. In this paper we present a family of P systems with active membranes (one P system for each value of N) that provides all the possible solutions to the puzzle.

1 Introduction

The N-queens puzzle is very popular among computer scientists. It is a generalization of a classic puzzle known as the 8-queens puzzle. The original one is attributed to the chess player Max Bezzel and it consists on putting eight queens on an 8×8 chessboard in such way that none of them is able to capture any other using the standard movement of the queens in chess, i.e., only one queen can be placed on each row, column and diagonal line.

The 8-queens puzzle was later generalized to the N-queens puzzle, with the same rules but placing N queens on a $N \times N$ board. The problem is computationally very expensive, since there exists $64!/(56! \times 8!) \sim 4.4 \times 10^9$ possible arrangements of 8 queens in a 8×8 chessboard and there are only 92 solutions. If two solutions are considered the same when one of them can be obtained from the other one via a rotation or a symmetry, then there are only 12 different solutions.

For this reason, the brute force algorithm is not useful with current computers. In fact, this simple puzzle is usually presented in Computer Science as an standard of use of heuristics which allows us discard options and deal with a little number of candidate solutions.

In this paper, we present a first solution to the N-queens puzzle in Membrane Computing. For that purpose, we propose a family of deterministic P systems with active membranes (the N-th element of the family solves the N-queens puzzle) such that the halting configuration encodes *all* the solutions of the puzzle. As usual, we use the massive parallelism to check all the feasible solutions at the same time and obtain the solution in a reduced number of steps.

The paper is organized as follows: In Section 2 we show how an instance of the N-queens puzzle can be expressed as a formula in conjunctive normal form. In Section 3, we briefly recall the P systems with active membranes and in Section 4, we present our family of P systems that solve SAT. The difference with other solutions is that in this case the P system does not only send `Yes` or `No` to the environment by showing the existence or not of a solution to the problem, but it keeps all the truth values that satisfy the formula. In section 5 we build the family of P systems that solve the N-queens problem by choosing the appropriate P systems from the previous family. Section 6 shows several experimental results obtained by running these solutions in an updated version of the P-lingua [1] simulator. Finally, some conclusions and new open research lines are presented.

2 Changing the Representation

The key idea of our solution is that an instance of the N-queens puzzle for a fixed N can be represented as a formula in conjunctive normal form (CNF) in such way that one truth assignment of the formula can be considered as a solution of the puzzle. In Section 5 we will show a family of P systems with active membranes associated to the N-queens puzzle which encodes the truth assignments of the associate formula in the halting configuration.

The N-queens puzzle can be represented by a formula in CNF with N^2 propositional variables s_{ij}, where s_{ij} stands for the cell (i,j) of the $N \times N$ chessboard. The variable s_{ij} is assigned true if and only if a queen is assigned to the cell (i,j). The different constraints of the puzzle can be expressed with this representation in the following way:

- There is at most one queen in each column.
  $$\psi_1 \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=j+1}^{n} (\neg s_{ij} \vee \neg s_{ik})$$
- There is at most one queen in each row.
  $$\psi_2 \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=j+1}^{n} (\neg s_{ji} \vee \neg s_{ki})$$

Next we deal with the restriction of diagonal lines. Let us call D_1 the set of diagonal lines parallel to the bottom-left to up-right diagonal and D_2 the set of diagonal lines parallel to the bottom-right to up-left diagonal. It is easy to see that any line of D_1 is characterized by a number from $\{-(n-1), -(n-2), \ldots, -1, 0, 1, \ldots, n-2, n-1\}$ which represents the difference $i-j$ in the cell (i,j). In order to fix ideas, let us consider an 8×8 and the diagonal line $\langle (1,3), (2,4), (3,5), (4,6), (5,7), (6,8) \rangle$. Cells (i,j) in this diagonal line are characterized by number -2, since in all of them $i - j = -2$.

On the other hand, any line of D_2 is characterized by a number from $\{2, 3, \ldots, 2n-1, 2n\}$ which represents the sum $i + j$. For example, in a 8×8

chessboard diagonal line $\langle(5,8),(6,7),(7,6),(8,5)\rangle$ can be characterized by number 13 since in all of them $i+j=13$.

Firstly, we consider the diagonal lines of D_1 corresponding to the bottom semi-square. Each of these lines is characterized by a number d in $\{0,\dots,n-2\}$, and each line is compounded by the cells (i,j) such that $i-j=d$. Notice that $d=n-1$ is not considered, since such a diagonal line has only one cell. The formula that codifies that there must occur one queen at most in these lines is

$$\psi_3 \equiv \bigwedge_{d=0}^{n-2} \bigwedge_{j=1}^{n-d} \bigwedge_{k=j+1}^{n-2} (\neg s_{d+j\,j} \vee \neg s_{d+k\,k})$$

The remaining diagonal lines from D_1 correspond to the values d from the set $\{-(n-2),\dots,-1\}$ and they are codified by the formula

$$\psi_4 \equiv \bigwedge_{d=-(n-2)}^{-1} \bigwedge_{j=1}^{n+d} \bigwedge_{k=j+1}^{n+d} (\neg s_{j\,j-d} \vee \neg s_{k\,k-d})$$

We also split the set D_2 into two subsets. The first of them corresponds to the lines associated with numbers d in $\{3,\dots,n+1\}$ which represents the bottom semi-square. Notice that the line with only one cell (the corner) is removed. The formula that codifies that there must appear one queen at most in these lines is

$$\psi_5 \equiv \bigwedge_{d=3}^{n+1} \bigwedge_{j=1}^{d-1} \bigwedge_{k=j+1}^{d-1} (\neg s_{j\,d-j} \vee \neg s_{k\,d-k})$$

Analogously, the upper semi-square is associated with numbers d in $\{n+2,\dots,2n-1\}$. The formula associated to these lines is

$$\psi_6 \equiv \bigwedge_{d=n+2}^{2n-1} \bigwedge_{j=d-n}^{n} \bigwedge_{k=j+1}^{d-1} (\neg s_{j\,d-j} \vee \neg s_{k\,d-k})$$

The conjunction of the previous formula says that in each column, row and diagonal line, there must be at most one queen. These conditions are satisfied by the empty board or by a board with only one queen. In order to fulfill the conditions of the N-queens puzzle we need to impose N queens to be placed. Since ψ_1 encodes that *There is at most one queen in each column*, it suffices to add the restriction *There is at least one queen in each column* in order to get that *There is exactly one queen in each column.* Since there are N columns, this leads us to place exactly N queens.

- There is at least one queen in each column.

$$\psi_7 \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} s_{ij}$$

The conjunction of these seven formulae

$$\Phi \equiv \psi_1 \bigwedge \psi_2 \bigwedge \psi_3 \bigwedge \psi_4 \bigwedge \psi_5 \bigwedge \psi_6 \bigwedge \psi_7$$

is a formula in conjunctive normal form and each truth assignment which makes it true represents a solution to the N-queens puzzle.

3 The P System Model

P systems with active membranes is one of the most studied models on Membrane Computing and it is very well-known by the P system community. It is one of the first models presented by Gh. Păun in [4]. Here we provide a brief recall of its features.

A *P system with active membranes* is a construct:

$$(V, H, \mu, w_1, \ldots, w_m, R)$$

where:

1. $m \geq 1$, is the initial *degree* of the system;
2. V is the alphabet of *symbol-objects*;
3. H is a finite set of *labels* for membranes;
4. μ is a *membrane structure*, of m membranes, bijectively labeled with elements of H;
5. $w_1, \ldots, w_m$ are strings over V, describing the initial *multisets* of objects placed in the m regions of μ;
6. R is a finite set of *evolution rules*, of the following forms:
   a) $[\, x \to y \,]_h^{\alpha}$, for $h \in H$, $\alpha \in \{+, -, 0\}$, $x \in V$, $y \in V^*$. This is an object evolution rule, associated with a membrane labeled with h and depending on the polarity of that membrane. The empty string is represented by $\lambda \in V^*$.
   b) $x\, [\,]_h^{\alpha_1} \to [\, y \,]_h^{\alpha_2}$, for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $x, y \in V$. An object from the region immediately outside a membrane labeled with h is introduced in this membrane, possibly transformed into another object, and simultaneously, the polarity of the membrane can be changed.
   c) $[\, x \,]_h^{\alpha_1} \to y\, [\,]_h^{\alpha_2}$, for $h \in H$, $\alpha_1, \alpha_2 \in \{+, -, 0\}$, $x, y \in V$. An object is sent out from membrane labeled with h to the region immediately outside, possibly transformed into another object, and simultaneously, the polarity of the membrane can be changed.
   d) $[\, x \,]_h^{\alpha} \to y$, for $h \in H$, $\alpha \in \{+, -, 0\}$, $x, y \in V$. A membrane labeled with h is dissolved in reaction with an object. The skin is never dissolved.
   e) $[\, x \,]_h^{\alpha_1} \to [\, y \,]_h^{\alpha_2} [\, z \,]_h^{\alpha_3}$, for $h \in H$, $\alpha_1, \alpha_2, \alpha_3 \in \{+, -, 0\}$, $x, y, z \in V$. A membrane can be divided into two membranes with the same label, possibly transforming some objects and their polarities.

These rules are applied according to the following principles:

- All the rules are applied in parallel and in a maximal manner. At one step, one object of a membrane can be used by only one rule (chosen in a non deterministic way), but any object being able to evolve by one rule of any form, should evolve.
- If a membrane is dissolved, its content (multiset and internal membranes) is left free in the surrounding region.

- All objects and membranes not specified in a rule and which do not evolve remain unchanged to the next step.
- At the same time, if a membrane h is divided by a rule of type (e) or dissolved by a rule of type (d) and there are objects in this membrane which evolve by means of rules of type (a), then we suppose that evolution rules of type (a) are used first and then, the division is produced. This process takes of course only one step.
- The rules associated with membranes labeled with h are used for all copies of this membrane. At one step, a membrane labeled with h can be the subject of *only* one rule of types (b)-(e).

4 A New Solution for the SAT Problem

Propositional Satisfiability is the problem to determine, for a formula of the propositional calculus, if there is an assignment of truth values to its variables for which that formula evaluates to true. By SAT we mean the problem of propositional satisfiability for formulas in conjunctive normal form.

According to Section 2, in order to solve the N-queens puzzle we need to find a truth assignment such that it makes true a formula in CNF. This problem is exactly SAT. In [3], we can find a uniform solution to the problem SAT. This design takes SAT as a decision problem and each P system of the family sends a `Yes` or `No` answer to the environment at the last step of computation specifying whether the solution exists or not. In this paper, we are not interested in the existence or not of a solution for the N-queens puzzle, but finding (and storing) the truth assignment in an effective way.

In this section we present a uniform family of deterministic recognizer P systems[1] which solves SAT as a decision problem (i.e., the P system sends a `Yes` or `No` answer to the environment at the last computation step) but it also stores truth assignments that make the formula true. We can find all the solutions to the N-queens puzzle encoded in the elementary membranes of the halting configuration.

Let us suppose that $\varphi = C_1 \wedge \cdots \wedge C_m$ in a formula in CNF, and $Var(\varphi) = \{x_1, \ldots, x_n\}$ is the set of variables in φ. Formula φ will be provided to the P systems as the following *input multiset*

$$cod(\varphi) = \{x_{ji} \mid x_i \in C_j\} \cup \{y_{ji} \mid \neg\, x_i \in C_j\}$$

Such initial multiset will be placed in the membrane with *input label* at the initial configuration. For each $(m, n) \in \mathbb{N}^2$ we consider the recognizer P system

$$(\Pi(\langle m, n\rangle), \Sigma(m, n), i(m, n))$$

where the *input alphabet* is

[1] A detailed description of recognizer P systems can be found in [3].

$$\Sigma(m,n) = \{x_{ij}, y_{ij} : \ 1 \leq i \leq m, \ 1 \leq j \leq n\}$$

the *input label* is $i(m,n) = 2$ and the P system

$$\Pi(\langle n,m \rangle) = (\Gamma(m,n), \{1,2\}, [\,[\,]_2\,]_1, w_1, w_2, R)$$

is defined as follows:
$\Gamma(m,n) = \Sigma(m,n) \cup \{d_k : 1 \leq k \leq 43n + 2m + 1\} \cup \{s_k : 1 \leq k \leq n\} \cup$
$\{t_j, f_j : 1 \leq j \leq n\} \cup \{z_{ij}, h_{ij} : 1 \leq i \leq m \, 2 \leq j \leq n,\} \cup$
$\{r_{ij}, \ 1 \leq i \leq m, 1 \leq j \leq 2n,\} \cup \{c_k : 1 \leq k \leq m+1\} \cup$
$\{v_k : 1 \leq k \leq 6n + 2m - 1\} \cup \{e, r, \texttt{Yes}, \texttt{No}\}$
The initial content of each membrane is $w_1 = \emptyset$ y $w_2 = \{d_0, v_0\}$. As usual, the initial polarization is 0. The set of rules, R, is given by:

$(a.1)$ $[\,d_j\,]_2^0 \rightarrow [\,s_{j+1}\,]_2^+ \, [\,s_{j+1}\,]_2^-$ for all $j \in \{0, \ldots, n-1\}$.
$(a.2)$ $[\,d_j\,]_2^+ \rightarrow d_j\,[\,]_2^0 \quad [\,d_j\,]_2^- \rightarrow d_j\,[\,]_2^0$ for all $j \in \{1, \ldots, n\}$.
$(a.3)$ $d_j\,[\,]_2^0 \rightarrow [\,d_j\,]_2^0$ for all $j \in \{1, \ldots, n-1\}$.
$(a.4)$ $[\,d_i \rightarrow d_{i+1}]_1^0$ for all $i \in \{n, \ldots, 3n-4\} \cup \{3n-2, \ldots, 3n+2m\}$.
$(a.5)$ $[\,d_{3n-3} \rightarrow d_{3n-2}e]_1^0$.
$(a.6)$ $[\,d_{3n+2m+1}\,]_1^0 \rightarrow \texttt{No}\,[\,]_1^+$.

By using these rules, a membrane with label 2 is divided into two membranes with the same label, but with different polarizations. These rules allow us to duplicate, in one step, the total number of internal membranes. When object d_n is reached, the counter changes its function. From d_n to $d_{4n+2m-3}$ the sequence of objects d_i is just a counter. If object $d_{4n+2m-3}$ is reached in the membrane labeled with 1 with polarization 0, then the answer No is sent to the environment.

(b) $[\,s_j \rightarrow t_j d_j\,]_2^+ \quad [\,s_j \rightarrow f_j d_j\,]_2^-$ for all $j \in \{1, \ldots, n\}$.

Instead of producing an object d_{j+1}, objects d_j $(0 \leq j \leq n-1)$ produce an object s_{j+1} (see the set $(a.1)$). With this new set of rules (b) we get such d_j from s_j. We also obtain markers t_j (true) and f_j (false) depending on the polarization of the membranes.

$$(c.1) \quad \left\{ \begin{array}{ll} [\,x_{i1} \rightarrow r_{i1}\,]_2^+ & [\,y_{i1} \rightarrow \lambda\,]_2^+ \\ [\,x_{i1} \rightarrow \lambda\,]_2^- & [\,y_{i1} \rightarrow r_{i1}\,]_2^- \end{array} \right\} \text{ for all } i \in \{1, \ldots, m\}.$$

$$(c.2) \quad \left\{ \begin{array}{ll} [\,x_{ij} \rightarrow z_{ij}\,]_2^+ & [\,y_{ij} \rightarrow h_{ij}\,]_2^+ \\ [\,x_{ij} \rightarrow z_{ij}\,]_2^- & [\,y_{ij} \rightarrow h_{ij}\,]_2^- \end{array} \right\} \text{ for all } i \in \{1, \ldots, m\} \text{ and } j \in \{2, \ldots, n\}.$$

The rules of $(c.1)$ implement a process allowing internal membranes to encode the *assignment* of a variable and, simultaneously, to check the value of all clauses by this assignment, in such a way that if the clause is true then an object $r_{i,1}$ will appear in the membrane. In other cases, the object encoding the variable will disappear. Rules from set $(c.2)$ perform a technical renaming.

$(d)\left\{\begin{array}{ll}[\,z_{ij} \rightarrow x_{ij-1}\,]_2^+ & [\,h_{ij} \rightarrow y_{ij-1}\,]_2^+ \\ [\,z_{ij} \rightarrow x_{ij-1}\,]_2^- & [\,h_{ij} \rightarrow y_{ij-1}\,]_2^-\end{array}\right\}$ for all $i \in \{1,\dots,m\}$ and $j \in \{2,\dots,n\}$.

The checking process previously described is always carried out with respect to the *first* variable appearing in the internal membrane. Hence, the rules of (d) take charge of making a cyclic path through all the variables to get that, initially, the first variable is x_1, then x_2, and so on.

$(e.1)$ $[\,r_{ij} \rightarrow r_{ij+1}\,]_2^0$ for all $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,2n-1\}$.
$(e.2)$ $[\,r_{1\,2n}\,]_2^+ \rightarrow r_{1\,2n}[\,]_2^-$.
$(e.3)$ $[\,r_{1\,2n} \rightarrow \lambda\,]_2^-$.
$(e.4)$ $[\,r_{j\,2n} \rightarrow r_{j-1\,2n}\,]_2^-$ for all $j \in \{2,\dots,m\}$.
$(e.5)$ $r_{1\,2n}\,[\,]_2^- \rightarrow [\,r\,]_2^+$.

In objects r_{jk}, index j represents a clause. Index i evolves in all the membranes until reaching $r_{j\,2n}$ for each $j \in \{1,\dots,m\}$. These objects $r_{j\,2n}$ play their role at the checking stage.

$(f)\, e\,[\,]_2^0 \rightarrow [\,c_1\,]_2^+$.

Objects e are created in membrane 1 by objects d_{3n-3}. They send objects c_1 into the elementary membranes and start the checking stage.

$(g.1)$ $[\,v_i \rightarrow v_{i+1}\,]_2^0$ $[\,v_i \rightarrow v_{i+1}\,]_2^+$ $[\,v_i \rightarrow v_{i+1}\,]_2^-$, for all $i \in \{0,\dots,6n+2m-2\}$.
$(g.2)$ $[\,v_{6n+2m-1} \rightarrow \lambda]_2^-$.
$(g.3)$ $[\,v_{6n+2m-1}]_2^+ \rightarrow r$.

The sequence of objects v_i is merely a counter. If object $v_{6n+2m-1}$ appears in an elementary membrane with negative polarization, it just disappears. Otherwise, if the polarization is positive, it dissolves the membrane. The importance of this counter is crucial since all the membranes which do not encode a solution are dissolved.

$(h.1)$ $[\,c_j \rightarrow c_{j+1}\,]_2^-$ for all $j \in \{1,\dots,m\}$.
$(h.2)$ $[\,c_{m+1}\,]_2^+ \rightarrow c_{m+1}\,[\,]_2^-$.
$(h.3)$ $[\,c_{m+1}\,]_1^0 \rightarrow \texttt{Yes}\,[\,]_1^+$.

Evolution from c_1 to c_{m+1} is completed only in the elementary membranes that represents truth assignments that make the whole formula true. If an object c_{m+1} reaches the skin, then it sends out an object `Yes`.

(i) $[\,r \rightarrow \lambda]_2^+$.

Just a cleaning rule.

4.1 Some notes on the computation

All the P systems of the family are deterministic. The first stage of the computation finishes with configuration C_{4n-1}. In such configuration, there are 2^n elementary membranes, one for each possible truth assignment of the set of variables $\{x_1, \dots, x_n\}$. We also have 2^n copies of the object d_n in the membrane labeled by 1. The checking stage starts with the configuration C_{6n-2}. An object c_1 appears in every elementary membrane in such configuration. If the answer is `Yes`, then the halting configuration is C_{6n+2m}, otherwise, if the answer is `No`, then the halting configuration is $C_{6n+2m+1}$.

If the answer is `No`, all the elementary membranes have been dissolved and the unique membrane in the halting configuration is the skin. If the answer is `Yes`, at least one elementary membrane has not been dissolved. Each elementary membrane in the halting configuration represents a truth assignment that makes the formula true. The encoding is quite easy: for each $i \in \{1, \dots, n\}$, either object t_i or object f_i belongs to the elementary membrane. Objects t_i means that in this assignment, variable x_i takes the value *true*, and f_i means that such variable is false.

It is easy to check that if we have a formula with n variables and m clauses we need $10mn + 26n + 5m + 6$ rules.

5 A Family of P Systems

In Section 2, we have seen that an instance of the N-queens puzzle can be represented as a formula in CNF and in Section 4 we have shown that there exists one P system with active membranes which is able to solve any instance of the problem SAT with m clauses and n variables.

In this section, we will select one P system of the family for each N. Since there exist one P system associated to each pair (m, n) where m is the number of clauses and n the number of variables, it only remains to know how many variables and how many clauses there are in the CNF formula associated to each instance on the N-queens puzzle. Both amounts are fixed by the following theorem.

Theorem 1. *Given an integer $N \geq 3$, the formula Φ in conjunctive normal form that encodes the N-queens puzzle according to the previous description has N^2 variables and $\frac{1}{3}(5N^3 - 6N^2 + 4N)$ clauses.*

Proof. It is trivial to check that the number of variables is N^2, since each variable represents a cell in a $N \times N$ chessboard. In order to obtain the number of clauses, we will sum the number of clauses in clauses ψ_1, ...,ψ_7, since the global formula Φ is the conjunction of these seven formulae in CNF.

- Clause 1: $\psi_1 \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=j+1}^{n} (\neg s_{ij} \vee \neg s_{ik})$

For each column $i \in \{1,\ldots,n\}$ we compare the cells pairwise, so, in the formula $\wedge_{j=1}^{n} \wedge_{k=j+1}^{n} (\neg s_{ij} \vee \neg s_{ik})$ there are $1+2+\cdots+N-1 = \frac{N(N-1)}{2}$ clauses and in ψ_1 the number of clauses is

$$M_1 = \frac{N^2(N-1)}{2}$$

- Clause 2: $\psi_2 \equiv \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=j+1}^{n} (\neg s_{ji} \vee \neg s_{ki})$

The situation is symmetric to the previous one, but considering rows instead of columns, so the number of clauses is the same

$$M_2 = \frac{N^2(N-1)}{2}$$

- Clause 3: $\psi_3 \equiv \bigwedge_{d=0}^{n-2} \bigwedge_{j=1}^{n-d} \bigwedge_{k=j+1}^{n-2} (\neg s_{d+j\,j} \vee \neg s_{d+k\,k})$

In this set of diagonal lines, the first one corresponds to $d = 0$. In this line there are n cells. The number of pairwise comparisons between cells of this line is $S_{n-1} = 1+2+\cdots+n-1$. The following line corresponds to $d = 1$. In this line there are $n-1$ cells and the number of pairwise comparisons is $S_{n-2} = 1+2+\cdots+n-2$. The whole number of comparisons is the sum $M = S_1 + S_2 + \cdots + S_{n-1}$ where

$$S_k = 1+\cdots+k = \frac{k(k+1)}{2} = \frac{1}{2}k^2 + \frac{1}{2}k \quad \text{for all } k \in \{1,\ldots,n-1\}$$

Bearing in mind that

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

we have that

$$M_3 = \sum_{k=1}^{n-1} S_k = \frac{1}{2}\left(\sum_{k=1}^{n-1} k^2 + \sum_{k=1}^{n-1} k\right) = \frac{1}{2}\left(\frac{(n-1)\,n\,(2n-1)}{6} + \frac{n(n-1)}{2}\right)$$

$$= \frac{1}{6}\, n\,(n+1)\,(n-1)$$

- Clause 4: $\psi_4 \equiv \bigwedge_{d=-(n-2)}^{-1} \bigwedge_{j=1}^{n+d} \bigwedge_{k=j+1}^{n+d} (\neg s_{j\,j-d} \vee \neg s_{k\,k-d})$

The reasoning for ψ_3 is also valid in this case. The difference is that we sum from S_1 to S_{n-2}, so the number of clauses in this case is

$$M_4 = \frac{1}{6}\, n\,(n-1)\,(n-2)$$

- Clause 5: $\psi_5 \equiv \bigwedge_{d=3}^{n+1} \bigwedge_{j=1}^{d-1} \bigwedge_{k=j+1}^{d-1} (\neg s_{j\,d-j} \vee \neg s_{k\,d-k})$
- Clause 6: $\psi_6 \equiv \bigwedge_{d=n+2}^{2n-1} \bigwedge_{j=d-n}^{n} \bigwedge_{k=j+1}^{d-1} (\neg s_{j\,d-j} \vee \neg s_{k\,d-k})$

  Due to the symmetry, the number of clauses in these formulae are the same than in ψ_3 and ψ_4, which are

$$M_5 = \frac{1}{6}\,n\,(n+1)\,(n-1) \qquad \text{and} \qquad M_6 = \frac{1}{6}\,n\,(n-1)\,(n-2)$$

- Clause 7: $\psi_7 \equiv \bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} s_{ij}$

  Trivially ψ_7 has n clauses, $M_7 = n$.

Finally, a simple calculus show that the whole number of clauses is

$$M_1 + M_2 + \cdots + M_7 = \frac{1}{3}(5n^3 - 6n^2 + 4n)$$

From the previous theorem we have the set of all solutions of the N-queens puzzle are encoded in the elementary membranes of the halting configuration of the P system

$$\Pi(\langle \frac{1}{3}(5N^3 - 6N^2 + 4N), N^2 \rangle)$$

with input membrane $i(\langle \frac{1}{3}(5N^3 - 6N^2 + 4N), N^2 \rangle) = 2$ and input the appropriate multiset on $\Sigma(\langle \frac{1}{3}(5N^3 - 6N^2 + 4N), N^2 \rangle) = 2$ encoding the formula Φ.

6 Experimental Results

In this section, we show a couple of experimental results obtained by running the corresponding P systems with an updated version of the P-lingua simulator [1]. The experiments were performed on a one-processor Intel core2 Quad (with 4 cores at 2,83Ghz), 8GB of RAM and using a C++ simulator over the operating system Ubuntu Server 8.04.

The 3-queens puzzle. In this case the problem consists on putting three queens on a 3×3 chessboard. According to our representation, the puzzle can be expressed by a formula in CNF with 9 variables and 31 clauses. This means that we can use the P system $\Pi(\langle 31, 9 \rangle)$ from the family that solves the SAT problem to obtain the solution. The *input multiset* has 65 elements and the P system has 3185 rules.

Along the computation, $2^9 = 512$ elementary membranes need to be considered in parallel. Since the simulation is carried out in a one-processor computer, in the simulation, these membranes are evaluated sequentially. It takes 7 seconds to reach the halting configuration. It is the 117-th configuration and in this configuration

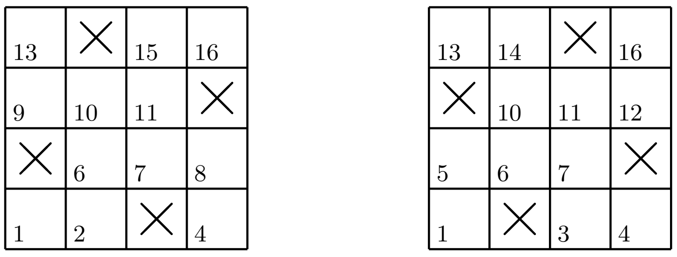


Fig. 1. Solutions to the 4-queens puzzle

one object `No` appears in the environment. As expected, this means that we cannot place three queens on a 3×3 chessboard satisfying the restriction of puzzle.

The 4-queens puzzle. In this case, we try to place four queens on a 4×4 chessboard. According to our representation, the puzzle can be expressed by a formula in CNF with 16 variables and 80 clauses. This means that we can use the P system $\Pi(\langle 80, 16\rangle)$ from the family that solves the SAT problem to obtain the solution. The *input multiset* has 168 elements.

Along the computation, $2^{16} = 65536$ elementary membranes need to be considered in parallel and the P system has 13622 rules.

The simulation takes 20583 seconds ($>$ 5 hours) to reach the halting configuration. It is the 256-th configuration and in this configuration one object `Yes` appears in the environment. This means that there exists at least one solution to the problem. In order to know such solutions, we check the multiset of the elementary membranes. In this case there are two elementary membranes in the halting configuration with the following multisets:

$$w_1 = \{f_1, f_2, t_3, f_4, t_5, f_6, f_7, f_8, f_9, f_{10}, f_{11}, t_{12}, f_{13}, t_{14}, f_{15}, f_{16}\}$$

$$w_2 = \{f_1, t_2, f_3, f_4, f_5, f_6, f_7, t_8, t_9, f_{10}, f_{11}, f_{12}, f_{13}, f_{14}, t_{15}, f_{16}\}$$

Such multisets encode the solution showed in the Figure 1

7 Conclusions and Future Work

In this paper we have presented a first solution to the N-queens puzzle based on Membrane Computing. The necessary resources and the number of computational steps for obtaining all the solutions of the puzzle are polynomial in N. Nonetheless, the simulation in one-processor computer needs an exponential amount of time.

Looking for solutions to toy puzzles as this one is not viable in the current conditions. This leads to us to three reflections: The first one is the necessity of giving the first steps for a wet implementation of P systems. The second aim, in the short-term, is to explore the possibilities of the most recent hardware, able to implement in parallel a big amount of simple rules as a realistic implementation of P systems [2]. A third research line is to follow the same path than other computation models: To avoid brute force algorithms and start to go deeply in the study of heuristics in the design of cellular solutions.

Acknowledgements

The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. D. Díaz-Pernil, I. Pérez-Hurtado, M.J. Pérez-Jiménez, A. Riscos–Núñez: A P-Lingua Programming Environment for Membrane Computing. LNCS 5391, Springer, 2009, 187–203.
2. M.A. Martínez-del-Amor, I. Pérez-Hurtado, M.J. Pérez-Jiménez, J.M. Cecilia, G. Guerrero, J.M. García: Simulation of Recognizer P Systems by Using Manycore GPUs. In these proceedings.
3. M.J. Pérez-Jiménez, A. Romero-Jiménez, F. Sancho-Caparrinini: A polynomial complexity class in P systems using membrane division. In E. Csuhaj-Varjú, C. Kintala, D. Wotschke, G. Vaszil, eds., *Proceedings of the 5th Workshop on Descriptional Complexity of Formal Systems*, DCFS 2003, Computer and Automaton Research Institute of the Hungarian Academy of Sciences, 2003, 284–294.
4. Gh. Păun: *Membrane Computing. An Introduction.* Springer-Verlag, Berlín, 2002.

Computing Backwards with P Systems

Miguel A. Gutiérrez-Naranjo, Mario J. Pérez-Jiménez

Research Group on Natural Computing
Department of Computer Science and Artificial Intelligence
University of Sevilla
Avda. Reina Mercedes s/n, 41012, Sevilla, Spain
magutier@us.es, marper@us.es

Summary. Searching all the configurations C' such that produce a given configuration C, or, in other words, computing backwards in Membrane Computing is an extremely hard task. The current approximations are based in heavy hand-made calculus by considering the specific features of the given configuration. In this paper we present a general method for characterizing all the configurations C' such that produce a given configuration C in transition P systems without cooperation and without dissolution.

1 Introduction

Given a computational model with a universal clock, where the time is considered in a discrete way and the transition from a state to the next one is made by a set of rules, it is usual to wonder about the previous state of a given one, or in other words, to wonder about the possibility of computing backwards.

Note that the determinism of the model does not make the solution easier, since the determinism of the computation does not lead to the determinism of the reverse computation. One can go deterministically from S to S_0 and from S' to S_0, but given S_0, the reversed computation is not deterministic. A special situation is considered when the rules are *reversible*. In this case, it suffices to apply the reversed rules to S_1 according to the computational model to obtain the desired states[1].

In this paper we study the problem of characterizing the set of configurations of a P systems that produce a given configuration in one computational step. We study the case in which the P system is not necessarily deterministic and the rules are not reversible in general. We will consider a restricted version of transition P systems without cooperation where the membrane structure does not change along the computation.

The paper is organized as follows: first we expose an example that shows the necessity of finding a method for computing backwards, avoiding the heavy calculus

[1] This case is studied for P systems in [1].

based on specific features of the given configuration. Next, our P system model is briefly introduced and we present our representation for configurations and rules in such a P system. In Section 6 we prove our main result: The computation of all the configurations C' such that produce a given configuration C can be reduced to find solutions of a system of linear equations with values[2] in $\mathbb{N}$. In Section 7 we provide a general method of calculus based on our theorem. Finally, some conclusions and new open research lines are presented.

2 Motivation

The reader is assumed to be familiar with basic elements of membrane computing, e.g., from [4] Let us start with the P system Π with alphabet $\Gamma = \{a, b, c\}$, set of labels $H = \{e, s\}$, membrane structure $\mu = [\,[\,]_e\,]_s$ and set of rules R

Rule 1: $[\,a \to b^2c\,]_e$	**Rule 4:** $[\,b \to a\,]_s$
Rule 2: $[\,a\,]_e \to a\,[\,]_e$	**Rule 5:** $a\,[\,]_e \to [\,c\,]_e$
Rule 3: $[\,b \to c^2\,]_s$	**Rule 6:** $[\,c \to a\,]_e$

In Section 3, we will give a detailed description of the P system model studied in this paper, but by now it is enough to know that all the rules are applied in a non-deterministic maximal parallel way as usual in the general framework of Membrane Computing.

Let us consider now configuration $C' = [\,[a^2b]_e\, a^2c\,]_s$, i.e., the configuration in which the multiset placed in the membrane labeled by e is a^2b and the multiset in the membrane s is a^2c. Our problem is to find the configuration (or configurations) C such that we can go from C to C' in *one* computational step. In other words, we want to compute backwards from C and characterize *all* the configurations C such that produce C' in one computation step.

We can reason in the following way:

- We find two objects a in the membrane labeled by e in the configuration C'. Since rules 1 and 2 consume all the objects in the membrane e from the previous configuration C, we conclude that such pair of objects a must be produced by the application of rule(s) of Π. It is easy to check that only rule 6 produces objects a in membrane e, then the number of objects c in configuration C must be at least 2. If we look at the set of rules again, we observe that object c in membrane e only triggers rule 6. Hence, if the number of objects c in e is higher than 2 we conclude that the number of objects a in the membrane e in the configuration C must be greater than 2. Therefore, we conclude that the number of objects c in the membrane e in configuration C is equal to 2.
- We find one object b in the membrane labeled by e in configuration C'. The unique rule that can produce it is rule 1, but the application of the rule produces at least two objects b in membrane e. Then we conclude that rule 1 is not

[2] We represent by $\mathbb{N} = \{0, 1, 2, \dots\}$ the set of natural numbers.

applied. The occurrence of such object b can only be explained by considering its occurrence in configuration C. As one can check, no rule is triggered by object b in the membrane e, then the number of objects b in membrane e in the configuration C equals to 1.

- No object c are placed in the membrane e in C'. All such objects from the previous configuration C are consumed by rule 6, so no object c in the membrane e imply that rules 1 and 5 have not been triggered. From the previous paragraph, it is known that rule 5 has not been applied. Since all the objects a in membrane s send objects e into membrane c by means of rule 5 and the numbers of objects c in such membrane in configuration C' is zero, we conclude that in configuration C no objects a are placed in the membrane s.
- We find one object c in the membrane labeled by s in configuration C'. The unique rule that can produce it is rule 3, but the application of the rule produces at least two objects c in membrane s. Then we conclude that rule 3 is not applied. The occurrence of such object b can only be explained by considering its occurrence in configuration C. As one can check, no rule is triggered by the object c in the membrane s, then the number of objects c in membrane s in the configuration C equals 1.
- Finally, we find two objects a in the membrane labeled by s in the configuration C'. Since rule 5 consumes all the objects in the membrane e from the previous configuration C, we conclude that such objects a must be produced by the application of rule(s) of Π. Rules 2 and 4 produce objects a in membrane s. Rule 2 is triggered by an object a in the membrane e and rule 4 is triggered by an object b in membrane s. We can also check that all the objects b in s produce objects a. Nonetheless, an object a in the membrane e can trigger rules 1 and 2. Fortunately, we have seen that rule 1 is not triggered, so can conclude that all the objects a in membrane e trigger rule 2. We conclude that the number of objects a in membrane e in the configuration C and the number of objects b in the membrane s must be less than or equal to 2 and the sum of both numbers must be equal to 2.

Bearing in mind these considerations, there are three configurations C such that produce C' in one computation step:

- $C_1 = [\,[\,bc^2\,]_e\, b^2c\,]_s$, i.e., $w_e = bc^2$ and $w_s = b^2c$. It is easy to check that by applying the rules 4 and 6 we obtain the configuration $C' = [\,[a^2b\,]_e\, a^2c\,]_s$.
- $C_2 = [\,[\,abc^2\,]_e\, bc\,]_s$, i.e., $w_e = abc^2$ and $w_s = bc$. In this case, C' is obtained by applying the rules 2, 4 and 6.
- $C_3 = [\,[\,a^2bc^2\,]_e\, c\,]_s$, i.e., $w_e = a^2bc^2$ and $w_s = c$. In this case, C' is obtained by applying the rules 2 and 6.

A question arises in a natural way: Could this reasoning be automatic? In other words, given a P system and a configuration C', is there an algorithm such that outputs the set $\mathcal{C}$ of configurations C and produce C' in one computational step?

We can even go beyond. We wonder if there exists an algorithm such that it takes a P system Π as input and it outputs a mapping $\mathcal{R}_\Pi$ which, for *every* configuration C' of Π, $\mathcal{R}_{\Pi}(C')$ is the set of all computations C such that C' is obtained from C in one computational step. In this paper, we will give a positive answer to both questions. Before, we need to recall the connections between P systems and Linear Algebra.

3 The P System Model

Throughout this paper, we will consider a restricted form of transition P systems without dissolution and without output membrane. Considering an output membrane is irrelevant for our study, since we are not interested in the objects placed in a particular membrane, but in the computation process itself. We also restrict the type of rules. Cooperation is not allowed and then rules are triggered by only one object.

Namely, along this paper a P system of degree m is a tuple

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

where:

- Γ is an alphabet whose elements are called *objects*;
- H is the set of m labels and m is called the *degree* of Π.
- μ is the membrane structure of the P system; membranes are bijectively labelled with the elements of H;
- $w_{h_1}, \ldots, w_{h_m}$ are strings that represent multisets over Γ associated with each membrane of μ;
- $R = \{R_1, \ldots, R_m\}$ is the set of sets of rules, where R_i with $i \in \{1, \ldots, m\}$ is a finite set of *evolution rules* over Γ. The type of evolution rules of R_i depends on the membrane structure μ. Let $j_1, \ldots, j_r$ be the labels of membranes immediately inside the membrane i. An evolution rule of R_i is of the form $a \to v$, where a is an object from Γ and v is an string over Γ^i_{tar}, where $\Gamma^i_{tar} = \Gamma \times TAR_i$, for $TAR_i = \{here, out\} \cup \{in_{j_k} \mid k \in \{1, \ldots, r\}\}$.

The symbols *here*, *out* and in_{j_k} are called *target commands*. The rules are applied in a non-deterministic maximally parallel way. Given a rule $a \to v$, the effect of applying this rule in a compartment i is to remove the object a and to insert the objects specified by v in the regions designated by the target commands associated with the objects from v. In particular,

- if v contains $(a, here)$, the object a will be placed in the same region where the rule is applied;
- if v contains (a, out), the object a will be placed in the compartment that surrounds the region where the rule is applied;
- if v contains (a, in_j), the object a will be placed in compartment j, provided that j is immediately inside i.

In one step, each object in a membrane can only be used for one rule (non deterministically chosen when there are several possibilities), but any object which can evolve by a rule of any form must do it. All the elements which are not involved in any of the rules to be applied remain unchanged. Several rules can be applied to different objects in the same cell simultaneously.

Along the computation, the multisets associated with the membranes can change, but the alphabet Γ, the set of labels H, the membrane structure μ and the set of rules R are constant. We will call the 4-uple (Γ, H, μ, R) the *skeleton* of the P system.

Notice that the P system presented in Section 2 is a particular case of this P system model with a slight change of notation in the rules

1. Notation $[a \to v]_h$ where $h \in H$, $a \in \Gamma$ and v is a string over Γ is a short notation to indicate that the rule $a \to (v_1, here) \dots (v_n, here)$ belongs to the set of rules R_h, with $v = v_1 \dots v_n$.
2. Notation $a[\,]_h \to [v]_h$ where $h \in H$, $a \in \Gamma$ and v is a string over Γ is a short notation to indicate that the rule $a \to (v_1, in_h) \dots (v_n, in_h)$ belongs to the set of rules R_{h^*}, with h^* the label of the membrane surrounding the membrane h and $v = v_1 \dots v_n$.
3. Notation $[a]_h \to v[\,]_h$ where $h \in H$, $a \in \Gamma$ and v is a string over Γ is a short notation to indicate that the rule $a \to (v_1, out) \dots (v_n, out)$ belongs to the set of rules R_h, with $v = v_1 \dots v_n$.

4 Changing the Point of View

The key idea of the present paper is to consider an algebraic representation for the configurations and the rules of a P system. The starting point is the representation used in [2], but we introduce several changes.

First, our elementary objects are pairs of type $(a, h) \in \Gamma \times H$ meaning that object $a \in \Gamma$ is placed in the membrane (labeled by) $h \in H$. Roughly speaking, transitions in P systems are performed by rules in which the occurrence of an element a_0 in a membrane h_0 produces the occurrence of β_1 copies of element a_1 in membrane h_1, β_2 copies of element a_2 in membrane h_2, etc.

More formally, the rules in the P system model presented above can be reformulated as follows:

$$(a_0, h_0) \to (a_1, h_1)^{\beta_1} (a_2, h_2)^{\beta_2} \dots (a_n, h_n)^{\beta_n}$$

Note that, for all $i \in \{1, \dots, n\}$, if $h_0 = h_i$ then, (a_i, h_i) is equivalent to the pair $(a_i, here)$. Otherwise, if $h_0 \neq h_i$ both membranes must be adjacent (one membrane is the father of the other one). If h_0 is the father of h_i, then the pair (a_i, h_i) is equivalent to (a_i, in_{h_i}). Finally, if h_i is the father of h_0, then the pair (a_i, h_i) is equivalent to (a_i, out). For each $i \in \{1, \dots, n\}$, β_i represents the multiplicity of (a_i, h_i) in the right-hand side (RHS) of the rule.

The second basic idea in the representation appears in [3] too. It consists on settling a total order in the set $\Gamma \times H$. Along the paper, in order to simplify the notation, given an alphabet Γ and a set of labels H, d will denote the cardinal $\Gamma \times H$. Let us consider a total order $\mathcal{O}$ on the set $\Gamma \times H$, $\mathcal{O} : \{1, \dots, d\} \to \Gamma \times H$. By using this order, we will represent $\Gamma \times H$ as the finite sequence $\langle \gamma_1, \dots, \gamma_d \rangle$, where γ_i is the i-th pair of $\Gamma \times H$ in the order $\mathcal{O}$.

By using this order, each rule

$$(a_0, h_0) \to (a_1, h_1)^{\beta_1} (a_2, h_2)^{\beta_2} \dots (a_n, h_n)^{\beta_n}$$

can be represented as

$$\gamma \to \gamma_1^{\alpha_1} \gamma_2^{\alpha_2} \dots \gamma_d^{\alpha_d}$$

where $(a_0, h_0) = \gamma$ and for all $i \in \{1, \dots, d\}$:

- If there exists $j \in \{1, \dots, n\}$ such that $\gamma_i = (a_j, h_j)$ then $\alpha_i = \beta_j$.
- Otherwise $\alpha_i = 0$.

We will say that $\gamma \to \gamma_1^{\alpha_1} \gamma_2^{\alpha_2} \dots \gamma_d^{\alpha_d}$ is the *pairwise* representation of the rule.

The use of an order on $\Gamma \times H$ leads us to a more homogeneous representation of rule $\gamma \to \gamma_1^{\alpha_1} \gamma_2^{\alpha_2} \dots \gamma_d^{\alpha_d}$. It can be represented by a pair $\langle \gamma, \vec{v} \rangle$ where γ (the LHS of the rule) belongs to $\Gamma \times H$, and $\vec{v}$ is a vector of dimension d whose arguments are in $\mathbb{N}$. Formally, we have the following definition:

Definition 1. *Let us consider a P system Π with Γ the alphabet and H the set of labels. Let $\Gamma \times H$ be the ordered set $\langle \gamma_1, \dots, \gamma_d \rangle$. The algebraic representation of the rule*

$$\gamma \to \gamma_1^{\alpha_1} \gamma_2^{\alpha_2} \dots \gamma_d^{\alpha_d}$$

is the pair $(\gamma, \vec{v})$ where $\vec{v} = (\alpha_1, \dots, \alpha_d)$. We will say that $\vec{v}$ represents the right-hand side of the rule r_i.

Remark 1: Given an order $\langle \gamma_1, \dots, \gamma_d \rangle$ on $\Gamma \times H$, a pair $\langle \gamma, \vec{v} \rangle$ where $\gamma \in \Gamma \times H$ and $\vec{v}$ is a vector of dimension d (with values in $\mathbb{N}$) defines a unique rule and vice-versa, each rule having a unique algebraic representation.

Remark 2: If the P system is not deterministic, then there exists at least one $\gamma \in \Gamma \times H$ such that there exists two different vectors $\vec{v}_1$ and $\vec{v}_2$ such that pairs $\langle \gamma, \vec{v}_1 \rangle$ and $\langle \gamma, \vec{v}_2 \rangle$ represent two different rules.

Let us see an example of this algebraic representation.

Example 1. Let us consider the skeleton of the P system considered in Section 2 with $\Gamma = \{a, b, c\}$, $H = \{e, s\}$, $\mu = [\,[\,]_e\,]_s$ and R the set of rules

Rule 1: $[\,a \to b^2 c\,]_e$	**Rule 4:** $[\,b \to a\,]_s$
Rule 2: $[\,a\,]_e \to a\,[\,]_e$	**Rule 5:** $a\,[\,]_e \to [\,c\,]_e$
Rule 3: $[\,b \to c^2\,]_s$	**Rule 6:** $[\,c \to a\,]_e$

The set of objects is $\Gamma = \{a, b, c\}$ and the set of labels is $H = \{e, s\}$. Let us consider the following total order in $\Gamma \times H$

$$\langle (a,e),(b,e),(c,e),(a,s),(b,s),(c,s)\rangle$$

The six rules of the P system can be settled as

r_1: $(a,e) \to (b,e)^2(c,e)$	r_4**:** $(b,s) \to (a,s)$
r_2**:** $(a,e) \to (a,s)$	r_5**:** $(a,s) \to (c,e)$
r_3**:** $(b,s) \to (c,s)^2$	r_6**:** $(c,e) \to (a,e)$

By using the previous total order in $\Gamma \times H$, these rules have the following algebraic representation

Rule 1: $\langle (a,e),(0,2,1,0,0,0)\rangle$	**Rule 4:** $\langle (b,s),(0,0,0,1,0,0)\rangle$
Rule 2: $\langle (a,e),(0,0,0,1,0,0)\rangle$	**Rule 5:** $\langle (a,s),(0,0,1,0,0,0)\rangle$
Rule 3: $\langle (b,s),(0,0,0,0,0,2)\rangle$	**Rule 6:** $\langle (c,e),(1,0,0,0,0,0)\rangle$

4.1 Configurations

A *configuration* of such a P system is the description of the multiset placed in the membranes of the P system in a given instant. Formally, given a P system with working alphabet Γ and set of labels H, a configuration C is a multiset over $\Gamma \times H$, $C : \Gamma \times H \to \mathbb{N}$, and we denote by $C(a,m)$ the multiplicity of object a in the membrane labeled by m of that configuration. The support of C, $supp(C)$, is defined as $supp(C) = \{(a,m) \in \Gamma \times H \,|\, C(a,m) \neq 0\}$ and, as usual in multisets theory, C will be represented as $\{(a,m)^{C(a,m)} \,|\, (a,m) \in supp(\mathcal{C})\}$. For example, the configuration of our example $[\,[\,b\,]_e\, c^3\,]_s$ can be represented as $\{(b,e),(c,s)^3\}$.

From the idea of setting an order on $\Gamma \times H$, the representation of a configuration via a vector is quite natural.

Definition 2. *Let us consider a P system Π with Γ the alphabet, H the set of labels and order $\langle \gamma_1,\ldots,\gamma_d\rangle$ on $\Gamma \times H$. An algebraic representation of a configuration $C : \Gamma \times H \to \mathbb{N}$ is a vector*

$$\vec{C} = (C(\gamma_1),\ldots,C(\gamma_d))$$

that is, the j-th element in $\vec{C}$ is a number representing the multiplicity of the j-th element of $\Gamma \times H$.

Let us remark that, if the order on $\Gamma \times H$ is set, then there exists a bijective correspondence between a configuration C and its algebraic representation $\vec{C}$.

Example 2. As we saw before, the initial configuration $[\,[\,b\,]_e\, c^3\,]_s$ can be expressed as the multiset $C = \{(b,e),(c,s)^3\}$. If we consider order

$$\langle (a,e),(b,e),(c,e),(a,s),(b,s),(c,s)\rangle$$

then the algebraic representation of the configuration is $\vec{C} = (0,1,0,0,0,3)$.

In order to formalize the concept of computation with this new representation, we will fix some notations. We denote by RHS_r the right-hand side of rule r and for all $\sigma \in \Gamma \times H$, $|RHS_r(\sigma)|$ denotes the multiplicity of σ in the multiset RHS_r.

Example 3. Let us consider the pairwise representation of the rule $r_1 : (a,e) \rightarrow (b,e)^2(c,e)$, then $RHS_{r_1} = (b,e)^2(c,2)$ and $|RHS_{r_1}|(b,e) = 2$.

Definition 3. *Let us consider an alphabet Γ, a set of labels H and the set of rules R of a P system. We will denote by $\mathcal{LHS}(R)$ the set of all the pairs from $\Gamma \times H$ that are the left-hand side of a rule from R. Formally*

$$\mathcal{LHS}(R) = \{\gamma \in \Gamma \times H \,|\, \exists r \in R \, (\gamma = LHS(r))\}$$

Example 4. Let us consider $\Gamma = \{a,b,c\}$, $H = \{e,s\}$ and R the set of rules

$$\begin{array}{lll} r_1\colon (a,e) \rightarrow (c,e)^2 & r_2\colon (a,e) \rightarrow (a,s) & r_3\colon (b,e) \rightarrow (c,e) \\ r_4\colon (a,s) \rightarrow (b,s) & r_5\colon (a,s) \rightarrow (b,s)(c,s)^2 & \end{array}$$

In this case $\mathcal{LHS}(R) = \{(a,e),(b,e),(a,s)\}$.

Definition 4. *Let us consider an alphabet Γ and a set of labels H of a P system Π and let $R = \langle r_1,\dots,r_p\rangle$ be an enumeration of its set of rules with $r_j = (LHS(r_j), \vec{v_j})$. Let $C : \Gamma \times H \rightarrow \mathbb{N}$ be a configuration of Π.*

A partition *of C with respect to R is a p-uple*

$$\mathcal{P} = \langle (r_1,k_1),\dots,(r_p,k_p)\rangle$$

such that for all $j \in \{1,\dots,p\}$, $k_j \geq 0$ and for all $\gamma \in \mathcal{LHS}(\mathcal{R})$

$$\sum_{LHS(r_j)=\gamma} k_j = C(\gamma)$$

Example 5. Let us consider an alphabet $\Gamma = \{a,b,c\}$ a set of labels $H = \{e,s\}$, $\mu = [\,[\,]_e\,]_s$ and R the set of rules from the example 4

$$\begin{array}{lll} r_1\colon (a,e) \rightarrow (c,e)^2 & r_2\colon (a,e) \rightarrow (a,s) & r_3\colon (b,e) \rightarrow (c,e) \\ r_4\colon (a,s) \rightarrow (b,s) & r_5\colon (a,s) \rightarrow (b,s)(c,s)^2 & \end{array}$$

Let us consider a configuration with algebraic representation $\vec{C} = \langle 3,0,1,7,4,1\rangle$ associated with order $\langle (a,e),(b,e),(c,e),(a,s),(b,s),(c,s)\rangle$ of $\Gamma \times H$. In this case, one possible partition of C with respect to R is

$$\mathcal{P} = \langle (r_1,2),(r_2,1),(r_3,0),(r_4,2),(r_5,5)\rangle$$

the number associated to each rule is a natural number and $\mathcal{LHS}(R) = \{(a,e),(b,e),(a,s)\}$, so in order to check that $\mathcal{P}$ is a partition it suffices to check

$$\begin{array}{l} \sum_{LHS(r_j)=(a,e)} k_j = k_1 + k_2 = 2+1 = 3 = C(a,e) \\ \sum_{LHS(r_j)=(b,e)} k_j = k_3 = 0 = C(b,e) \\ \sum_{LHS(r_j)=(a,s)} k_j = k_4 + k_5 = 2+5 = 7 = C(a,s) \end{array}$$

The different possible partitions capture the idea of different choice of rules in the case of non-deterministic P system. Notice that in the case of a deterministic P system, there exists only one partition

$$\mathcal{P} = \langle (r_1, C(LHS(r_1))), (r_2, C(LHS(r_2))), \ldots, (r_p, C(LHS(r_p))) \rangle$$

In order to obtain a new configuration C' from a given configuration C and from the set of rules $\{r_1, \ldots, r_p\}$, we need to describe the multiplicity of any $\sigma \in \Gamma \times H$ in C'. For the calculus of such multiplicity we need

- A partition $\mathcal{P} = \langle (r_1, k_1), \ldots, (r_p, k_p) \rangle$ of C with respect to R.
- The set $\mathcal{LHS}(R)$

In such multiplicity, each rule $r_i : \gamma_i \to RHS_{r_i}$ adds the multiplicity of σ in the right hand side of the rule multiplied by the value k_i in the partition $\mathcal{P}$. If the object is not consumed by any rule, we also add the multiplicity in the original configuration.

Formally, for every $\sigma \in \Gamma \times H$ we have:

$$C'(\sigma) = \begin{cases} \sum_{i=1}^{i=p} k_i \cdot |RHS_{r_i}(\sigma)| & \text{if } \sigma \in \mathcal{LHS}(R) \\ \sum_{i=1}^{i=p} k_i \cdot |RHS_{r_i}(\sigma)| + C(\sigma) & \text{if } \sigma \notin \mathcal{LHS}(R) \end{cases}$$

Example 6. Let us come back again to our P system Π with alphabet $\Gamma = \{a, b, c\}$, set of labels $H = \{e, s\}$, membrane structure $\mu = [\,[\,]_e\,]_s$ and the set of rules R

Rule 1: $[\,a \to b^2c\,]_e$	**Rule 4:** $[\,b \to a\,]_s$
Rule 2: $[\,a\,]_e \to a\,[\,]_e$	**Rule 5:** $a\,[\,]_e \to [\,c\,]_e$
Rule 3: $[\,b \to c^2\,]_s$	**Rule 6:** $[\,c \to a\,]_e$

Let us consider configuration $C_1 = [\,[\,bc^2\,]_e\,b^2c\,]_s$, i.e., $w_e = bc^2$ and $w_s = b^2c$. It is easy to check that by applying rules 4 and 6 we obtain configuration $C' = [\,[a^2b\,]_e\,a^2c\,]_s$. Such configuration can also be obtained by considering the multiplicity of each pair in $\Gamma \times H$ and using the previous formula. First we consider the partition $\mathcal{P} = \langle (r_1, 0), (r_2, 0), (r_3, 0), (r_4, 2), (r_5, 0), (r_6, 2) \rangle$ and $\mathcal{LHS}(R) = \{(a, e), (b, s), (a, s), (c, e)\}$. Then, for example,

$$C'(a, s) = k_1 \cdot 0 + k_2 \cdot 1 + k_3 \cdot 0 + k_4 \cdot 1 + k_5 \cdot 0 + k_6 \cdot 0 = 2 \cdot 1 = 2$$
$$C'(b, e) = k_1 \cdot 2 + k_2 \cdot 0 + k_3 \cdot 0 + k_4 \cdot 0 + k_5 \cdot 0 + k_6 \cdot 0 + C(b, e) = 0 \cdot 2 + 1 = 1$$

the remaining multiplicities in configuration C' can be obtained in a similar way.

5 Matrix Associated with the Skeleton

After defining the algebraic representation of rules and configurations, we will define a numerical matrix associated with the skeleton of a P system. The next definition of *extended* set of rules will be used in the definition of the matrix.

Definition 5. *Let Γ be the alphabet, H the set of labels and R the set of rules of a P system where R is a set of rules in its pairwise form. The extended set of rules of R in this skeleton, R^* is the set of rules R together with the identity rule $\gamma \rightarrow \gamma$ for all the $\gamma \in \Gamma \times H$ such that there is no rule in R with γ in its left-hand side.*

Considering identity rules, we obtain P systems whose computations never stop. In this paper, we are interested only in the evolution of computation in time and not in halting conditions. Let us remark two important considerations related with the extended set of rules:

- If R^* is the extended set of rules of R, then $\mathcal{LHS}(R^*) = \Gamma \times H$.
- Consequently, if C is a configuration of a P system Π with $\langle \gamma_1, \dots, \gamma_d \rangle$ an order on $\Gamma \times H$ and $\mathcal{P}^* = \langle (r_1, k_1), \dots, (r_p, k_p) \rangle$ is a partition of a configuration C of a P system with respect to its extended set of rules, then configuration C' that can be obtained from C in one computation step following such partition is $C'(\gamma_j) = \sum_{i=1}^{i=p} k_i \cdot |RHS_{r_i}(\gamma_j)|$ for all $j \in \{1, \dots, d\}$.

Example 7. Let us consider again the skeleton of example 1, and its set of rules,

$$\begin{array}{ll} r_1\text{: } (a,e) \rightarrow (b,e)^2(c,e) & r_4\textbf{: } (b,s) \rightarrow (a,s) \\ r_2\textbf{: } (a,e) \rightarrow (a,s) & r_5\textbf{: } (a,s) \rightarrow (c,e) \\ r_3\textbf{: } (b,s) \rightarrow (c,s)^2 & r_6\textbf{: } (c,e) \rightarrow (a,e) \end{array}$$

Note that the pairs γ from $\Gamma \times H$ such that there is no rule in R with γ as its left-hand side are (b,e) and (c,s), therefore to obtain R^* we have to add to R the rules

$$r_7\text{: } (b,e) \rightarrow (b,e) \qquad r_8\textbf{: } (c,s) \rightarrow (c,s)$$

Obviously, the set of rules R^* has also an algebraic representation

$$\begin{array}{ll} \textbf{Rule 1: } \langle (a,e), (0,2,1,0,0,0) \rangle & \textbf{Rule 5: } \langle (a,s), (0,0,1,0,0,0) \rangle \\ \textbf{Rule 2: } \langle (a,e), (0,0,0,1,0,0) \rangle & \textbf{Rule 6: } \langle (c,e), (1,0,0,0,0,0) \rangle \\ \textbf{Rule 3: } \langle (b,s), (0,0,0,0,0,2) \rangle & \textbf{Rule 7: } \langle (b,e), (0,1,0,0,0,0) \rangle \\ \textbf{Rule 4: } \langle (b,s), (0,0,0,1,0,0) \rangle & \textbf{Rule 8: } \langle (c,s), (0,0,0,0,0,1) \rangle \end{array}$$

With the help of the concept of extended set of rules, we define the matrix associated with a skeleton.

Definition 6. *Let us consider skeleton $Sk = (\Gamma, H, \mu, R)$ of a P system and let $\langle r_1, \dots, r_p \rangle$ be an enumeration of the extended set of rules R^* of R in its algebraic form. The matrix associated with skeleton Sk, M_{Sk} is the matrix whose rows are vectors $\vec{v_1}, \dots, \vec{v_p}$, where for each i with $1 \leq i \leq p$, $\vec{v_i}$ is the vector which represents the right-hand side of rule r_i.*

Before showing an example, some remarks are necessary.

- The matrix associated with a skeleton depends on the skeleton, as well as on the enumeration of the rules of the extended set and the order on $\Gamma \times H$. A different enumeration produces a different order in the rows of the matrix.
- In case of deterministic P systems, the number of rules in the extended set, p, and the number of pairs in $\Gamma \times H$, d are the same and we have a square matrix[3]. In general, M_{Sk} is a $d \times p$ matrix with $d \leq p$.

Example 8. If we consider the skeleton of example 7 and the enumeration of the eight rules of the extended set R^* and the usual order on $\Gamma \times H$, $\langle (a,e),(b,e),(c,e),(a,s),(b,s),(c,s)\rangle$

Rule 1: $\langle (a,e),(0,2,1,0,0,0)\rangle$ **Rule 5:** $\langle (a,s),(0,0,1,0,0,0)\rangle$
Rule 2: $\langle (a,e),(0,0,0,1,0,0)\rangle$ **Rule 6:** $\langle (c,e),(1,0,0,0,0,0)\rangle$
Rule 3: $\langle (b,s),(0,0,0,0,0,2)\rangle$ **Rule 7:** $\langle (b,e),(0,1,0,0,0,0)\rangle$
Rule 4: $\langle (b,s),(0,0,0,1,0,0)\rangle$ **Rule 8:** $\langle (c,s),(0,0,0,0,0,1)\rangle$

we have the following matrix

$$M_{Sk} = \begin{pmatrix} 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

6 Computing Backwards

The definition of these algebraic objects allows us to define an algebraic method to characterize the set of configurations $\mathcal{C}$ which can produce a given configuration C_0 in one computation step. First, we need to find the solutions of a system of linear equations.

Definition 7. *Let Π be a P system, $\langle r_1,\ldots,r_p\rangle$ a enumeration of its set of extended rules, M_{Sk} the matrix associated with the skeleton of Π based on that enumeration of R^* and let $\vec{C_0}$ be the vectorial representation of a configuration C_0. We will define the* solution set *of M_{Sk} and $\vec{C_0}$ and we will denote it by $SOL(M_{Sk},\vec{C_0})$ the set of real-valued vectors $\vec{x}$ with dimension p such that $\vec{C_0} = \vec{x}\cdot M_{Sk}$.*

Notice that according to the definition, $SOL(M_{Sk},\vec{C_0})$ can be the empty set. It is well known in Linear Algebra that if the range of the matrix M_{Sk} and the

[3] This kind of matrices were studied in [3].

range of the matrix M_{Sk} augmented with the vector of coefficients $\vec{C_0}$ is not the same, then the system of equations has no solution.[4]

$SOL(M_{Sk}, \vec{C_0})$ is a manifold of dimension p minus the range of the matrix M_{Sk} embedded in a vectorial space of dimension p, but the study of the algebraic properties of such manifold is out of the scope of this paper.

Example 9. Let us come back to our main example. If we take the matrix M_{Sk} from example 8, configuration $C' = [\,[\,a^2b\,]_e\,a^2c\,]_s$ from Section 2 and algebraic representation $\vec{C'} = (2,1,0,2,0,1)$, then in order to get $SOL(M_{Sk}, \vec{C'})$ we need to solve the system

$$(2,1,0,2,0,1) = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8) \begin{pmatrix} 0 & 2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

or equivalently,

$$\begin{aligned} x_6 &= 2 & x_2 + x_4 &= 2 \\ 2x_1 + x_7 &= 1 & 2x_3 + x_8 &= 1 \\ x_1 + x_5 &= 0 \end{aligned}$$

Then, $SOL(M_{Sk}, \vec{C'})$ is the following 3-dimensional manifold embedded in an 8-dimensional vectorial space

$$SOL(M_{Sk}, \vec{C'}) = \{(\alpha, \beta, \gamma, 2-\beta, -\alpha, 2, 1-2\alpha, 1-2\gamma) \,:\, \alpha, \beta, \gamma \in \mathbb{R}\,\}$$

Definition 8. *Let Π be a P system and an order $\langle \gamma_1, \ldots, \gamma_d \rangle$ on $\Gamma \times H$, $\langle r_1, \ldots, r_p \rangle$ a enumeration of its set of extended rules, M_{Sk} the matrix associated with the skeleton of Π based on that enumeration of R^* and let $\vec{C}$ be the vectorial representation of a configuration C. We define the* constructor mapping *as*

$$\psi_\Pi \,:\, SOL(M_{Sk}, \vec{C}) \to \mathbb{R}^d$$

such that for all $(x_1, \ldots, x_p) \in SOL(M_{Sk}, \vec{C'})$, $\psi_\Pi((x_1, \ldots, x_p)) = (y_1, \ldots, y_d)$ verifying for all $i \in \{1, \ldots, d\}$,

$$y_i = \sum_{\gamma_i = LHS(r_k)} x_k$$

[4] This result is called the Rouche-Frobenius theorem, especially in the Spanish speaking world. This is almost certainly because the Spanish mathematician Julio Rey Pastor referred to the theorem by this name.

Notice that the set $SOL(M_{Sk}, \vec{C})$ depends on the way in which the set of extended rules is enumerated, but $\psi_{\Pi}(SOL(M_{Sk}, \vec{C}))$ is independent of such enumeration. Obviously, if all the coordinates of $\vec{x} \in SOL(M_{Sk}, \vec{C'})$ are natural numbers, then all the coordinates of $\psi(\vec{x})$ are also natural numbers.

Example 10. Following with the set $SOL(M_{Sk}, \vec{C'})$ from Example 9 and order $\langle((a,e),(b,e),(c,e),(a,s),(b,s),(c,s)\rangle$ on $\Gamma \times H$, we have

$$\begin{aligned}
y_1 &= \textstyle\sum_{(a,e)=LHS(r_k)} x_k = x_1 + x_2 = \alpha + \beta \\
y_2 &= \textstyle\sum_{(b,e)=LHS(r_k)} x_k = x_7 = 1 - 2\alpha \\
y_3 &= \textstyle\sum_{(c,e)=LHS(r_k)} x_k = x_6 = 2 \\
y_4 &= \textstyle\sum_{(a,s)=LHS(r_k)} x_k = x_5 = -\alpha \\
y_5 &= \textstyle\sum_{(b,s)=LHS(r_k)} x_k = x_3 + x_4 = 2 + \gamma - \beta \\
y_6 &= \textstyle\sum_{(c,s)=LHS(r_k)} x_k = x_8 = 1 - 2\gamma
\end{aligned}$$

Therefore $\psi_{\Pi}(SOL(M_{Sk}, \vec{C}))$ is a 3-dimensional manifold embedded in an 6-dimensional vectorial space

$$\psi_{\Pi}(SOL(M_{Sk}, \vec{C})) = \{(\alpha + \beta, 1 - 2\alpha, 2, -\alpha, 2 + \gamma - \beta, 1 - 2\gamma) \mid \alpha, \beta, \gamma \in \mathbb{R}\}$$

Finally, we only consider the elements of $SOL(M_{Sk}, \vec{C})$ such that all its coordinates are natural numbers. We will prove below that the image of such vectors by means of the constructor mapping represent the searched configurations.

Definition 9. *Let Π be a P system, $\langle r_1, \ldots, r_p\rangle$ a enumeration of its set of extended rules, M_{Sk} the matrix associated with the skeleton of Π based on that enumeration of R^* and let $\vec{C}$ be the vectorial representation of a configuration C. We define*

- $NSOL(M_{Sk}, \vec{C})) = \{(x_1, \ldots, x_p) \in SOL(M_{Sk}, \vec{C})) \mid \forall i \in \{1, \ldots, p\}\, (x_i \in \mathbb{N})\}$
- *A constructed configurations C_1 of Π is a configuration such that $\vec{C_1} \in \psi_{\Pi}(NSOL(M_{Sk}, \vec{C}))$.*

Example 11. If we take $\psi_{\Pi}(SOL(M_{Sk}, \vec{C}))$ from Example 10

$$\psi_{\Pi}(NSOL(M_{Sk}, \vec{C})) = \left\{ (\alpha + \beta, 1{-}2\alpha, 2, -\alpha, 2 + \gamma{-}\beta, 1{-}2\gamma) \mid \begin{array}{l} \alpha, \beta, \gamma \in \mathbb{R} \\ \alpha + \beta \in \mathbb{N} \\ 1 - 2\alpha \in \mathbb{N} \\ -\alpha \in \mathbb{N} \\ 2 + \gamma - \beta \in \mathbb{N} \\ 1 - 2\gamma \in \mathbb{N} \end{array} \right\}$$

The set $\psi_{\Pi}(NSOL(M_{Sk}, \vec{C}))$ has only three elements

$$\vec{C_1} = (0, 1, 2, 0, 2, 1) \quad \vec{C_2} = (1, 1, 2, 0, 1, 1) \quad \vec{C_3} = (2, 1, 2, 0, 0, 1)$$

which correspond to the three configurations obtained in Section 2. Next we prove that the result holds in the general case.

Theorem 1. *Let Π be a P system with skeleton $Sk = (\Gamma, H, \mu, R)$ and let C be a configuration of Π. Let $\langle \gamma_1, \ldots, \gamma_d \rangle$ be an order on $\Gamma \times H$ and $\langle r_1, \ldots, r_p \rangle$ an enumeration of the extended set of rules R^* of R. Let M_{Sk} be the matrix associated with the skeleton Sk following such order and enumeration. Then, the configuration C_1 produces C in one computation step if and only if $\vec{C_1} \in \psi_\Pi(NSOL(M_{Sk}, \vec{C}))$.*

Proof. Let us consider a configuration C_1 such that $\vec{C_1} \in \psi_\Pi(NSOL(M_{Sk}, \vec{C}))$. Such configuration is a multiset C_1 on the set $\Gamma \times H$ such that for all $i \in \{1, \ldots, n\}$, $C_1(\gamma_i) \in \mathbb{N}$.

$\vec{C_1} \in \psi_\Pi(NSOL(M_{Sk}, \vec{C}))$ if and only if there exist $(x_1, \ldots, x_p) \in SOL(M_{Sk}, \vec{C})$ with $x_i \in \mathbb{N}$ for all $i \in \{1, \ldots, p\}$ such that $\psi_\Pi(x_1, \ldots, x_n) = (C_1(\gamma_1), \ldots, C_1(\gamma_d))$. By definition of the constructor mapping $\psi_\Pi : SOL(M_{Sk}, \vec{C}) \to \mathbb{R}^d$ we have for all $i \in \{1, \ldots, d\}$,

$$C_1(\gamma_i) = \sum_{\gamma_i = LHS(r_k)} x_k$$

On the other hand, we also know that $(x_1, \ldots, x_p) \in SOL(M_{Sk}, \vec{C})$, i.e.,

$$(C(\gamma_1), \ldots, C(\gamma_d)) = (x_1, \ldots, x_d) \cdot M_{Sk}$$

By construction of the matrix M_{Sk}, the previous equality means that for all $i \in \{1, \ldots, n\}$,

$$C(\gamma_i) = \sum_{j=1}^{p} x_j \cdot |RHS_{r_j}(\gamma_i)|$$

To sum up, $\vec{C_1} \in \psi_\Pi(NSOL(M_{Sk}, \vec{C}))$ if and only if there exist $(x_1, \ldots, x_p)$ such that for all $i \in \{1, \ldots, p\}$

(a) $x_i \in \mathbb{N}$
(b) $C_1(\gamma_i) = \sum_{\gamma_i = LHS(r_k)} x_k$
(c) $C(\gamma_i) = \sum_{j=1}^{p} x_j \cdot |RHS_{r_j}(\gamma_i)|$

Since R^* is a set of extended rules, $\mathcal{LHS}(R^*)$ is the set $\Gamma \times H$. Bearing this equality in mind, properties (a) and (b) claim that $\mathcal{P}^* = \langle (r_1, x_1), \ldots, (r_p, x_p) \rangle$ is a partition of C_1 with respect to R^* and property (c) claims that the configuration C can be obtained from C_1 by using the partition $\mathcal{P}^*$.

On the other hand, if C_1 produces C in one computation step, then there exist a vector $(x_1, \ldots, x_n)$ such that $\langle (r_1, x_1), \ldots, (r_p, x_p) \rangle$ is a partition of C_1 with respect to R^* verifying properties (a), (b) and (c) and therefore $\vec{C_1} \in \psi_\Pi(NSOL(M_{Sk}, \vec{C}))$.

7 A General Method

After the proof of Theorem 1, we come back to the questions asked at the end of Section 2. We wondered if there exists an algorithm such that it takes a P system

Π as input and it outputs a mapping $\mathcal{R}_\Pi$ which, for *every* configuration C' of Π, $\mathcal{R}_\Pi(C')$ is the set of all computations C such that C' is obtained from C in one computational step. A method for computing such algorithm is the following:

Given a P system Π with skeleton $Sk = (\Gamma, H, \mu, R)$,

1. Fix an order $\langle \gamma_1, \dots, \gamma_d \rangle$ for $\Gamma \times H$.
2. Consider the pairwise representation of the rules in R according to such order.
3. Consider the extended set of rules R^* from R and fix an enumeration $\langle r_1, \dots, r_p \rangle$ of the rules from R^* in its algebraic representation.
4. Define matrix M_{Sk} following the orders $\langle \gamma_1, \dots, \gamma_d \rangle$ and $\langle r_1, \dots, r_p \rangle$.

Matrix M_{Sk} is the same for all configurations. Next we provide a method for finding all the configurations C' such that C' produce a given configuration C in one computation step.

Given a configuration C of Π

1. Obtain the algebraic representation $\vec{C}$ of C according to the order $\langle \gamma_1, \dots, \gamma_d \rangle$.
2. Find all the vectors $\vec{x}$ with natural coordinates such that $\vec{C} = \vec{x} \cdot M_{Sk}$. The set of all these vectors is called $NSOL(M_{Sk}, \vec{C})$.
3. For each $\vec{x} \in NSOL(M_{Sk}, \vec{C})$, we consider $C_{\vec{x}} = (y_1 \dots, y_d)$ where, for all $i \in \{1, \dots, n\}$
$$y_i = \sum_{\gamma_i = LHS(r_k)} x_k$$
4. The set $\{C_{\vec{x}} \mid \vec{x} \in NSOL(M_{Sk}, \vec{C})\}$ is the set of the algebraic representations of all the configurations such that produce C in one computation step.

8 Conclusions and Future Work

In this paper, we provide a general method for finding all the configurations that produce a given one in one computational step. For that purpose, we have used an algebraic representation of rules and configurations and a matrix associated with the skeleton of the P systems.

The key step of the algorithm is to find all the vectors of natural numbers that are solutions of a system of linear equations. In such a system, the number of equations is the number of objects in the alphabet multiplied by the number of labels. The number of variables in the system is the cardinal of the set of extended rules which is at least the same as the number of equations and has no upper bound.

The problem of finding the solutions with natural values of a system of linear equations is a heavy problem, specially if we consider a high number of variables and equations (which is the usual case for P systems). Nonetheless, currently there exist some powerful software tools able to deal with large numerical matrices and solve the corresponding systems under the restriction of finding natural-valued vectors.

In this way, we hope that this method can be useful for researchers interested in computing backwards in Membrane Computing, since it can consider the problem of finding the previous configurations as a problem of Integer Programming.

Finally, this work can be extended in several ways. Not only by going deeper in the concept of computing backwards along a computation (and not only in one step) but exploring if these ideas can be extended to other P system models.

Acknowledgment

The authors acknowledge the support of the project TIN2006-13425 of the Ministerio de Educación y Ciencia of Spain, cofinanced by FEDER funds, and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200.

References

1. O. Agrigoroaiei, G. Ciobanu: Dual P Systems. *Proceedings of Ninth Workshop on Membrane Computing* (P. Frisco, D. Corne, Gh. Păun, eds.), Technical report HW-MACS-TR-0061, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK, July 2008, 45–58.
2. A. Cordón-Franco, M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez, A. Riscos-Núñez: Exploring Computation Trees Associated with P Systems. In *Membrane Computing*, LNCS 3365, Springer, 2005, 278–286.
3. M.A. Gutiérrez-Naranjo, M.J. Pérez-Jiménez: Efficient Computation in Rational-Valued P Systems. Submitted, 2009.
4. Gh. Păun: *Membrane Computing. An Introduction*. Springer, Berlin, 2002.

Notch Signalling and Cellular Fate Choices: A Short Review

Beverley M. Henley

University of Seville, Spain
bhenley@us.es

Summary. During mammalian central nervous system (CNS) development, an enormous variety of cell types are generated. This cell diversity is due in part to asymmetrical cell division. Asymmetrical segregation of Numb, a cell-determinant protein, can result in the differential activation of the Notch pathway. The Notch pathway defines one of the few fundamental signalling pathways that govern metazoan development. Notch signals link the fate decisions of one cell to those of its neighbours. Notch activation has a profound effect on many aspects of nervous system development. Here we present a brief overview of Notch signalling and reiterate some relevant questions relating to the Notch pathway.

1 Cell Diversity

The mammalian central nervous system (CNS) contains an enormous variety of cell types each with a unique morphology, physiology and function (Lein et al., 2006). Understanding how neuroepithelial cells (stem cells) of the developing CNS choose between alternative cell fates to generate cell diversity is a challenge (Cayouette and Raff, 2002). During development, cell-fate diversity is brought about, in part, by asymmetric cell divisions (Gho and Schweisguth, 1998). Asymmetric segregation of cell determinants, such as Numb can result in the differential activation of the Notch pathway, which can generate cell diversity (Fichelson and Gho, 2004). In invertebrates, asymmetric segregation of cell-fate determining proteins or mRNAs to the two daughter cells during precursor cell division plays a crucial part in cell diversification. There is increasing evidence that this mechanism also operates in vertebrate neural development and that the Numb protein, which function as cell-fate determinant during Drosophila development, may also function in this way during vertebrate development (Cayouette and Raff, 2002). A very clear illustration of symmetric and asymmetric segregation of a cell fate-determining protein can be found in Figure 2 of Cayouette and Raff (2002).

2 Notch Pathway

Cells that receive Numb antagonize Notch activity. Those cells that do not receive numb will adopt the fate associated with Notch activation. Despite the complexity of the action of Notch, some general principles underlying the action of this fundamental cell-interaction mechanism have become known. During development, animals use Notch signalling to amplify molecular differences between neighbouring cells. The implementation of a particular developmental program modulated by Notch depends, however, on how Notch integrates its activity with other cellular factors (Artavanis-Tsakonas et al., 1999), and is dependent on cellular context (Bray, 2006). The Notch cell interaction mechanism defines one of the few fundamental signalling pathways that govern metazoan development. Notch signals link the fate decisions of one cell to those of its neighbours and have been shown to have a profound effect on many aspects of nervous system development (Louvi and Artavanis-Tsakonas, 2006). Delta-Notch signalling is involved in cell fate decisions in developing vertebrates.

3 Notch Activation

Notch, a transmembrane protein, is activated by the ligands Delta and Jagged, which are also transmembrane proteins expressed by neighbouring cells. Notch activation is described in detail in Kageyama et al. (2005), which we now summarise. Upon activation, the intracellular domain of Notch is transferred into the nucleus and forms a complex with RBP-J. This complex induces Hes1 and Hes5 expression. Hes1 and Hes5 inhibit the expression of activator-type bHLH. In a differentiating neuron, Notch is not activated, and RBP-J represses Hes1 and Hes5 expression. The activator-type bHLH genes are expressed. The activator-type bHLH factors induce expression of Hes6, which inhibits Hes1 functions and reinforces the neurogenic process. The activator-type bHLH factors also induce neuronal specific genes. One of them includes the Notch ligand Delta, which activates Notch signalling in neighbouring cells. An excellent illustration of Delta-Notch signalling is found in Kageyama et al. (2005): Figure 1, entitled "Regulation of neural development by the repressor-type and activator-type bHLH genes". Put simply, Delta ligands bind and activate Notch receptors on a neighbouring cell, this Delta-Notch signalling can dictate the fate of the neighbouring cells (by influencing intracellular gene expression).

4 Notch Pathway: Questions

Notch signals affect specific cell fates in a context-specific manner, a schematic summarising the effects of Notch signalling and its affect on cell fate decisions can be found in Louvi and Artavanis-Tsakonas (2006), Figure 3. Understanding how and why different target genes are activated according to cell type and time is a very important question, in other words: how and why is Notch activation context dependant (Bray, 2006)? This and other important questions are posed in Bray (2006). The response to Notch differs greatly between cell types, for example Notch promotes cell proliferation in some contexts and apoptosis in others. What is the reason for this? Bray also states that recent data reveals that the precise location of the Notch ligand and the receptor in the cell can have profound effects on signalling. How does the different ligand locations exactly impact on Notch activity? All of these questions are extremely important in untangling the role of Notch during diverse developmental and physiological processes.

5 Modelling

Modelling biological processes can be a fruitful undertaking (Fussengger et al., 2000). As stated in Fisher and Henzinger (2007) computational and mathematical modelling of biological systems is becoming increasingly important in efforts to better understand complex biological behaviours. Over the years, diagrammatic models have been used to summarise and understand experimental data. Despite the many benefits of such models, as well as their simplicity, they give a stationary picture of cellular processes. Therefore, translating these models into more dynamic forms may be very useful (Fisher and Henzinger, 2007). Fussengger et al. (2000) believe “The model represents a flexible yet rigorous method to store, visualise and interact with current and newly emerging biological information.” A modelling approach that permits rigorous tests of mutual consistency between experimental data and mechanistic hypotheses and can identify specific conflicting results can provide a useful tool to developmental biology (Kam et al., 2008).

The Notch pathway is a fundamental pathway in metazoan development and the design and implementation of a good dynamic model of this pathway, and of crosstalk between Notch and other signalling pathways, may be beneficial to developmental biologists.

Acknowledgements

The author acknowledges support from the Project of Excellence from the Junta de Andalucía, grant TIC 581. A sincere thanks to Mario J. Pérez–Jiménez.

References

[1] S. Artavanis-Tsakonas, M.D. Rand, R.J. Lake: Notch signaling: Cell fate control and signal integration in development. *Science*, 284, 5415 (1999), 770–776.

[2] S.J. Bray: Notch signalling: a simple pathway becomes complex. *Nature Reviews Molecular Cell Biology*, 7, 9 (2006), 678–689.

[3] M. Cayouette, M. Raff: Asymmetric segregation of numb: a mechanism for neural specification from Drosophila to mammals. *Nat. Neurosci.*, 5, 12 (2002), 1265–1269.

[4] P. Fichelson, M. Gho: Mother-daughter precursor cell fate transformation after Cdc2 down-regulation in the Drosophila bristle lineage. *Dev. Biol.*, 276, 2 (2004), 367–377.

[5] J. Fisher, T. Henzinger: Executable cell biology. *Nature Biotechnology*, 25, 11 (2007), 1239–1249.

[6] M. Fussenegger, J. E. Bailey, J. Varner: A mathematical model of caspase function in apoptosis. *Nature Biotechnology*, 18 (2000), 768–774.

[7] M. Gho, F. Schweisguth: Frizzled signalling controls orientation of asymmetric sense organ precursor cell divisions in Drosophila. *Nature*, 393 (1998), 178–181.

[8] R. Kageyama, T. Ohtsuka, J. Hatakeyama, R. Ohsawa: Roles of bhlh genes in neural stem cell differentiation. *Experimental Cell Research*, 306, 2 (2005), 343–348.

[9] N. Kam, H. Kugler, R. Marelly, L. Appleby, J. Fisher, A. Pnueli, D. Harel, M. Stern, E. Hubbard: A scenario-based approach to modeling development: A prototype model of C. elegans vulval fate specification. *Dev. Biol.*, 323, 1 (2008), 1–5.

[10] E.S. Lein et al.: Genome-wide atlas of gene expression in the adult mouse brain. *Nature*, 445 (2006), 168–176.

[11] A. Louvi, S. Artavanis-Tsakonas: Notch signalling in vertebrate neural development. *Nature Reviews Neuroscience*, 7 (2006), 93–102.

Mutation Based Testing of P Systems

Florentin Ipate[1], Marian Gheorghe[2]

[1] Department of Computer Science
Faculty of Mathematics and Computer Science
The University of Pitesti
Str Targu din Vale 1, 110040 Pitesti
`florentin.ipate@ifsoft.ro`

[2] Department of Computer Science, The University of Sheffield
Regent Court, Portobello Street, Sheffield S1 4DP, UK
`M.Gheorghe@dcs.shef.ac.uk`

Summary. 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. 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. 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, thus making a significant progress in the area of P system testing.

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 behavior 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 – a comprehensive bibliography of P systems can be found at [16]. The most investigated membrane computing topics are related to the computational power of different variants, their capabilities to solve hard problems, like NP-complete ones, decidability, complexity aspects and hierarchies of classes of languages produced by these devices. In the last years there have also been significant developments in using the P systems paradigm to model, simulate and formally verify various systems [2]. Suitable classes of P systems have been associated with some of these applications and software packages have been developed. Of the many variants of P systems that have been defined, in this paper we consider cell-like P systems without priority rules 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 behavior 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 may be based on a formal specification or model, in which case the process could be automated. There is a large class of formal models used in software specification: finite state machines, Petri nets, process algebras, Z, VDM etc. We can add now P systems as a formal approach [2] to specifying various applications in linguistics, graphics and, more recently, biology, especially for defining signalling pathways.

A number of recent papers devise testing strategies 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 $(x \geq 0)\&\&a$ then
  $y = y + 1$
else
  $y = y + 2$
```

Then mutants for this code fragment can be obtained by either

- substituting && with another logic operator, e.g., $||$;
- substituting $\geq$ with another comparison operator, e.g., $>$, $=$;
- substituting $+$ with another arithmetic operators, e.g., $-$;
- 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 $(x \geq 0)||a$ then
  $y = y + 1$
else
  $y = y + 2$
```

```
if $(x > 0)\&\&a$ then
  $y = y + 1$
else
  $y = y + 2$

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

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

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

if $(x \geq 0)\&\&a$ then
  $y = y + 1$
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 specialized 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 (and hence produce an incorrect output). Obviously, the generation of a weak mutation test suite is less complex; on the other hand, a strong mutation test suite is easier to apply as only the program outputs need to be measured.

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 formalization 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. Based on it, we formally describe the process of generating the mutants for the given tree. Furthermore, the paper shows how the proposed method can be used to construct mutants for a P system specification.

2 Preliminaries

Before proceeding we introduce the notation to be used in this paper. For an alphabet $V = \{a_1, \ldots, a_p\}$, 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 $(|u|_{a_1}, \ldots, |u|_{a_p})$. This is denoted by $\Psi_V(u)$.

2.1 Context-free grammars

In this section basic concepts and results related to context-free grammars are introduced. For more details on automata, grammars and languages we refer to a classical textbook [6]. A context-free grammar is a system $G = (V, T, P, S)$, where

- V is the set of variables (nonterminals);
- T is the set of terminals;
- P is the set of production rules of the form $A \to w$; where A is a single nonterminal symbol, and w is a string of terminals and/or nonterminals;
- S is the start symbol.

For any strings $u, v \in (V \cup T)^*$, we write $u \Longrightarrow v$ if there exists a production rule $A \to w$ and $\alpha, \beta \in (V \cup T)^*$ such that $u = \alpha A \beta$ and $v = \alpha w \beta$. That is, v is the result of applying the rule $A \to w$ to u.

The $\Longrightarrow$ relation can be extended to a sequence of zero or more production rules: for any $u, v \in (V \cup T)^*$, we write $u \Longrightarrow^* v$ if there exist $u_1, \cdots, u_k$, $k \geq 1$ such that $u = u_1$, $v = u_k$ and $u_i \Longrightarrow u_{i+1}$, $1 \leq i \leq k-1$. We say that u *derives* v. If the derivation has at least one step (i.e., $k > 1$) then we denote $u \Longrightarrow^+ v$.

The *language* described by the context-free grammar G is the set $L(G) = \{v \in T^* \mid S \Longrightarrow^* v\}$. A language $L \subseteq T^*$ is said to be context-free if there is a context-free grammar G such that $L = L(G)$.

A context-free grammar is said to be proper if

- it has no useless symbols (inaccessible symbols or unproductive symbols), i.e., $\forall A \in V$, $\exists \alpha, \beta \in (V \cup T)^*, v \in T^*$ such that $S \Longrightarrow^* \alpha A \beta$ and $A \Longrightarrow^* v$;
- it has no λ-productions, i.e., $A \to \lambda$;
- it has no renaming production rules, i.e., $A \to B$, for $A, B \in V$.

For every context-free language L, if $\lambda \notin L$ then there exists a proper context-free grammar that describes L. For simplicity, in the sequel we consider only proper context-free grammars.

A derivation (parse) tree for a (proper) context-free grammar $G = (V, T, P, S)$ is a tree that satisfies the following conditions:

- each non-leaf node is labeled by a nonterminal in V;
- each leaf node is labeled by a terminal in T;
- if a non-leaf node is labeled A and its children are labeled $X_1, \ldots, X_k$ then $A \rightarrow X_1 \ldots X_k$ is a production rule of G.

If the root node is labeled by S then the yield of the tree is the string of terminals obtained by concatenating the leaves from left to right. For any string of terminals $w \in T^*$, $S \Longrightarrow^* w$ if and only if w is the yield of some derivation (parse) tree with root S [6]. Consequently, $w \in L(G)$ if and only if w is generated by some parse tree with root S. Parse trees have very high practical value as they are used by compilers to represent the structure of the source code.

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. Usually, ambiguity is a feature of the grammar, not of the language and unambiguous grammars can be found to describe the same context-free language. However, there are certain context-free languages which can only be generated by ambiguous grammars; such languages are called *inherently ambiguous*. An ambiguous grammar presents a practical problem since a string may be associated with more than one parse tree. However, there are well-known techniques for eliminating the causes of ambiguity which are used in compiler construction.

In the sequel we will consider (possibly ambiguous) context-free grammars which describe languages that are not inherently ambiguous. We will tacitly assume that mechanisms for solving the causes of ambiguity exist and so there is a one-to-one mapping between a string and its parse tree.

2.2 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 system, 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* $\Pi = (V, \mu, w_1, \ldots, w_n, R_1, \ldots, R_n)$*, where*

- V *is a finite set, called* alphabet*;*

- *μ defines the membrane structure; 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, each with the label of the membrane it points to; μ also describes the position of each membrane in the hierarchy. For instance, a structure of three membranes in which membrane 1 contains membranes 2 and 3 can be described by either $[_1[_2]_2[_3]_3]_1$ or $[_1[_3]_3[_2]_2]_1$. The rules in each region have the form $u \rightarrow (a_1, t_1) \ldots (a_m, t_m)$, 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, the symbol a 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 $c = (u_1, \ldots, u_n)$, 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; $c = (u_1, \ldots, u_n)$ 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, in this paper we will only focus on P systems with only one membrane.

2.3 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 total, that is, for every state $s \in S$ there is a state $s' \in S$ such that $(s, s') \in H$;*
- *$L : S \rightarrow 2^{AP}$ 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 $AP = \{(val_i = val) \mid 1 \leq i \leq n, val \in Val_i\}$. 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 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.

Definition 3. *An infinite path in a Kripke structure $M = (S, H, I, L)$ from a state $s \in S$ is an infinite sequence of states $\pi = s_0 s_1 \ldots$, such that $s_0 = s$ and $(s_i, s_{i+1}) \in H$ for every $i \geq 0$. A finite path π is a finite prefix of an infinite path.*

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 $G = (V, T, P, S)$ and $L(G)$ the language defined by G. We assume that, for every production rule p of G of the form $A \to X_1 \ldots X_k$, 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 $A \to X'_1 \ldots X'_n$ such that each symbol $X'_1, \ldots, X'_n$ is either a terminal or is found among $X_1, \ldots, X_k$. Furthermore, p' is either a production rule of G itself or has the form $A \to A$, $A \in V$; 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 $A \to X'_1 \ldots X'_k$ 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 $A \to w$ where w is obtained by inserting one terminal into the string $X_1 \ldots X_k$ (at any position).
- A *string deletion mutant* is a production rule of the form $A \to w$ where w is obtained by removing one or more symbols from $X_1 \ldots X_k$.
- A *string reordering mutant* is a production rule of the form $A \to w$ where w is obtained by reordering the string $X_1 \ldots X_k$.

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 $Tr_1, \ldots, Tr_k$ having root nodes $X_1, \ldots, X_k$, respectively and $p \in P$ the corresponding production rule of G, of the form $A \to X_1 \ldots X_k$. This is denoted by $Tr = MakeTree(A, Tr_1, \ldots, Tr_k)$. Let Tr' denote a mutant of Tr. Then either

– (**subtree mutation**) $Tr' = MakeTree(A, Tr'_1, \ldots, Tr'_k)$, 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**) $Tr' = tree(A, Tr'_1, \ldots, Tr'_n)$, where there exists a mutant p' of p of the form $A \rightarrow X'_1 \ldots X'_n$ such that for every i, $1 \leq i \leq n$, there exists j, $1 \leq j \leq n$, such that $Tr'_i = Tr_j$.

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 1. *Every mutant of a parse tree for 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.

Example 1. Let $G = (V, T, P, S)$ where

- $V = \{S\}$;
- $T = \{0, \ldots, N\} \cup \{+, -\}$ where N is a fixed upper bound;
- $P = \{p_1, p_2\} \cup \{p_3^i \mid 0 \leq i \leq N\}$, where $p_1 : S \rightarrow S + S$, $p_2 : S \rightarrow S - S$, $p_3^i : S \rightarrow i$, $0 \leq i \leq N$.

Suppose we have the following rule mutants:

- Mutants for $p_1 : S \rightarrow S - S$ (terminal replacement), $S \rightarrow S$ (string deletion)
- Mutants for $p_2 : S \rightarrow S + S$ (terminal replacement), $S \rightarrow S$ (string deletion)
- Mutants for p_3^i : $S \rightarrow i - 1$ and $S \rightarrow i + 1$ if $1 < i < N$, $S \rightarrow 1$ if $i = 0$ and $S \rightarrow N - 1$ 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 neighboring 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.

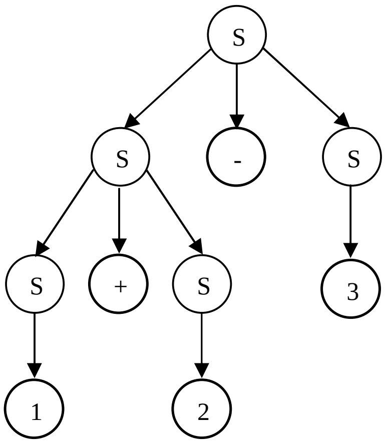


Fig. 1. Example parse tree

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 $u_i \to v_i$, 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 $\Psi_V(u) \in \mathbf{N}^k$, where k denotes the number of symbols in V.

In order to keep the number of configuration 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 $\mathbf{N}^k_{Max} = \{u \in \mathbf{N}^k \mid u \leq Max\}$. 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: $MaxParal(u, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$, $u \in \mathbf{N}^k_{Max}$, $n_1, \ldots, n_m \in \mathbf{N}$ signifies that a derivation of the configuration u in maximally parallel mode is obtained by applying rules $r_1 : u_1 \to v_1, \ldots, r_m : u_m \to v_m$ for $n_1, \ldots, n_m$ times, respectively; $Apply(u, v, u_1, v_1, n_1, \ldots, u_m, v_m, n_m)$, $u \in$

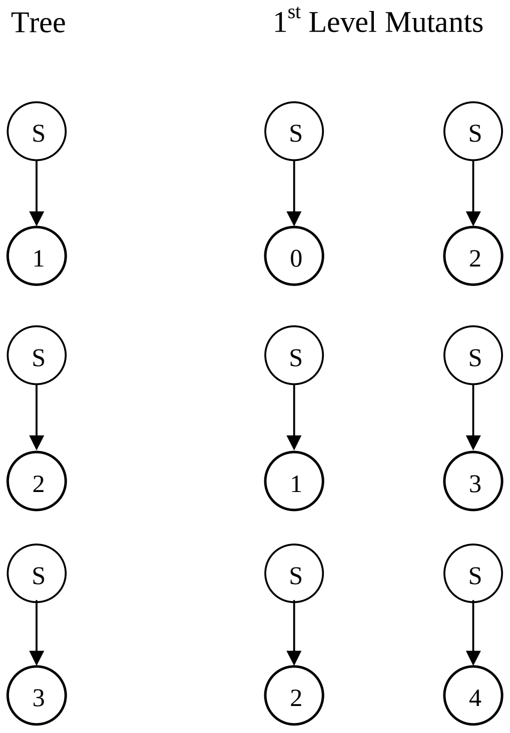


Fig. 2. 1st level mutants

$\mathbf{N}^k_{Max}$, $n_1, \dots, n_m \in \mathbf{N}$, denotes that v is the result of applying rules $r_1, \dots, r_m$ for $n_1, \dots, n_m$ 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 $Halt, Crash \notin \mathbf{N}^k_{Max}$, $Halt \neq Crash$;
- $I = w$;
- H is defined by:
  - $(u, v) \in H$, $u, v \in \mathbf{N}^k_{Max}$, if $\exists n_1, \dots, n_m \in \mathbf{N} \cdot MaxParal(u, u_1, v_1, n_1, \dots, u_m, v_m, n_m) \wedge Apply(u, v, u_1, v_1, n_1, \dots, u_m, c_m, n_m)$;
  - $(u, Halt) \in H$, $u \in \mathbf{N}^k_{Max}$, if $\neg \exists v \in \mathbf{N}^k_{Max}, n_1, \dots, n_m \in \mathbf{N} \cdot Apply(u, v, u_1, v_1, n_1, \dots, u_m, v_m, n_m)$;
  - $(u, Crash) \in H$ if $\neg \exists v \in \mathbf{N}^k_{Max} \cup \{Halt\} \cdot (u, v) \in H$;
  - $(Halt, Halt) \in H$;
  - $(Crash, Crash) \in H$.

It can be observed that the relation H is total.

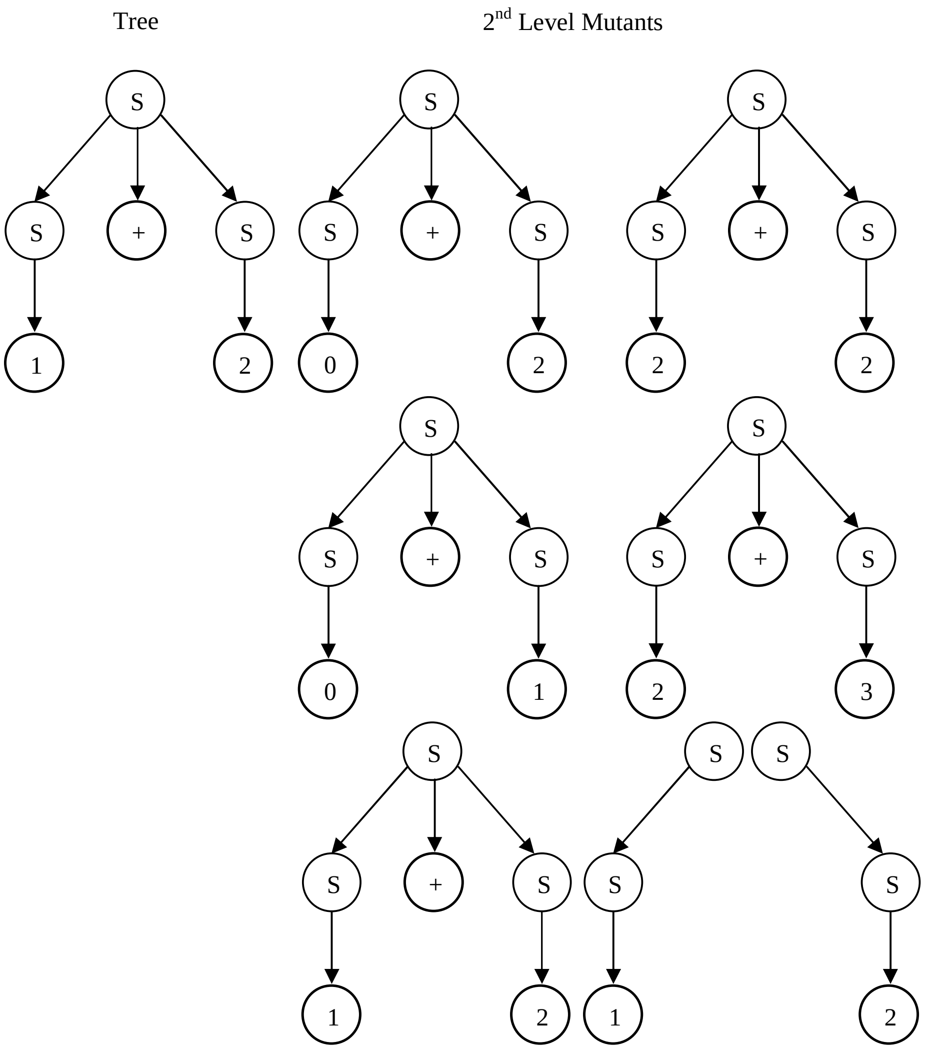


Fig. 3. 2nd level mutants

A multi-membrane P system without dissolving rules can be collapsed into a 1-membrane P system so, without loss of generality we will only consider 1-membrane P systems.

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.

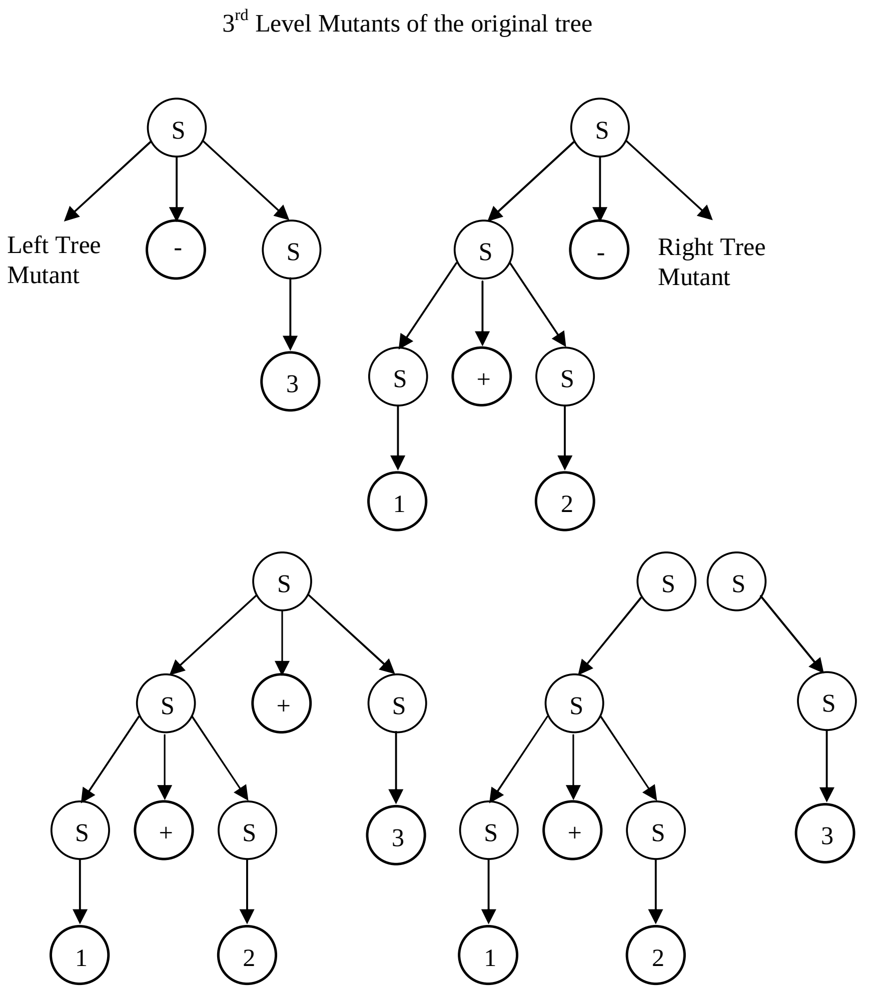


Fig. 4. 3rd level mutants

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 at lower level of testing. On the other hand, it is possible to incorporate the definitions of the two predicates 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. Naturally, for complexity reasons, in practice a multi-phase testing strategy is normally preferred.

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

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

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

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

Suppose the following rule mutants are defined:

- Mutants for $p_1 : S \rightarrow S$;
- Mutants for $p_2 : S \rightarrow S \vee S,\ S \rightarrow S$;
- Mutants for $p_3 : S \rightarrow S \wedge S,\ S \rightarrow S$;
- Mutants for $p_4 : S \rightarrow \neg true$;
- Mutants for $p_5 : S \rightarrow \forall n_1 \cdot S_1$;
- Mutants for $p_6 : S_1 \rightarrow \forall n_2 \cdot S_2$;
- Mutants for $p_7 : S_1 \rightarrow S \vee S_1,\ S_1 \rightarrow S_1$;

- Mutants for p_8 : negate de predicate, change parameters such that the obtained formula is syntactically correct, e.g. switch u and u_1;
- p_9 : negate de predicate, change parameters such that the obtained formula is syntactically correct;
- remaining rules: change each integer value by adding or removing 1.

Now, consider the P system Π with rules $r_1 : a \to ab$, $r_2 : a \to c$. Among the P-system mutants generated using the above procedure are the following:

- P systems in which one rules is changed: r_1 can be replaced by any of $\lambda \to ab$ (this is an invalid rule and the resulting mutant will have no P system equivalent), $aa \to ab$, $ab \to ab$, $ac \to ab$, $a \to a$, $a \to b$, $a \to c$, $a \to a^2b$, $a \to ab^2$, $a \to abc$; r_2 can be substituted in a similar manner. Note that only one rule is mutated at a time.
- r_1 and r_2 are interchanged (this will result in an equivalent mutant).
- A system with rules r_1 and r_2, but which are not applied in a maximal parallel mode (this is obtained by negating the *maxParal* predicate in the expression of H).
- Other erroneous Kripke systems which may have no P system equivalent; these can be obtained, for example, by negating the *Apply* predicate in the expression of H or by changing one of its "state" parameters (u or v).

Note that in this example we have used a grammar that generates 1-membrane P systems with (at most) two rules and so all generated mutants have this form. More generally, we can use a grammar that describes any 1-membrane P system (with any number of rules). Naturally, in this case the mutant generation process will be much more complex.

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 based rigorously defined operations with rules and subtrees of derivation trees is introduced for context-free grammar formalisms and extended to P systems. Some examples illustrate the approach.

Acknowledgment

Work supported by the CNCSIS grant IDEI 643/2009 (EvoMT).

References

1. F. Bernardini, M. Gheorghe, F.J. Romero-Campero, N. Walkinshaw: A hybrid approach to modelling biological systems. *Workshop on Membrane Computing 2007*, LNCS 4860, 138–159.
2. G. Ciobanu, Gh. Păun, M. J. Pérez-Jiménez, eds.: *Applications of 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*, 11, 3 (2006), 279–298.
4. M. Gheorghe, F. Ipate: On testing P systems. *Workshop on Membrane Computing, 2008*, LNCS 5391, 204–216.
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*, 227 (2009), 113–126.
8. F. Ipate, M. Gheorghe: Finite state based testing of P systems. *Natural Computing*, 2009, to appear.
9. J. Offutt: A practical system for mutation testing: Help for the common programmer. *International Test Conference*, 1994, 824–830.
10. Y.-S. Ma, J. Offutt, Y.R. Kwon: MuJava – An automated class mutation system. *Software Testing, Verification and Reliability*, 15, 2 (2005), 97–133.
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*, 61, 1 (2000), 108–143.
13. Gh. Păun: *Membrane Computing. An Introduction.* Springer, 2002.
14. http://en.wikipedia.org/wiki/Mutation_testing
15. http://cs.gmu.edu/ offutt/mujava/
16. http://ppage.psystems.eu

Author Index